ZeroBlock: Preventing Selfish Mining in Bitcoin

Siamak Solat, Maria Potop-Butucaru

ZeroBlock: Preventing Selfish Mining 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 nodes as a distributed shared register. New blocks in the Blockchain contain the last transactions in the system and are added by nodes (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* nodes in order to either earn undue rewards or waste the computational power of *honest* nodes. Contrary to recent solutions, our solution, *ZeroBlock*, prevents block withholding using a technique free of forgeable timestamps. Moreover, we show that our solution is also compliant with nodes churn.

Keywords: Electronic Currency; Bitcoin; Cryptocurrency; ZeroBlock; block withholding; *selfish* mining

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's possession and employs Blockchain in order to prevent double-spending attacks. Blockchain is broadcasted 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 (nodes that follow the protocol).

Bitcoin is still vulnerable to various attacks including double-spending [27], *selfish* mining [9], Goldfinger [28], 51% attack [28] 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* nodes to waste their computational power such that their public chain becomes useless, whereas the private chain of the adversarial nodes is accepted as a part of the blockchain. To this end, the adversarial nodes reveal selectively their private blocks to make useless the blocks made by *honest* nodes.

Our contribution In this paper, we introduce a new solution named ZeroBlock which prevents block withholding (*selfish* mining) without using timestamps contrary the recent solution proposed by [26]. In our solution, if a *selfish* node wants to keep its block privately more than the expected time calculated by *honest* nodes, its block will expire and will be rejected by *honest* nodes. We show that, by using this solution, *honest* nodes never accept chains infested with block withholding . Also, we demonstrate that in a dynamic network if a *honest* node joins the network, it can diagnose the correct chain from chains infected with block withholding .

Paper Roadmap The rest of this paper is organized as follows: Section 2 presents an overview of Bitcoin, Section 2.1 presents how to generate a block and proof-of-work process, Section 2.2 presents block withholding attack and selfish mining, Section 2.3 mentions some other Bitcoin problems, Section 3 presents structure of information in Bitcoin netwok including *3.1* Transactions, *3.2* Blocks and *3.3* Blochchain. Section 4 presents our ZeroBlock solution altogether with its correctness proof. Finally, Section 5 presents some related work on crypto-currencies while Section 6 concludes the paper and presents some future research directions.

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 ownership's chain 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.

2.1 Block Generation and Proof-of-Work

A PoW is a cryptographic puzzle that is difficult to generate but easy to verify. Bitcoin network uses Hashcash [2] PoW system for block generation such that

a block is accepted by the network if miners perform PoW properly and successfully. The difficulty of PoW is adjustable regarding to the network power. Currently, the block generation rate is one block per minute. Note that since the success of solving PoW by every node in the network has a very low probability, the prediction on which node will generate the next block is almost impossible.

After a new block is generated, it is broadcasted in the entire network. If this new block is accepted as "head" of blockchain, other nodes start to work on this new block to extend it. A competition between two new blocks occurs if their respective hash on the previous blocks used in PoW process are equal and also they are broadcasted simultaneously. At this point a *fork* on blockchain happens. Since the block broadcast takes only a few seconds but the average of the time to generate a new block is around 10 minutes, "accidental forks" occur almost every 60 blocks [4,3]. In such situation, nodes have to select the block which is received first and, as a result, the second block will be ignored by the network. Consequently, nodes that worked on the second block wasted their computational power with no reward. Moreover, this may generate a fork on the blockchain which may divide itself into two branches. This competition continues until nodes generate new blocks on each of fork branches and finally the longest chain will be chosen by *honest* nodes.

2.2 Block Withholding or Selfish Mining

Block withholding attack was introduced as *"Selfish mining"* in [4] and also as *"Block Discarding Attack"* in [29]. This attack relies on "block concealing" and revealing only at a special time selected by some nodes called as Selfish Miners who does follow Bitcoin protocol. This attack is contrary to this concept that the first favor of a miner is to reveal his new block as soon as possible to be a part of main blockchain for achieving more revenue. According to [4], these Selfish Miners can learn revenues more than their share computational resources in comparison with a fair situation [30].

In block withholding strategy, a Selfish Miner after solving PoW and finding a new block, neither publish it in its mining pool nor in entire the network.

In [31] authors extends the selfish mining strategy and provides an algorithm to find optimal policies for Selfish Miners. [16] shows that expanding main blockchain method by adding newest block on the last block creates a simple model of "weak and non-unique" Nash equilibrium [31]. [31] shows that this optimal policies compute threshold such that honest mining would be a "strict and unique" Nash equilibrium. According to [32] miners are not motivated enough to propagate transactions. Authors in [33] introduces a cooperative game theory analysis relying on interactive mining pools.

According to [4], the *selfish* mining in Bitcoin network occurs as follows: In the case of the generation of a new block by the *honest* nodes, (1) if the size of *honest* nodes' blockchain is longer than the adversarial branch, the adversarial cartel tries to set its private chain equal to the public chain of the *honest* nodes. (2) If the adversarial branch is one block more than the *honest* nodes chain,

then adversarial nodes publish their private chain completely (3) If the adversarial branch is more than one block longer than the *honest* nodes chain, then the adversarial nodes publish only the head of their private chain. In the case of generation of a new block by adversarial nodes, they keep this new block private and in case of a competition with the *honest* nodes, publish it to win the competition. According to [4], the success of adversarial nodes in this competition is contingent on the α (i.e. adversarial nodes power) and γ (i.e. the proportion of *honest* nodes power which in a block competition work on an adversarial node block). According to the 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 $\alpha \geq 0.009$ [26].

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

Eyal and Sirer [4] suggest a solution according to which γ is fixed to 0.5 and consequently the threshold of successful block withholding increases to $\alpha \geq 0.25$. Ethan Heilman [26] introduces an approach named "Freshness Preferred" (FP). Using a random beacons and timestamps, *honest* nodes select more fresh blocks and the threshold becomes $\alpha \geq 32$ %.

2.3 Other Bitcoin Problems

In this section we mention some other Bitcoin problems. Due to publishing transaction logs fully publicly, the user's anonymity is protected only via the use of pseudonyms addresses [15]. In accordance with [8] the user's transactions can be simply linked together. Also, in case of linking one user's transaction to his identity, all of the other transactions performed by the user could be revealed. The user's privacy in electronic transactions comes back to Chaum's proposal about "anonymous" e-cash by using the "blind signatures" [8].

PoW causes some problems such that imposing participation in a mining pool managed by a single miner that decreases and may collapse decentralized nature of Bitcoin network particularly when a significant percentage of mining power belongs to a single mining pool (e.g. GHash.io [9,10]), a point at which a significant mining power of the network is in hand of a miner who is the manager of the mining pool and may lead to 51% attack.

The history of transactions as a public log is called a blockchain which is maintained, recorded and run by a group of miners who are rewarded with respect to their participation in the Bitcoin network [4]. [4] formally defined an attack to demonstrate that Bitcoin is not an "incentive-compatible" protocol which means that miners can collude in order to earn a reward larger than their fair share. As a result, the size of this colluding group may increase until converting into a majority. Since, Bitcoin needs a majority of *honest* miners (who perform exactly the prescribed protocol) if a colluding group is able to become a majority it can either prevent some transactions [4] or decide to reject originated coins belonging to a particular address leading to coins devaluation. Therefore,

users become disinclined to use these coins for the payment [11]. In other words, the decisions need to be accepted via a majority that by default is supposed to be *honest* [11]. In fact, the users vote according to their computing power to prevent double-spending. This action restricts the power of a specific group of users and leads to performing Sybil attack (i.e. forging different IDs in a "peer to peer" network) [11,24]. However, [11] demonstrates some critical decisions in the Bitcoin's network which may be controlled by a small set of entities.

On the other hand, some actions such as protocol updates are not considered to be decentralized, but they are managed by limited entities as administrators without respecting the "computing power" that they control, but depending on their roll in the network. In [11] the authors tried to enhance decentralized property of Bitcoin by reducing the impact of mining pools on the Bitcoin system.

3 Bitcoin Model

Bitcoin network consists of two major types of information. The first one are *3.1* Transactions that are entities to transfer the coins entire the network and the second one are *3.2* Blocks inserted into a "*public ledger*" called as *3.3* Blochchain and employed to "synchronize status among all Bitcoin nodes". We explain these key entities in the following sections [3].

3.1 Transactions

Briefly, a *transaction* transfers the coins from one or multiple "source account address" to "destination account address", where each "account address" has a unique key pair. Each "address" derives from related "public key" and employed to recognize the related "account". For each "Bitcoin transfer" between two account, a *transaction* must be generated including the "destination account address" and signed by "private key" of the "source account".

To calculate the "balance of account addresses", the public ledger calculates each *transaction* outputs (i.e. a "numeric value of coins") which has transferred the coins to an account. Thus, the balance is calculated as the "sum of these values of the coins" that are not still spent.

Each *transaction* is recognized by using "hash of serialized context of *transaction*". A *transaction* has also the inputs as "references to *transaction* output ownership".

When *transactions* are broadcast entire the network, the "public ledger" will be updated locally in each node. By passing the time, this copy may be unalike in "different miners" which may causes "inconsistency". This inconsistency itself may lead to some problems. For example, consider a miner receives a *transaction* that shows transferring some coins from *account "A"*, whereas this miner has not still gotten related *transaction* that shows these coins are "available" for *account "A"*. Or in another situation, multiple transactions may try to transfer some coins at several times that such "subversive behavior" called as "*double spending*" [3].

3.2 Blocks

Achieving a "consensus" in a distributed system is not a "trivial" concept and Bitcoin solution uses a "uncertain commitment of the *transactions*" and then "synchronizing" them at "orderly time intervals" by using *Blocks* that has been generated and broadcast by one miner entire the network.

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, where all miners have a "consensus" on the *transactions* of the *block*. [3]

3.3 Blockchain

A *blockchain* is a chain of *blocks* in "chronological order", where the *blocks* are put like a "directed tree form" such that each *block* is referenced by its "*parent block*". In this tree, the "*genesis block*" is the root. The "most distant *block*" from "*genesis block*" is called as the "*Head of blockchain*" [3].

Fork There is a "*blockchain fork*", when in a *blockchain* there are more than one *Head*. In such situation, the network miners do not have a "*consensus*" on a unique *Head*. At this point, a miner N whose "*Head of blockchain*" distance from *genesis block* is H, when it receives another "*blockchain Head*" whose distance from "*genesis block*" is H' such that $H < H'$ then miner N changes his *blockchain Head* from H to H' [3]. The "*block withholding attack*" is based on *blockchain fork* that we introduce ZeroBlock solution to prevent it.

4 ZEROBLOCK

In this section we introduce a new solution, ZeroBlock, to prevent block withholding and *selfish* mining in Bitcoin. The key of our solution is that each block must be generated and received by the network within an *expected time* (its computation is detailed in section 4.1). *Honest* nodes perform their computation in rounds with the length equal to the *expected time* (computed locally by each node). Instead of broadcasting the entire chain as in Bitcoin, nodes broadcast only the new block (the head of the chain). Hence, within expected time a *honest* node either receives a block, or broadcasts a block. Otherwise, it generates a dummy block called in the following Zeroblock. A detailed description of the algorithm is proposed in section 4.2.

We demonstrate that if a *selfish* node wants to keep its block private more than the expected time calculated by *honest* nodes, its block will expire and will be rejected by *honest* nodes. In Theorem 1, we show that by using this solution *honest* nodes, regardless of their percentage in the network, never accept chains infested with block withholding . In Theorem 2 we demonstrate that in a dynamic network, regardless of the *honest* nodes percentage, if a *honest* node joins the network, it can diagnose an infested chain with block withholding .

In Theorem 3, 4 and 5, we show that if $\lfloor \frac{n+2}{2} \rfloor$ nodes are *honest*, then when an *honest* node joins the network, it can diagnose the correct chain which belongs to *honest* nodes. Moreover, the *selfish* cartel cannot impose its *selfish* chain as the common chain. Furthermore, in Theorem 5 we prove that our solution also prevents the intentional forks.

4.1 Preliminaries

Expected Time computation. Note that the time for generating a block depends on the difficulty level for solving PoW and on the size of the network. This time is predictable (currently it is around 10 minutes). We call this time as *block generation time*. Note also that depending on the size of the network, the block *broadcast time* is also predictable [3]. As a result, we define the *ExpectedTime* as follows:

$$ExpectedTime = BGT + BT \tag{2}$$

where: BGT = *block generation time* , BT = *block propagation time*

Estimation of PoW time. Consider the following condition:

$$IF\ FirstHF(hash + answer) \leq D\ then$$

$$PoW succeeded$$

where:

$$\begin{cases} hash : SecondHF(Input : [Previous\ Blocks]) \\ answer : a\ Random\ as\ PoW\ answer \\ D : Difficulty\ Level \\ FirstHF : First\ Hash\ Function \\ SecondHF : Second\ Hash\ Function \end{cases}$$

In this condition, if D has a small value then PoW is more difficult. So, the time of solving PoW is predictable by regulating D. Thus one can estimate the required number of invoking $FirstHF(hash + answer)$. As a result, one can estimate the total required time to learn PoW answer.

Estimation of Block Propagation Time We use the same model of [3] to estimate the block propagation time entire the network. We define $DT_{b,d}$, where b is *generated block* and d is *destination node*, as time difference between first announcement by block generator (creative node) and the time at which destination node receives block such that $DT_{b,d} = 0$, when d is *creative node*. According to [3], timing information consists of a local time-stamps when announcement has been received and $DT_{b,d}$ is estimable by subtracting first announcement timestamp from all announcements for the same block.

4.2 ZeroBlock Algorithm

In the following we detail the Zeroblock algorithm (Algorithm 1).

Notations Algorithm 1 uses the following notations:

- *correct chain* : a chain which is not infested with block withholding .
- *creative node* : is a node which in an *Expected Time* interval can solve PoW and generate a new block.
- *Block Propagation Time (BT)* : is the delay necessary to the network to learn the existence of a block once it has been announced by the block generator (see Section 4.1).
- *Block Generation Time (BGT)* : the time needed to solve PoW (see Section 4.1).
- *Expected Time* is the time interval computed by Equation 1. During an *Expected Time* interval a node should solve PoW, otherwise if the node does not receive a new block from the network, it has to generate a ZeroBlock.
- ZeroBlock : is a dummy block including the index of *Expected Time* and the hash of previous blocks.
- *consumer node* : is a node which in an *Expected Time* interval cannot solve PoW and hence cannot generate a new block. Thus, it either receives a new block or generates a ZeroBlock.

We assume that at the beginning of the computation, all nodes (*honest*) agree on a first block, the *genesis* block. The *genesis* block should be hard-coded into the software. Each node has a local chain which initially is equal to the *genesis* block. Nodes have access to a common clock and each node maintains locally a variable *LTime*, a seconds counter (initially 0) that is updated at each time second. Each node maintains locally the variable ET, initially equal to *zero*, which will be increased with the value of *Expected Time* at each loop. Moreover, each node keeps a boolean parameter, $FlagNewBlock$. As soon as a new block is generated, its value is changed to $True$. After the initialization phase, the Zeroblock algorithm, Algorithm 1, starts an infinite loop. In this loop, ET value is updated by using $refresh()$ function as follows :

$$ET = ET + (BGT + BT)$$

where *ExpectedTime* is computed using Equation 1. While the counter *LTime* is less than or equal to ET, a *honest* node checks its input if there is a new block. If there is a new block head, the *honest* node investigates if its PoW has been properly computed. by verifying if the new block head includes the hash of the current local chain. Then this new bock becomes the head of the local chain. If during the *Expected Time*, there is no new input, a *honest* node generates a ZeroBlock that includes the hash of a fixed value and adds this new block to its local chain. This block helps to prevent block withholding , because, the next block will include the hash of the local chain that includes also the hash of this ZeroBlock. Thus, if an adversarial node does not reveal a block during the

expected time corresponding to the generation of this ZeroBlock, it cannot use this expired block. In the case when a *honest* node receives more than one new block during an expected time interval it accepts the first one.

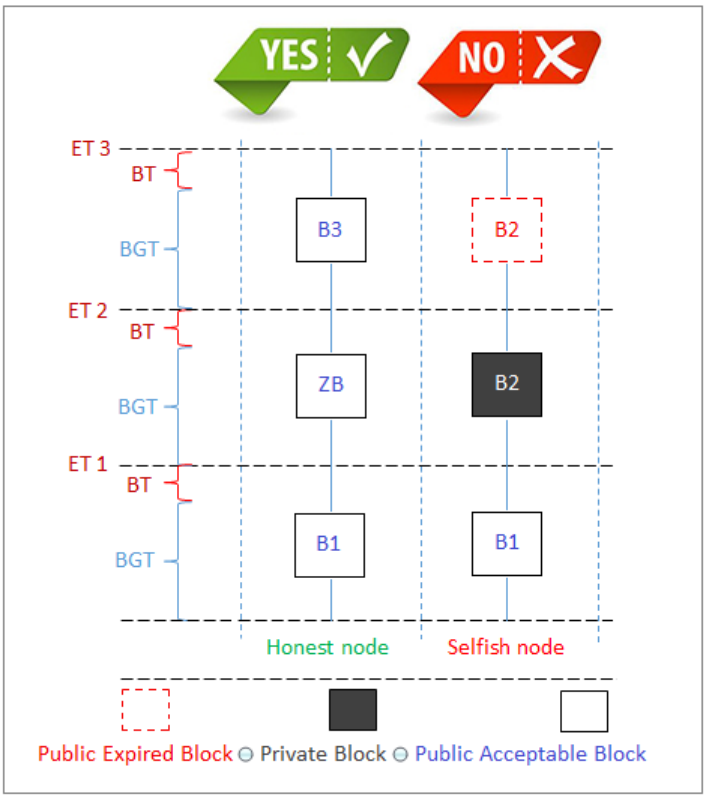

Fig. 1. ZeroBlock generation by a *honest* node to prevent block withholding . B_2: B_2 has been generated by a *selfish* node and has been kept until ET_2 and thus has been expired. Thus, PoW of B_2 includes only hash of $B1_1$ and thus will be rejected by *honest* node. ZB (ZeroBlock): the *honest* node didn't receive a new block until ET_2 and thus generates a ZeroBlock. B_3: PoW of B_3 includes hash of $(B_1 + ZeroBlock)$ and will be accepted.

More in details, each node, N, that executes Algorithm 1 performs the following steps.

- The initialisation : (*Lines* 1 and 2) where ET (*Expected Time* counter) and *LTime* (the seconds counter) are set to 0. Then, *Line* 3, node N has a local chain that at the beginning equals to Genesis block (with following definition). Also, FlagNewBlock is set to false in *line* 4.
- In *Line* 11 node N starts an infinite loop where:

Algorithm 1 ZeroBlock algorithm

```
1: ET ← 0
2: LTime ← 0                                          ▷ LTime is a seconds counter
3: localChain ← Genesis
4: FlagNewBlock ← False
5: counter ← 0
6: HPrB ← 0                                           ▷ hash of previous blocks
7: D ← DifficultyLevel
8: NewBlock ← Null
9: IndexET ← 0                                        ▷ Index of Expected Time
10: ansPoW ← 0                                        ▷ answer of PoW
11: while (True) do
12:     if (FlagNewBlock = False) ∧ (ET ≠ 0) then
13:         ZeroBlock ← SecondHF(getHead(localChain)) + SecondHF(ZB) + IndexET
14:         localChain ← join(ZeroBlock,localChain)
15:     end if
16:     refresh(ET)
17:     IndexET ← IndexET + 1
18:     while (LTime ≤ ET) do
19:         NewBlock ← checkInput()
20:         if (NewBlock ≠ Null) then
21:             HPrB ← SecondHF(getHead(localChain))
22:             if (FirstHF(HPrB,ansPoW) ≤ D) ∨ (HPrB = NewBlock.HPrB) then
23:                 localChain ← join(NewBlock,localChain)
24:                 NewBlock ← Null
25:                 FlagNewBlock ← True
26:                 Break
27:             end if
28:         end if
29:         if (FlagNewBlock = False) then
30:             HPrB ← SecondHF(getHead(localChain))
31:             if (FirstHF(HPrB , counter) ≤ D) then     ▷ Proof-of-work succeeded
32:                 ansPoW ← counter
33:                 NewBlock ← GenerateBlock(ansPoW,HPrB)
34:                 BroadcastBlock(NewBlock,ansPoW)
35:                 FlagNewBlock ← True
36:                 counter ← 0
37:                 Break
38:             end if
39:             counter ← counter + 1
40:         end if
41:     end while
42: end while
```

- In *Line* 12) node N checks if there is no new block from the network (FlagNewBlock equals to False) and if the local chain's block head is not the genesis block (ET is not equal to zero). For the first time, always these two conditions are *False*. Then in *Line* 16) node N invokes refresh() function that does following operation: $ET = ET + (BGT + BT)$. Then, in *Line* 17 one unit is added to the index of ET.
- While node N has time to solve PoW, (*Line* 18, *LTime* (seconds counter) is less than ET), N checks its input if there is a new block received from the network (*Line* 19). If yes, (*Line* 22) node N verifies the answer of PoW of the new block. If PoW has been done correctly (*Line* 23) , N adds the new block to its local chain and changes value of FlagNewBlock = True (*Line* 25) and then leaves the while loop and goes (*Line* 12) and then (*Line* 16). If there is no new block in its input (condition of (*Line* 20 is incorrect)) node N tries to solve PoW (*Line* 31), if it succeeds, it generates a new block (*Line* 33) and broadcasts it (*Line* 34).
- In case PoW does not succeed (the condition of (*Line* 31) is incorrect), node N increases the counter and goes back to *Line* 18.
- If *LTime* is more than *Expected Time*, immediately node N generates a ZeroBlock (*Line* 13) and then adds it to its local chain (*Line* 14).
- Then, node N refreshes *Expected Time* (*Line* 16) and increases one unit the variable index of expected time (*Line* 17).
- If a ZeroBlock has been generated, node N (honest) rejects a selfish block in (*Line* 22) because the two conditions are incorrect. Since HPrB(the hash of node N's local chain head) that includes the ZeroBlock is not equal to $NewBlock.HPrB$ (hash of selfish block) that does not include the ZeroBlock and thus its answer of PoW is incorrect and then goes (*Line* 29).

4.3 Correctness of Algorithm 1

Theorem 1. Honest *nodes, regardless of their percentage in the network, never accept chains infested with block withholding .*

Proof. Each *ExpectedTime* interval, a *honest* node generates a new block and broadcasts it and waits for a new block. If it does not receive a new block, it generates a ZeroBlock.

A block withholding can occur if a *selfish* miner has generated a new block but kept it privately, a least an *ExpectedTime* interval. At this point, to prevent block withholding , all *honest* nodes which did not receive a new block, generate a ZeroBlock including the hash of previous blocks.

After an *ExpectedTime* interval (*Line* 18), all *honest* nodes leave the while loop and generate a ZeroBlock since two conditions in (*Line* 12 are true. This is due to the fact that no new block has been received (*FlagNewBlock = False*) and ET has been refreshed ((*Line* 16 ET $\neq 0$). A *selfish* miner (which here is a creative node) has kept a new block after *ExpectedTime* and solved PoW for an expired block. This block will not be accepted by *honest* nodes since all of them generated a ZeroBlock. Therefore, PoW must be renewed such that it should include the recent ZeroBlock.

Theorem 2. *In a dynamic network, regardless of* honest *nodes percentage in the network, if a* honest *node joins the network, it can diagnose the correct chain from infested chain with block withholding .*

Proof. For n *ExpectedTime* intervals, there are n different blocks (including standard block and ZeroBlock). In each *ExpectedTime* interval, a *honest* node is either a creative node which generates a new block or a consumer node which either receives a new block from other nodes or generates a ZeroBlock. Thus, for each different *ExpectedTime*, a *honest* node has a different block. If a block withholding occurs, then after n *ExpectedTime* intervals there are $(n-1)$ different blocks (see Figure 4.2: for 3 *ExpectedTime* intervals, on the side of *honest* nodse there are 3 different blocks including $B_1, ZeroBlock_2, B_3$ but on the side of adversarial node there (3 - 1) = 2 different blocks including B_1, B_2).

When a *honest* node joins the network at time t, it announces its entrance into entire network in order to receive the last version of the correct chain from other nodes. Then, using the division of the current time into *ExpectedTime* intervals, it can find the correct chain. The new node computes $EDB = \lfloor \frac{t}{ExpectedTime} \rfloor$, where t = current time and accepts a chain if and only if (1) it contains EDB different blocks (2) each block includes the hash of the previous blocks. This last operation can be done by using *verifyPoW (input: NewBlock)* to investigate if each block's PoW has been properly computed.

Theorem 3. *In a dynamic network, if* $\lfloor \frac{n+2}{2} \rfloor$ *nodes are* honest*, then when a* honest *node joins the network, it can diagnose the correct chain which belongs to* honest *nodes.*

Proof. Following the proof of Theorem 2, when a new node joins the network and broadcasts a message including its address to announce its entrance, other nodes respond with the last version of their local chain. The new node then compares the received chains and selects the chain sent by a majority. Thus, if the half of the network plus one is *honest*, the *honest* chain will win and the new node will have the correct chain that belongs to *honest* nodes. After that, the new node, similar to other nodes, will continue the protocol as described in algorithm. According to this process, each node is able to leave and re-join the network infinitely.

Theorem 4. *If* $\lfloor \frac{n+2}{2} \rfloor$ *nodes are* honest*, then a* selfish *cartel cannot impose its* selfish *chain as common chain.*

Proof. In the worst case, there is only a single *selfish* cartel with a single common *selfish* chain. In that case, there are two groups of chain: (1) the *selfish* chain belonging to the *selfish* cartel and (2) the correct chain belonging to *honest* nodes. Since $\lfloor \frac{n+2}{2} \rfloor$ nodes are *honest*, in this case, the majority chain (the one of honest nodes) will be chosen by the network.

Note that in [3] the authors show that the accidental forks occur rarely due to the long propagation time. Therefore, most of the forks occur intentionally. In the following we prove that our solution prevents intentional forks.

Theorem 5. *Algorithm 1 prevents intentional forks.*

Proof. Since by using ZeroBlock, none of *honest* nodes accept on *selfish* miners' chains (as described in Theorem 1), *selfish* miners thus are not able to impose working on their chain on *honest* nodes and our approach thus prevents block withholding , thus it is not possible to intentionally forking blochchain "out of *selfish* cartels' chain".

5 Related work on crypto-currency

In this section, we present briefly the state of the art concerning electronic currencies and peer to peer electronic payment systems. Most of the work targeted so far anonymity, privacy and avoiding "double-spending" attacks. Ripple [17] is an electronic currency such that all users are able to issue a currency but it will be accepted only by the peers who have reliance on the issuer. Moreover the acceptance of a transaction needs a chain of trusted mediators between the sender and the receiver. i-WAT [19] is another similar system where the chain of trusted mediators is initiated using digital signatures. KARMA [20] is another electronic currency in which the central authority is formed of a set of users who manage all transactions. In the event of numerous transactions this central authority risk to be overheaded. PPay [18] is a "peer to peer" network in which the coin issuer has the responsibility of keeping track of the transactions in the system. However, it has the same problem as KARMA concerning the overhead. Mondex as smart card electronic currency [21] is similar to an issuer bank for currency issuance that stores the value as electronic information on a smart card instead of physical notes and coins [22]. However, the remained problem is its inability to determine performing "double-spending" via one of the owners in the chain. To avoid using trusted central authority such as the coin issuer, all transactions must be announced publicly [23] to achieve an agreement on an "unique" history of the transaction [1]. Bonneau *et al.* [8] introduce Mixcoin, fully compatible with Bitcoin that adds an accountable service (Mixes) to determine the theft of service, means the "accountable" mixes sign warranties to users such that if a user sends n coins to Mixes until time of t_1, the Mixes have to return n coins to this user by time t_2. As a result, in case of misbehaving a Mix, the user can devalue the reputation of the Mix by publishing Mix's warranty. Mixing services exchange the user's coins arbitrarily with other users' coins in order to make unclear the ownership, but without ensuring protection from theft by the service (i.e. Mixing services). Another attempt to enhance the anonymity of Bitcoin was proposed in [15]. The authors propose Zerocoin a cryptographic extension of Bitcoin which despite strong anonymity [25], it still needs an advanced cryptography model and significant changes to be compatible with Bitcoin [8]. According to [25] Zerocoin reveals the payment destinations along with the transaction amount. In [25] the authors introduce the concept of Decentralized Anonymous Payment (abbreviated DAP) according to which a user is able to pay privately another one. Transactions do not reveal the sender

of the payment, the destination and the transaction amount. DAP scheme is used in Zerocash as a decentralized anonymous payment system [25].

6 Conclusion and future work

In this paper, we introduced a new solution to prevent block withholding and *selfish* mining as one of the most important problems of Bitcoin. We demonstrated that if an adversarial *selfish* node wants to keep its block privately more than the *Expected Time* calculated by *honest* nodes, its block expires and will be rejected by the *honest* ones. Also, by using this approach, it is not possible to intentionally fork a blockchain. Furthermore, we show 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*.

As future research we would like to explore solutions for other weaknesses of Bitcoin such as Goldfinger attacks or double spending. Moreover, we would like to investigate the privacy and anonymity of users. Note that the use of public logs leads to transparency but it has also negative effects on the privacy and anonymity of the users [14]. On the other hand, PoW causes some problems such that imposing participation in a mining pool managed by a single miner that decreases and may collapse decentralized nature of Bitcoin network particularly when a significant percentage of mining power belongs to a single mining pool (e.g. GHash.io [9,10]), a point at which a significant mining power of the network is in hand of a miner who is the manager of the mining pool and may lead to 51% attack.

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. Stalder, Felix. "Failures and successes: notes on the development of electronic cash." The Information Society 18.3 (2002): 209-219
22. `http://www.mastercardconnect.com/mol/molbe/public/login/ebusiness/smartcards/mondex/about/index.jsp`
23. Dai, Wei. "B-money." Consulted 1 (1998): 2012
24. Douceur, John R. "The sybil attack." Peer-to-peer Systems. Springer Berlin Heidelberg, 2002. 251-260
25. Ben Sasson, Eli, et al. "Zerocash: Decentralized anonymous payments from Bitcoin." Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014
26. Heilman, Ethan. "One weird trick to stop *selfish* miners: Fresh bitcoins, a solution for the *honest* miner." (2014)
27. 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.
28. 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
29. Bahack, Lear. "Theoretical Bitcoin Attacks with less than Half of the Computational Power (draft)." arXiv preprint arXiv:1312.7013 (2013)

30. 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
31. Sapirshtein, Ayelet, Yonatan Sompolinsky, and Aviv Zohar. “Optimal selfish mining strategies in Bitcoin.” arXiv preprint arXiv:1507.06183 (2015)
32. Babaioff, Moshe, et al. “On bitcoin and red balloons.” Proceedings of the 13th ACM conference on electronic commerce. ACM, 2012
33. 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