Broadcasting Intermediate Blocks as a Defense Mechanism Against Selfish-Mine in Bitcoin

Ren Zhang
KU Leuven ESAT/COSIC, iMinds, Leuven, Belgium
Email: ren.zhang@esat.kuleuven.be

***Abstract*—**

I. Introduction

Bitcoin [1] is a decentralized crytocurrency system launched by a mysterious creator Satoshi Nakamoto. Its novel and open design attracts not only a lot of users but also much attention from academia. The core of Bitcoin's operation is a data structure called the *blockchain*, which records all past transactions to prevent double spending. Some participants of the Bitcoin network, called *miners*, work on solving a cryptographic puzzle generated from a set of new transactions. Each time a puzzle is solved and recognized by the network, the set of transactions, together with the puzzle solution constituting a *block*, is added to the blockchain. The miner who solves the puzzle can claim a *block reward* of new bitcoins. This incentive mechanism encourages miners to contribute their resources to the system, and thus protects the integrity of the blockchain. Hence the fairness of this mechanism is essential to the security of the system. Nakamoto's original analysis inherently assumes that the chance that a miner can earn the next block reward is linear to its proportion of the computational power in the network.

Unfortunately, this assumption has been disproved. Research has shown that a malicious miner can earn more block rewards than its share of computational power by deploying a sybil attack to interfere with the propagation of other miners' blocks [2] or selectively delaying publication of his own blocks [3]–[5].

In this paper we focus on the second attack, known as the *selfish-mine* strategy. The attack exploits the following *fork-resolving policy* of Bitcoin. When more than one blocks are mined with the same preceding block and the blockchain is *forked* into a tree, a miner will adopt and mine on the longest chain, or the first block she receives when several chains are of the same length. The competing situation is called a *block race*. Eventually, all miners would convert to the same longest chain as the *main chain*, and discard blocks that are not in the main chain. A selfish miner keeps its discovered blocks private and continues to mine on top of it, hoping to gain a longer lead ahead of the public chain. When the public chain approaches the private chain in length, the selfish miner reveals her private chain to the public and claims the block rewards. There are two situations when the selfish miner may win a block race: (1) when competing chains have equal length, the selfish miner can increase her chance by having her blocks reach more miners first; (2) the selfish miner's chain is longer, in this case she always wins. By causing other miners' work to be discarded, the expected block rewards of a selfish miner is larger than its computational power share.

This attack has not been observed in practice. However, it is considered to be important by many researchers, since the attack targets the incentive mechanism of Bitcoin, which is fundamental to Bitcoin's security. Specifically, since the revenue of a selfish miner rises superlinearly with its computational power, rational miners would prefer to join the selfish miner for a higher input-output ratio. This is contrary to the conventional belief which the expected revenue of a miner is the same despite of which pool she joins. Once a majority pool is formed, it is capable of controlling the entire blockchain, and the decentralized nature of Bitcoin is damaged.

Existing defenses mainly focus on situation (1) by either lowering the chance of other miners working on the selfish miner's block [4], [6] or reducing block rewards of the winner blocks for all block races [5]. The first approach only mitigates the attack: even when no honest miner works on the selfish miner's branch when situation (1) happens, the selfish-mine strategy is still more profitable as long as the selfish miner controls more than one third of the total mining power. In the second approach, honest miners would suffer some collateral damage along with the selfish miner.

In this work we propose another defense mechanism against selfish-mine. Our defense addresses both situations by requiring miners to publish *intermediate blocks*, or *in-blocks* for short, blocks that are valid with slightly lower puzzle difficulty, and mine on each others' in-blocks. When a fork happens, the branch with most total amount of work, rather than the longest chain, will be adopted. The defense is based on one key observation: although the selfish miner may be lucky to find a longer chain than the public one, the total amount of work of the private chain is almost certain to be less than that of the public chain. Under our scheme, a selfish miner needs to have almost half of the computational power of the network to gain an unfair advantage, thus the selfish-mine strategy is no longer a threat to the system. With the help of some block compression methods, the additional communication and computation overhead of our defense is marginal. Our defense is evaluated with both formal analysis and simulations. Finally, we discuss how broadcasting in-blocks may help the mitigation of double-spending attacks on unconfirmed transactions in Bitcoin.

II. Background

A. Basics of the Bitcoin Protocol

An account in Bitcoin is actually a public/private key pair. To receive payments, a user transfers an *address*, which is the hash of a public key, to the sender. The sender can then construct a transaction, which consists of at least one input and one output. An *input* is usually an output of a previous transaction, owned by the sender. An *output* is the receiver's address and an amount. The total amount of inputs must be no less than the total amounts of outputs in a transaction. The difference is called *transaction fee*, which will go to the miner who includes the transaction in the blockchain. The entire transaction must be signed by the sender's private key to be considered valid, so that only the one with the corresponding private key to the address can spend an output.

To ensure all participants have a consensus on valid transactions, a copy of the blockchain is maintained by each node of the Bitcoin overlay network. Each block in the chain contains its distance from the first block, called *height*, the double-hash of the preceding block, a set of transactions, and a nonce. Information about the preceding block guarantees that a miner has to choose which block to mine on before she starts mining. A special *coinbase* transaction in the block allocates some amount of new bitcoins to the miner herself without any input. To construct a valid block, miners work on finding the right nonce so that the double-hash of the block is smaller than a certain value, called *block difficulty*. The block difficulty value determines the difficulty of the task, which is adjusted every 2016 blocks so that on average a block is generated every ten minutes. Once a valid block is found, the miner publishes it to the entire overlay network. If the block ends up being in the main chain, the coinbase output and all transaction fees in the block belong to the miner. Miners are typically organized into *pools* to decrease the variance of mining revenues.

A natural fork only happens when another block is found before one block finishes its propagation. Hence the occurrence of natural forks can be roughly seen as a function of block interval and block propagation time. Nowadays a block can reach 90% of publicly reachable nodes in around 20 seconds on average [7], which is generally considered satisfactory. As a result, block forks happen relatively rare, about once per 60 blocks [8]. An important fact to note is that in the current implementation, nodes only propagate the first block they receive in case of a block fork. Since the competing blocks are not propagated, when a miner receives an unseen longer branch, she cannot determine whether the branch was kept private or none of her connected nodes happens to receive it first.

Since Bitcoin nodes choose the longest chain by default when a block fork happens, an attacker controls more than 50% of the computational power of the network can generate a longer chain than that of honest miners, thus have total control over the blockchain. The attacker can then reverse past transactions of herself and deploy a *double-spending attack*. Most existing research [2]–[6], [9], [10] follows the assumption proposed by Nakamoto's original paper [1] that the majority of computational power would follow the protocol.

By convention a transaction needs to be "buried" under six blocks in the blockchain to be considered valid. Bitcoin by default provides no protection against double spending on *fast payments*, which do not require several blocks buried on top of the transaction to confirm. However researchers are still trying to increase the difficulty of such attacks [9], [10].

B. The Selfish-Mine Strategy

The idea of selectively delaying publication of blocks to gain an unfair advantage of block rewards appears as early as 2010 in a Bitcoin forum thread [3]. The attack is given the name "selfish-mine" and formally described and analyzed by Eyal and Sirer in 2013 [4].

All research papers about selfish-mine are based on the following assumptions, implicitly or explicitly:

Assumption 1. *More than half of the computational power of the network follows the honest mining strategy.*

This assumption is adopted for most Bitcoin research.

Assumption 2. *There is only one colluding pool of miners.*

Malicious miners can achieve higher input-output ratio by joining their forces, therefore we only consider this stronger form of attack. We will use "the selfish miner" instead of "the colluding pool of miners" for simplicity.

Assumption 3. *The selfish miner may be able propagate her blocks faster than honest miners, but does not have the power to downgrade the propagation speed of other blocks.*

Defending against a sybil attacker in P2P network is a known hard problem.

Assumption 4. *Once the public branch becomes strictly longer, the selfish miner would give up her secret branch and start mining on the public branch.*

Since in that case the majority of mining power would work on the public branch, the chance that the selfish miner can finally surpass that branch is slim.

Algorithm 1 Selfish-Mine

```
on consensus
 1: $publicChainLen \leftarrow 0$
 2: $privateChainLen \leftarrow 0$
 3: mine on the public branch
on my pool found a block
 4: $privateChainLen \leftarrow privateChainLen + 1$
 5: if $publicChainLen = 1$ and $privateChainLen = 2$
 6:     publish the entire private chain
 7:     goto consensus
 8: else
 9:     keep mining on the private chain
on others found a block
10: $publicChainLen \leftarrow publicChainLen + 1$
11: if $privateChainLen - publicChainLen < 0$
12:     goto consensus
13: if $privateChainLen - publicChainLen = 0$
14:     publish the entire private chain
15:     mine on the private chain
16: if $privateChainLen - publicChainLen = 1$
17:     publish the entire private chain
18:     goto consensus
19: if $privateChainLen - publicChainLen > 1$
20:     publish the first unpublished private chain block
```

Algorithm 1 is a description of the selfish-mine strategy. Generally speaking, the selfish miner keeps her blocks private and selectively reveals some blocks to invalidate the work of honest miners. Since the selfish miner has less mining power than honest miners, once in a while a new honest block will surpass the height of the private branch. At this moment, according to Assumption 4 the selfish miner would give up her private chain and start mining on the public chain. We call this state a *consensus* and start the strategy description here (line 1 to 3). When the selfish miner finds a block, she usually keeps mining on top of it (line 9) unless there is already a block race with an equal-length public chain. In that case she publishes the block and claims the block rewards (line 5 to 7). When a block is found by honest miners, if the public chain is longer, the selfish miner gives up and goes to consensus (line 11 to 12). If the public chain length is the same with the private chain length, the selfish miner publishes her secret chain and hopes to win a block race (line 13 to 15). At this state the mining power working on the selfish miner's block depends on the propagation speed of two competing blocks. If the next block is mined on top of the selfish miner's block, whether by an honest miner or the selfish miner, the selfish miner gets the block reward. Otherwise she loses the block race and starts from consensus. When the lead of the private chain is reduced to one, the selfish miner publishes her blocks (line 16 to 18). When the lead is more than two, every time an honest miner finds a block, the selfish miner publishes one block along with it, so that the publicly visible part of the private chain is the same as the public chain (line 19 to 20), until the lead is reduced to one.

The selfish miner's expected revenue depends on two parameters: α, the mining power share of the selfish miner, and γ, the ratio of honest mining power that would work on the selfish miner's block during a block race. It can be seen that in a block race, the mining power share working on the selfish miner's block is $\alpha + (1 - \alpha)\gamma$. In current implementation, a selfish miner can increase her γ value by connecting to a large number of publicly reachable nodes to achieve a higher block propagation rate. The analysis in [4], [5] shows that when $\gamma = 1$, the expected reward share of the selfish miner is larger than α for any $\alpha > 0$; even when $\gamma = 0$, the selfish miner can still gain an unfair advantage when $\alpha > 1/3$. For the remainder of the paper we use *profitable threshold* to denote the minimum α that enables the selfish miner to gain an unfair advantage.

Bahack had shown that under three reasonable assumptions, any mining strategy that may cause other miners' blocks to be discarded is a member of a family of strategies $st_k, k = 0, 1, 2, \cdots, \infty$ [5]. Specifically, st_0 is the honest strategy, and st_1 is the sefish-mine strategy in [4]. The author further proved that for a given pair of α and γ, if st_1 is not more profitable than st_0, no member in the family is. Therefore it is adequate that we focus our discussion on st_1. Our defense and analysis are applicable to all members of the family.

C. Existing Defenses

It is a consensus that when a block fork happens, nodes should broadcast all competing valid blocks, instead of just the first one they receive [4]–[6]. The current implementation is to prevent DoS attack, however it is difficult to deploy a DoS attack with valid blocks, since they are relatively expensive to generate [8]. This modification limits the selfish miner's ability to interfere with the propagation of competing blocks and enables miners to freely choose which block to mine on. Another benefit is that blocks that are intentionally kept secret by their finders would be identified more easily.

Eyal and Sirer proposed that when a miner learns of competing branches of the same length, she should choose which block to mine on uniformly at random [4]. By doing this, γ is fixed at $1/2$, and the profitable threshold becomes $1/4$.

Bahack proposed a *fork-punishment rule*: blocks with a competing block receive no block reward. The first miner who incorporates a proof of the block fork in the blockchain can get half of the forfeited block reward. Although rendering the selfish-mine strategy unprofitable, this defense requires a fundamental change on the blockchain protocol and is not backward compatible. Besides, honest miners would suffer from collateral damage of this defense when natural fork happens.

Heilman suggested that each miner incorporates the latest unforgeable timestamp issued by a trusted party into the block she is working on [6]. The publicly accessible timestamp is issued periodically with a suggested interval of one minute. When two competing blocks are received within w seconds, a miner should prefer the block whose timestamp is fresher. They claimed that this mechanism can raise the profitable threshold to 32%. There are two disadvantages of this scheme. First, introducing an extra trusted party into the system is inconsistent with Bitcoin's decentralized philosophy. Second, this scheme enables a new attack: when a block is found, a selfish miner can keep mining with a fresher timestamp within the w-second window. Once success, the first block would be discarded by honest miners thus the selfish miner can again earn a higher expected revenue than its fair mining power share [11].

The last countermeasure is brought up in a draft paper written by Shultz [11]. It requires each solved block be accompanied by a certain number of signatures proving that the block is witnessed by the network before miners can continue to work on it. Moreover, each block and each signature needs to incorporate an unforgeable timestamp. However, the paper does not explain how to prevent the selfish miner from generating these signatures herself or refusing to sign other miners' blocks.

III. Our Defense Mechanism

It can be seen that existing defenses only focus on the situation that competing blocks have equal length. However, a relatively resourceful selfish miner may abuse her luck of finding consecutive blocks to gain an unfair advantage even if she loses all equal-length block races (line 5 to 7 and 16 to 20 of Algorithm 1). As a result, no existing defense can defend against a selfish miner with $\alpha > 1/3$ without any collateral damage. In this part, we present a defense mechanism that can raise the profitable threshold to more than 1/3. Our key observation is inspired by the GHOST rule proposed by Sompolinsky and Zohar [12]: a selfish miner with less than half of the total mining power may be able to generate a longer chain, however the total amount of work of the private branch is almost certain to be less than that of the public branch. In this section we first describe the mining algorithm incorporated with our in-block broadcasting mechanism and the corresponding main chain selection policy, then discuss the computation and communication overhead of our design and the incentives for nodes to comply.

A. Mining and In-block Broadcasting

Definition 1. *We say Q is a* in-block*, if Q is a valid block in every aspect except that the double-hash of Q is not lower than the block difficulty d, but satisfies $d \leq H(Q) < d \cdot k$, in which H is the double-hash function, k is a predefined parameter of the system.*

Definition 2. *A transaction T_1 is* in conflict *with a block/in-block A, if there exists a transaction T_2 in A, such that T_1 and T_2 are two different transactions that share at least one common input; a transaction T is* new *to a in-block Q, if T is not in conflict with all blocks in the chain containing Q, and T is not included in the chain.*

Definition 3. *A in-block Q is called a* in-predecessor *of a block/in-block A, if they have the same preceding block, and either of the following conditions holds: (1) there exists a in-block B such that Q is a in-predecessor of B and B is a in-predecessor of A; (2) after removing all transactions in Q that are in conflict with A, the remaining transaction set Q' is a proper subset of that of A; besides, there exists at least one transaction in A that is new to Q. We use $Pd(A)$ to denote the in-predecessor set of A known by a node.*

Definition 4. *From a node's local perspective, the* in-height *of a block/in-block A is defined as*

$$h_q(A) = \begin{cases} 1 & Pd(A) = \emptyset \\ \max\{h_q(Q) | Q \in Pd(A)\} + 1 & otherwise \end{cases}$$

We can see a block B, its immediate follow-up block(s), and the in-blocks between them as a directed acyclic graph. The system parameter k determines the expected number of in-blocks between two blocks. Every edge of the graph indicates a in-predecessor relation. If Q is a in-predecessor of A, we can say with high certainty that A is mined after Q. Our definition of in-predecessor can tolerate inconsistency in terms of double-spending transactions, but does not tolerate missing transactions. The in-height of a block/in-block is the length of its longest path to the preceding block.

Our mining and in-block broadcasting algorithm is described in Algorithm 2. Our defense requires miners to publish in-blocks they found during mining, and include transactions of received in-blocks in the blocks they are working on. A in-block is valid only if it is received in-time: a miner should delete a in-block under two conditions (line 6 to 7): (1) when we have already received one follow-up block of the same preceding block; (2) when the in-height of the new in-block is more than τ behind the current maximum in-height of in-blocks with the same preceding block. The parameter τ is a tolerance threshold of in-block propagation. When $\tau = 0$, as soon as a node has received a in-block of in-height h, all later-received in-blocks mined on the same block with in-height less than h would be rejected. Every time a block/in-block is received, a miner should re-evaluate which block to work on and adjust her working transaction set to make sure her future block includes as many in-blocks as possible (line 14 to 18). To ensure all blocks/in-blocks that have a in-block Q as their in-predecessor must be mined after Q, each miner can include some transactions

Algorithm 2 Mining and in-block Broadcasting

```
on found/received a new valid block $B$
 1: Broadcast $B$ to the network
 2: $maxQHeight(B) \leftarrow 0$
 3: $QbSet(B) \leftarrow \emptyset$
 4: update mining status
on found/received a new valid in-block $Q$
 5: $B_Q \leftarrow$ the preceding block of $Q$
 6: if I have a valid block on top of $B_Q$
    or $maxQHeight(B_Q) > h(Q) + \tau$
 7:     delete $Q$
 8: else
 9:     Broadcast $Q$ to nodes connected with me
10:     $QbSet(B_Q) \leftarrow QbSet(B_Q) \bigcup \{Q\}$
11:     if $maxQHeight(B_Q) < h(Q)$
12:         $maxQHeight(B_Q) \leftarrow h(Q)$
13:     update mining status
on update mining status
14: $B \leftarrow$ the block to mine on according to Algorithm 3
15: $W \leftarrow$ my working transaction set
16: $W \leftarrow W -$ transactions in conflict with $B$'s branch
17: $W \leftarrow W \bigcup$ transactions new to $W$ in $QbSet(B)$
    $\bigcup$ some new transactions, preferably only I know
18: mine on $B$ with transaction set $W$
```

only known to herself or a random set of newly received transactions when adjusting her working transaction set.

We shall see in Section III-B that when a block fork happens, the more in-predecessors a block has, the more likely it would be selected by miners. The rationale behind this requirement is, when the selfish miner delays publication of a block, some in-blocks only seen later in the network would not be included in her block. Therefore when a block race happens, the block mined by an honest miner would contain more in-predecessors previously seen by the network.

A malicious miner might generate a huge in-block so that any block/in-block mined after it would exceed the block size limit. Therefore the size of a in-block should be limited according to its in-height. We omit the details of this mechanism to simplify our description.

B. Main Chain Selection Policy

Definition 5. *A block/in-block A is called a* child *of a block B ($B \neq A$) if the chain containing A also contains B; block B's children set is denoted as $Ch(B)$.*

Definition 6. *The* weight *of a block B is defined as $W(B) = |Pd(B)| + |Ch(B)| + 1$, where the $||$ symbol denotes the number of elements in a set.*

In our notation, the child relation is transitive: a child of a child is also a child. The weight of a block is defined as the total number of its in-predecessors, children and itself.

Algorithm 3 Choosing Which Block to Mine on

```
1: $B \leftarrow$ the preceding block of the block fork
2: if $ChildrenBlock(B) = \emptyset$
3:     return $B$ and exit
4: $B \leftarrow \underset{C \in ChildrenBlock(B)}{\operatorname{argmax}} W(C)$
5: goto line 2
```

Our main chain selection rule is described in Algorithm 3. In short, when a block fork happens, a miner always chooses the branch whose head has a larger weight value. This is different from the current longest-chain rule, but in line with GHOST rule. The authors of [12] have proved that under GHOST rule, miners will eventually converge to the same history. They have also proved that a malicious miner with less than 50% of mining power cannot secretly create a heavier branch. Their proof is directly applicable to our variant. We now prove the resilience of our design against selfish-mine.

Proposition 1. *Assume that a selfish miner with mining power share $\alpha < 0.5$ finds a block B_1 at time T and secretly mines on top of it for time t before another block B_2 is found by an honest miner at $T + t$ and a block race happens. The probability that the selfish miner can win the block race by publishing all her blocks and in-blocks goes to zero as t goes to infinity.*

Proof: Let us consider the weight values of two competing branches when the block race happens. All public in-blocks found before T would be in-predecessors of both branches, thus they do not affect the comparison result. A secret in-predecessor of B_1 found by the selfish miner with in-height h_s would be rendered invalid as soon as a in-block with in-height $h_s + \tau + 1$ is published. Therefore, after the maximum in-height reaches $h_q(B_1) + \tau$, either the selfish miner publishes these secret in-blocks and honest blocks/in-blocks with in-height more than $h_q(B_1) + \tau + 1$ would have them as in-predecessors, or these in-blocks would be considered invalid. In both cases, they do not affect the comparison result. For the remaining parts of the branches, since the blocks and in-blocks contribute to the private chain must be children of B_1 and B_1 is kept secret, the mining power share working on the private chain is no more than α, and that of the public chain is $1-\alpha$. Therefore according to the law of large numbers, the probability that the selfish miner can win the block race becomes arbitrarily low as t grows. ■

Figure 1 illustrates a typical block race. All miners start working on the same block and the selfish miner has found two blocks before the honest miners found one. Note that a in-block with in-height 5 is found by the selfish miner and kept secret for a while. In this example, $\tau = 0$, so the selfish miner has to publish this in-block before the first in-block

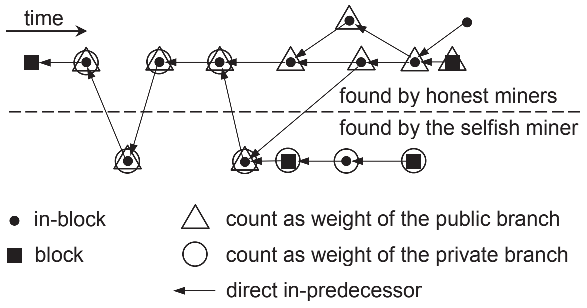
time
found by honest miners
found by the selfish miner
in-block
block
count as weight of the public branch
count as weight of the private branch
direct in-predecessor

Figure 1. A typical block race. Although the selfish miner's branch contains more blocks, the total number of blocks and in-blocks is smaller than that of the public branch.

with in-height 6 is broadcast to the network. The in-block at the top-right corner is rejected by most honest miners because it is received after the valid block. The weight value of the public branch is 10 and that of the private branch is 8, so the private branch would not be selected by honest miners.

C. Costs and Incentives

The additional computational cost of our design is negligible. To miners, in-blocks are merely by-products found during mining. Adjusting working transaction set according to newly received in-blocks would introduce a little computational overhead, however for the majority of miners, scheduling and mining are performed on different chips, if not different devices. Therefore the influence on their mining power is very little.

The current Bitcoin implementation transfers a block along with all transactions within it. This is not necessary in our scheme. A block/in-block can refer to the hash values of its direct in-predecessors instead of repeating all transactions when there is no conflicting transaction. The receiver can reconstruct and verify the block/in-block in its memory and ask for the missing in-predecessors if necessary. Note that when $\tau = 0$, these unseen in-predecessors would not be counted in the weight of the following blocks. New transactions in a block/in-block would need to be transferred anyway. Therefore by applying this technique, the only communication overhead of our scheme is the in-block headers and the hash values of their direct in-predecessors. These are only a few dozens of bytes per in-block, which is less than the smallest transaction.

There are several incentives for nodes to publish and relay in-blocks. First, by broadcasting a in-block, a miner increases the weight value of its possible future block, thus increases the chance of winning a potential block race. Second, broadcasting in-blocks can accelerate propagation of the next block, therefore reducing the occurrence of block forks. The current implementation requires a node to verify a block before broadcasting it, as a result, a major contributor to the block propagation delay is the verification time of new transactions within the block [8]. In-blocks would help synchronize most transactions before the actual block is received, therefore reducing the propagation delay of blocks. Last but not least, as we have shown, broadcasting in-blocks would stop selfish miners and help guarantee the fairness of the system, which is inline with the interests of most participants of the network.

IV. Evaluation

A. The Selfish Miner's Strategy

In this part we outline the selfish miner's strategy under our scheme. We assume the selfish miner can always learn the existence of new blocks/in-blocks immediately and broadcast her blocks/in-blocks to all nodes with no propagation delay. In reality the selfish miner can achieve that by connecting to all publicly reachable nodes.

When the selfish miner finds a in-block, it is in her best interest to keep it secret before it expires, hoping that it will only be counted in the weight of her block. Therefore such a in-block of in-height h_s will only be broadcast when the selfish miner learns that a in-block with in-height $h_s+\tau+1$ is found by an honest miner. Then the selfish miner broadcasts it before the $h_s+\tau+1$ in-block reaches any other node except its finder. When the selfish miner finds a block, she keeps tracking the weight of her private branch and the public branch. When the weight of the public branch reaches her private branch, she publishes her entire branch immediately according to Assumption 4. This is because if the next public broadcast is a block, the public branch would be strictly heavier than her private branch and no honest miner would work on her branch. Any further delay on the publication of blocks/in-blocks would lead the selfish miner to gain less than her fair share, making the selfish-mine strategy less profitable than following the protocol.

B. Simulation Settings

We implement a Bitcoin simulator to validate our theoretical result and compare with other defense mechanisms. We simulate 1000 honest miners with identical mining power whose total mining power share is $1-\alpha$, and 1 selfish miner with mining power share α. In reality the selfish miner might be a pool and suffer from internal network delay. Each block and in-block found by an honest miner can reach the selfish miner immediately, and all honest miners in 30 seconds at an even speed. This speed is an underestimation of the reality [7] and is also in favor of the attacker. The unfair propagation delay enables us to estimate a lower bound on the effectiveness of our defense. This is the first simulation to incorporate the attacker's advantage of network propagation in the context of selfish-mine.

C. The Effect of k and τ

D. Comparison With Other Defense Mechanisms

V. Discussion

A. Mitigation of Double Spending Unconfirmed Transactions

VI. Conclusion

Acknowledgment

References

[1] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar, "On bitcoin and red balloons," in *Proceedings of the 13th ACM conference on electronic commerce*. ACM, 2012, pp. 56–73.

[3] btchris, Bytecoin, mtgox, and RHorning. (2010) Mining cartel attack. [Online]. Available: https://bitcointalk.org/index.php?topic=2227

[4] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 436–454.

[5] L. Bahack, "Theoretical bitcoin attacks with less than half of the computational power (draft)," *arXiv preprint arXiv:1312.7013*, 2013.

[6] E. Heilman, "One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner." *IACR Cryptology ePrint Archive*, vol. 2014, p. 7, 2014.

[7] Bitcoin stats - data propagation. [Online]. Available: http://bitcoinstats.com/network/propagation/

[8] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), IEEE Thirteenth International Conference on*, 2013.

[9] G. O. Karame, "Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin," in *In Proc. of Conference on Computer and Communication Security*, 2012.

[10] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten, "Have a snack, pay with bitcoins," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*. IEEE, 2013, pp. 1–5.

[11] B. L. Shultz. (2015) Certification of witness: Mitigating blockchain fork attacks. [Online]. Available: http://bshultz.com/paper/Shultz_Thesis.pdf

[12] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," *Financial Cryptography and Data Security. Springer*, 2015.