Broadcasting Intermediate Blocks as a Defense Mechanism against Selfish Mining in Bitcoin

Abstract. Although adopted by many cryptocurrencies, the Bitcoin mining protocol is not incentive-compatible, as the selfish mining strategy enables a miner to gain unfair mining rewards. Existing defenses either demand fundamental changes to block validity rules or have little effect on an attacker with more than one third of the total mining power. This paper proposes an effective defense mechanism against resourceful selfish miners. Our defense requires miners to publish intermediate blocks, the by-products of the mining process, then build on each other's work. By adding transparency to the mining process, block forks are resolved by comparing the amount of work of each branch. Moreover, this mechanism has the advantages of backward compatibility, low communication and computational costs, accelerating block propagation, and mitigating double-spending attacks on fast payments. To evaluate our design, we computed the most profitable mining strategy within our defense with analysis and simulation. Our simulation showed that within our defense, a selfish miner with almost half of the total mining power can only gain marginal unfair block rewards.

1 Introduction

Bitcoin [17], a decentralized cryptocurrency system, attracts not only a lot of users but also significant attention from academia. The vitality of Bitcoin relies on an incentive mechanism named *mining*. Participants of the mining process, 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*, a ledger organized as a chain of blocks. 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. Bitcoin's designer implicitly assumed the *fairness* of this incentive mechanism: as long as more than half of the mining power follows the protocol, the chance that a miner can earn the next block reward is proportional to her computational power [17].

Unfortunately, this assumption has been disproven. In a recent paper, Eyal and Sirer highlighted an attack that enables a malicious miner to earn relatively more block rewards [11]. The attack exploits the following *fork-resolving policy* of Bitcoin: when more than one block is mined with the same preceding block and the blockchain is *forked* into a tree, a miner adopts and mines on the longest chain, or the first received block when several chains are of the same length. We refer to this forked situation as a *block race*, and to an equal-length block race as a *tie*. Eventually, all miners will convert to the same longest *main chain*, and will discard blocks that are not in the main chain. Unintentionally, this policy allows a selfish miner to deviate from the prescribed mining behavior as follows. The selfish miner keeps discovered blocks private and continues to mine on top of them, hoping to gain a larger lead on the public chain. When the public chain approaches the private chain in length, the selfish miner makes the private chain public and claims the block rewards. There are two situations in which the selfish miner wins a block race: either the block race is a tie, and the next block is mined on the selfish miner's chain; or the selfish miner's chain is longer. Accordingly, the selfish miner increases relative block rewards by causing other miners' work to be discarded.

Existing defenses focus on two approaches: lowering the chance of honest miners working on the selfish miner's blocks during a tie [13], as suggested by Eyal and Sirer [11] and Heilman [13]; or making fundamental changes to the block validity rules, as suggested by Bahack [6] and Shultz [20]. The former approach only mitigates the attack: even when no honest miner works on the selfish miner's chain during a tie, the selfish mining strategy is still more profitable as long as the selfish miner controls more than one third of the total mining power [6,11]; while the second approach would render the protocol backward incompatible. Besides, no defense considers an adversary that can alter her strategy accordingly in order to avoid losing block rewards.

This paper proposes another defense against selfish mining. We require miners to publish *intermediate blocks*, or *in-blocks* for short, which are blocks that are valid with lower puzzle difficulty, but confer no mining reward onto the miner who discovers one. When a fork happens, miners adopt the branch with the largest total amount of work, rather than the longest chain. 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.

This defense has the following advantages: (1) the block validity rules remain unchanged; (2) the extra communication and computational costs are low; (3) in-block broadcast can synchronize most transactions among all participants before the full block is mined, thus decreasing block propagation time; (4) an in-block can be seen as the miner's commitment on a set of new transactions, and such commitment contributes to double-spending mitigation on fast payments. Other contributions of this paper include: (1) analyzing how a rational selfish miner can modify her mining strategy to adapt to our defense; (2) computing the most profitable mining strategy for a wide range of mining power under the defense proposed by Eyal and Sirer [11] and our defense with Monte Carlo method; (3) presenting a simple rule to help system designers choose parameters in our defense, and verifying this rule with simulation; (4) comparing the performance of our defense with two other defenses in thwarting the adaptive selfish miner under real-world network propagation delay. Our simulation shows that a selfish miner with 45% of mining power earns only 51% of relative block rewards, compared with the 67% best result of prior defenses, given that an in-block is broadcast every 30 seconds on average.

2 Background

2.1 The Bitcoin Protocol

We summarize here some essential characteristics for our discussion and refer to Nakamoto's original paper [17] for a complete view of the system. A transaction in Bitcoin consists of at least one input and one output. An *input* is usually an output of a previous transaction. An *output* contains the receiver's public key and an amount. The difference between the total amount of inputs and outputs is called *transaction fee*, which will go to the miner who includes the transaction in the blockchain. Transaction fees are supposed to substitute mining rewards in the long run [16]. More than one transactions sharing at least one common input are called *double-spending transactions*, and at most one can get into the blockchain. The entire transaction is protected by cryptographic techniques so that only the owner of an output can spend it. Every transaction is broadcast to the entire peer-to-peer overlay network.

To ensure participants have a consensus on valid transactions, all nodes follow the same *block and transaction validity rules* and only maintain blocks in the *longest chain*. Each block contains its distance from the first block, called *height*, the hash value of the preceding block, a set of transactions, and a nonce. Information about the preceding block guarantees that a miner must choose which block to mine on before she starts mining. A special *coinbase* transaction in the block allocates some new bitcoins to the miner without any input. To construct a valid block, miners work on finding the right nonce so that the hash of the block is smaller than the *block difficulty target*. The block difficulty target 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 longest chain, the coinbase output and all transaction fees in the block belong to the miner. To decrease the variance of mining revenues, miners often form *mining pools* to work on the same puzzle and split the rewards according to their contributions.

2.2 The st_k Strategy Family

The idea of selectively delaying publication of blocks to gain an unfair advantage of block rewards appears as early as 2010 [8]. A specific strategy is given the name "selfish mining" and formally described and analyzed by Eyal and Sirer in 2013 [11]. Bahack generalized the strategy to a family of strategies st_k, where $k = 0, 1, 2, \cdots, \infty$, and showed that any optimal mining strategy that may cause other miners' blocks to be discarded must be a member of the family [6]. Specifically, st_0 is the honest strategy, and st_1 is the sefish mining strategy in [11].

Algorithm 1 describes the st_k strategy. On the premise that the selfish miner has less mining power than honest miners combined, once in a while a new honest block will surpass the height of the private chain. The selfish miner then gives up her private chain and mines 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

Algorithm 1 The st_k Strategy, $k = 0, 1, 2, \cdots, \infty$

```
on consensus
 1: $l_s \leftarrow 0$                    ▷ $l_s$ denotes the length of the private chain
 2: $l_p \leftarrow 0$                    ▷ $l_p$ denotes the length of the public chain
 3: mine on the public chain
on I found a block
 4: $l_s \leftarrow l_s + 1$
 5: if $l_s - l_p = 1$ and $l_p \geq k$ then
 6:     publish the newly mined block
 7:     goto consensus
 8: else
 9:     keep mining on the private chain
on others found a block
10: if the block is mined on top of my block then
11:     $l_s \leftarrow l_s - l_p$
12:     $l_p \leftarrow 1$
13: else
14:     $l_p \leftarrow l_p + 1$
15: if $l_s - l_p < 0$ then
16:     goto consensus
17: else if $l_s - l_p = 0$ then
18:     publish the entire private chain
19:     mine on the private chain
20: else if $l_s - l_p = 1$ and $l_p \geq k$ then
21:     publish the entire private chain
22:     goto consensus
23: else if $l_s - l_p \geq 1$ then
24:     publish the first unpublished private chain block
```

a block, she keeps mining on top of it (line 9) unless there is already a tie of length k, which we will discuss later. When a block is found by honest miners, if the public chain is longer, the selfish miner gives up and goes to consensus (line 15, 16). If both chains have the same length, the selfish miner publishes her secret chain and hopes to win a block race (line 17 to 19). 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 is at least 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 has the same length as the public chain (line 23, 24), until the lead is reduced to one and the length of the block race reaches k.

The strategies in the family differ from each other when the selfish miner's lead is one (line 5 to 7, 20 to 22). For any $l_p \geq k$, the st_k strategy publishes the latest block in the private chain and claim all the block rewards, whereas a strategy with larger k keeps the last block secret. By keeping the last block secret, the selfish miner gains more time for herself to mine on top of it before anyone else could. However she also risks losing all block rewards for the first l_p blocks in the private chain.

3 Our Defense Mechanism

No existing defense can defend against a selfish miner with mining power share $\alpha > 1/3$ without any collateral damage, because such resourceful selfish miner can abuse her luck of finding consecutive blocks to gain an unfair advantage even if she loses all ties (line 5 to 7 and 20 to 24 of Alg. 1). In this section, we present a defense mechanism that is effective against resourceful selfish miners. Our key observation is inspired by the GHOST rule proposed by Sompolinsky and Zohar [21]: 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. By demanding miners to publish and mine on each other's in-blocks, we can make the amount of computational power working on each branch visible.

Broadcasting lowered-difficulty blocks is first utilized by mining pools, such as P2Pool [3], to help distribute mining rewards among participants. Several Bitcoin developers reckoned that this mechanism can also be used for other purposes [4,22]. However, to the best of our knowledge, this is the first work to describe and analyze how this mechanism can defend against selfish mining.

3.1 Threat Model

We follow the four assumptions of most selfish mining studies [6,11,13,19,20]:

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

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

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

Assumption 3. *The selfish miner does not have the power to downgrade the propagation speed of others' blocks.*

A malicious miner can earn unfair block rewards by deploying a sybil (see Babaioff et al. [5]), DDOS (see Johnson et al. [14]), or eclipse attack (see Nayak et al. [18]). We do not consider the former two attacks because defending against them in peer-to-peer network is known to be difficult. Neither do we consider the eclipse attack because countermeasures are already implemented [1].

Assumption 4. *When the public chain becomes strictly longer, the selfish miner abandons her secret chain and starts mining on the public chain.*

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

At last, this paper only considers the strongest attacker regarding network connectivity:

Assumption 5. *The selfish miner receives and broadcasts blocks and in-blocks with no propagation delay.*

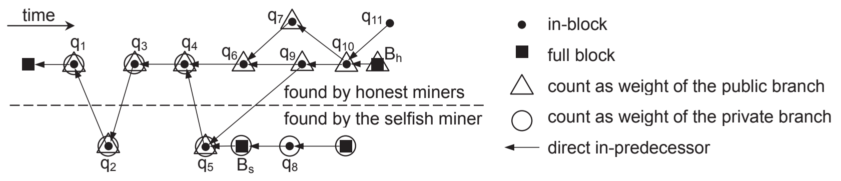

time
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
Bh
Bs
found by honest miners
found by the selfish miner
in-block
full block
count as weight of the public branch
count as weight of the private branch
direct in-predecessor

Fig. 1. A typical block race. Although the selfish miner finds two blocks before honest miners find one, honest miners would not mine after her branch because its weight $W(\mathrm{B_s}) = 8$ is smaller than the public branch weight $W(\mathrm{B_h}) = 10$.

3.2 Mining and In-Block Broadcasting

We describe our mining and in-block broadcasting mechanism here and the corresponding main chain selection rule in Sect. 3.3. In both parts we introduce some necessary definitions before presenting the algorithm.

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

Definition 2. *An in-block Q_1 is an* in-predecessor *of a block or in-block Q_2, if they have the same preceding block, and either of the following conditions holds: (1) there exists an in-block Q_3 such that Q_1 is an in-predecessor of Q_3 and Q_3 is an in-predecessor of Q_2; (2) after removing all double-spending transactions from Q_1 and Q_2, the remaining transaction set of Q_1 is a proper subset of that of Q_2. We use $Pd(Q)$ to denote the in-predecessor set of Q known by a node.*

Definition 3. *From a node's local perspective, the* in-height *of a block or in-block Q is defined as*

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

The system parameter m determines the expected number of in-blocks between two blocks. Our definition of in-predecessor can tolerate inconsistency in terms of double-spending transactions, but does not tolerate missing transactions. If Q_1 is an in-predecessor of Q_2, we can say with high certainty that Q_2 is mined after Q_1. For example, in Fig. 1, the in-height of a block or in-block is the length of its longest path to the preceding block. We have $h_q(\mathrm{q}_{10}) = 7$ and $Pd(\mathrm{q}_{10}) = \{\mathrm{q}_1 \ ... \ \mathrm{q}_7, \ \mathrm{q}_9\}$; q_8 is not an in-predecessor of q_{10} because they are mined on different blocks. The transaction set of q_{10} is a superset of the union of its in-predecessors' transaction sets, excluding double-spending transactions.

Our mining and in-block broadcasting algorithm is described in Alg. 2. Our defense requires miners to publish in-blocks they found during mining, and include the transactions of received in-blocks into the blocks they are working on. In line with [6, 11, 13], we believe nodes should broadcast all competing valid

Algorithm 2 Mining and In-Block Broadcasting

```
on found/received a new valid block $B$
    Broadcast $B$ to the network
    $maxInHeight(B) \leftarrow 0$
    $IbSet(B) \leftarrow \emptyset$                    ▷ $IbSet(B)$ is the set of in-blocks mined on $B$
    goto update mining status
on found/received a new valid in-block $Q$
    $B_Q \leftarrow$ the preceding block of $Q$
    if I have a valid block on top of $B_Q$
    or $maxInHeight(B_Q) > h(Q) + \tau$ then
        delete $Q$
    else
        Broadcast $Q$ to nodes connected with me
        $IbSet(B_Q) \leftarrow IbSet(B_Q) \bigcup \{Q\}$
        if $maxInHeight(B_Q) < h(Q)$ then
            $maxInHeight(B_Q) \leftarrow h(Q)$
        goto update mining status
on update mining status
    $B \leftarrow$ the block to mine on according to Alg. 3
    $W \leftarrow$ my working transaction set
    $W \leftarrow W -$ transactions in conflict with $B$'s branch
    $W \leftarrow W \bigcup$ transactions new to $W$ in $IbSet(B)$
    $\bigcup$ some new transactions, preferably only I know
    mine on $B$ with transaction set $W$
```

blocks (line 1 and 9), instead of just the first one they receive, as in the current implementation. This modification to the current implementation increases the detectability of selfish mining. An in-block is rejected by a miner if either of two conditions holds (line 6, 7): (1) when she has 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 an in-block of in-height h, all later-received in-blocks mined on the same block with in-height less than h are rejected. Therefore in Fig. 1, q_5 must be broadcast to the network when q_6 is mined, in order to be considered valid; q_{11} is rejected by nodes who receive it after $\mathrm{B_h}$. Every time a block or 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 has as many in-predecessors as possible (line 14 to 18). To ensure all blocks and in-blocks that have an in-block Q as their in-predecessor must be mined after Q, each miner can include some transactions only known to herself or a random set of newly received transactions when adjusting her working transaction set (line 17).

We shall see in Sect. 3.3 that in the case of a block fork, the more in-predecessors a block has, the more likely it would be selected by miners. The rationale behind this requirement is that when the selfish miner delays publication of a block, some in-blocks only seen later in the network will not be included in her block. Therefore when a block race happens, the block mined by an honest miner will contain more in-predecessors previously seen by the network.

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

3.3 Main Chain Selection Policy

Definition 4. *A block or in-block Q is called a* child *of a block B ($B \neq Q$) if the chain ends with Q contains B; block B's children set is denoted as $Ch(B)$.*

Definition 5. *The* weight *of a block B is defined as $W(B) = |Pd(B)| + |Ch(B)| + 1$, where the $|\cdot|$ 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. In Fig. 1, $W(\mathrm{B_s}) = 5 + 2 + 1 = 8$, $W(\mathrm{B_h}) = 9 + 0 + 1 = 10$.

Algorithm 3 Choosing Which Block to Mine on

```
1: $B \leftarrow$ the preceding block of the block fork
2: $Suc_B \leftarrow$ all blocks mined on $B$
3: **if** $Suc_B = \emptyset$ **then**
4:     return $B$ and exit
5: $S_{maxW} \leftarrow \{C | C \in Suc_B$ and
   $\forall C' \in Suc_B, W(C') \leq W(C)\}$
6: $B \leftarrow$ a random element in $S_{maxW}$
7: **goto** line 2
```

Our main chain selection rule is described in Alg. 3. In short, when a block fork happens, a miner chooses the branch whose earliest block, or *head*, has the largest weight value. If multiple heads have the same weight, the miner chooses randomly which branch to mine on. This is different from the current longest-chain rule, but in line with the GHOST rule. Sompolinsky and Zohar proved that under the GHOST rule, miners will eventually converge to the same history [21]. They also proved that a malicious miner with less than 50% of mining power cannot secretly create a heavier branch. Their proofs are directly applicable to our variant. We now prove the resilience of our design against selfish mining.

Proposition 1. *Assume that a selfish miner with mining power share $\alpha < 0.5$ finds a block B_s at time T after consensus, and secretly mines on top of it for time t before another block B_h 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_s found by the selfish miner with in-height h_s would be invalidated as soon as an in-block with in-height $h_s + \tau + 1$ is published. According to Assumption 3, the selfish miner cannot delay or prevent such event. Therefore, after the $h_s + \tau + 1$ in-block finishes broadcasting, for any honest miner, either she has received the h_s selfish in-block earlier and the next block or in-block she finds will have it as its in-predecessor, or she has not received it and this in-block does not count as the in-predecessor of B_s. In both cases, the secret in-block does not help the selfish miner in 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_s and B_s 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. $\square$

3.4 Costs and Incentives

Computational Cost. To miners, in-blocks are merely by-products found during mining. Adjusting working transaction set according to newly received in-blocks introduces a small computational overhead. However, for the majority of miners, scheduling and mining are performed on different chips, if not different devices entirely, hence absorbing this overhead will not damage their mining power.

Communication Cost. The current Bitcoin implementation transfers a block along with all transactions within it. This is not necessary in our scheme. A block or 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 or in-block in the memory and ask for the missing in-predecessors if necessary. New transactions in a block or 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. This constitutes only a few dozen bytes per in-block, which is shorter than the smallest transaction.

Backward Compatibility. The validity rules of Bitcoin transactions and blocks remain unchanged in our scheme. Introducing several new message types is feasible in practice; even if some nodes do not understand the messages broadcasting in-blocks, they can still participate in the network with no difficulty. The only change to the protocol is the main chain selection rule, which requires consensus among honest miners. Due to the convergence of history under the GHOST rule, non-miner nodes follow the longest-chain rule will adopt the same history eventually [21].

Miners' Incentives. First, by broadcasting an in-block, a miner increases the weight value of its possible future block, thus increasing 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 [9]. In-blocks would help synchronizing most transactions before the full blocks are received, therefore reducing their propagation delay. 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 in line with the interests of most participants of the network.

Mitigating Double-Spending on Fast Payments. An in-block can be seen as a proof that its miner has accepted the transactions in it. Therefore, as an additional benefit, our defense mechanism can mitigate the double-spending attack on fast payments. An explanation on the attack and how it can be mitigated by our defense can be found in Appendix A.

4 The Adaptive Strategy of the Selfish Miner

As the original selfish mining strategy will lose most block races in our scheme, it makes no sense for the selfish miner to insist on that strategy. The adaptive selfish mining strategy family in our scheme is described in Alg. 4.

Algorithm 4 The ast_k Strategy, $k = 0, 1, 2, \cdots, \infty$

```
on consensus
 1: $W_s \leftarrow \emptyset$        ▷ $W_s$ denotes the set of blocks and in-blocks only count as weight of the private branch
 2: $W_p \leftarrow \emptyset$ ▷ $W_p$ denotes the set of blocks and in-blocks only count as weight of the public branch
 3: mine on the public branch
on I found a block or in-block $B_s$
 4: $W_s \leftarrow W_s \bigcup \{B_s\}$
 5: if $|W_s| - |W_p| = 1$ and $|W_p| \geq k$ then
 6:     publish the entire private branch
 7:     goto consensus
 8: else
 9:     keep mining on the private branch
on others found a block or in-block $B$
10: if $B$ is mined on my block in $W_s$ then
11:     remove all published elements from $W_s$
12:     $W_p \leftarrow \emptyset$
13: for all in-block $Q \in W_s$ that will be invalidated by $B$ do
14:     publish and remove $Q$ from $W_s$
15: if $B$ is the first block of the public branch then
16:     publish but do not remove the head block and the next $|W_p|$ blocks and in-blocks from $W_s$
17: if $B$ is a block or I have at least one secret block then
18:     $W_p \leftarrow W_p \bigcup \{B\}$
19: if $|W_s| - |W_p| < 0$ then
20:     goto consensus
21: else if $|W_s| - |W_p| = 0$ then
22:     publish the entire private branch
23:     mine on the private branch
24: else if $|W_s| - |W_p| = 1$ and $|W_p| \geq k$ then
25:     publish the entire private branch
26:     goto consensus
27: else if $|W_s| - |W_p| \geq 1$ and
    the head block in the private branch is published then
28:     publish the first unpublished block or in-block
```

Most steps of the algorithm can be traced back to Alg. 1 by substituting the heights of the public and private branches with the weights of the branches.

Therefore we refer to Bahack's work [6] for an analysis on the optimality of the strategy family and only explain the differences. When the selfish miner finds an in-block on top of a public 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 when a block race happens. Therefore such an in-block of in-height h_s will not be broadcast until the adaptive selfish miner learns that a block or an in-block with in-height $h_s + \tau + 1$ is found by an honest miner. Then she broadcasts it before the block or $h_s + \tau + 1$ in-block reaches any other node except its finder (line 13, 14). If the first block after consensus is found by an honest miner, this mechanism would result in an empty private branch and a non-empty public branch, therefore the consensus state (line 17 to 20). Once a block is mined by the selfish miner, she can start working secretly on top of it. An in-block mined by an honest miner belongs only to the public branch if the private branch has a secret block (line 17, 18), because otherwise it can be an in-predecessor of both branches. Lastly, we note that when the lead of the private chain is more than 1 and the public branch has no block, there is no need for the selfish miner to publish the earliest secret block, since the honest miners are unaware that a block race is happening (line 27). Only when a competing block is found by an honest miner does the selfish miner publish the head block and some subsequent blocks and in-blocks, so that the public visible part of the private branch and public branch are of the same weight (line 15, 16). We will discuss how to choose the most profitable strategy from the family for a selfish miner with certain mining power share in Sect. 5.2.

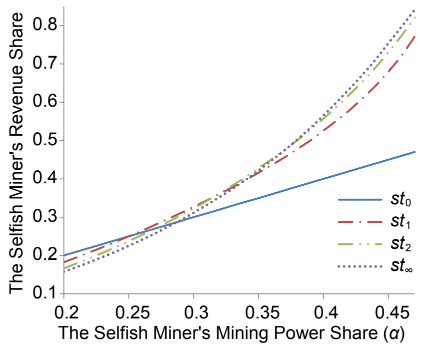
5 Simulations

We are aware that simulation is not the only way to analyze mining strategies. A formal analysis of the Bitcoin protocol is given by Garay et al. [12]. Bahack presented some analytical results on how to choose the most profitable strategy from the st_k family [6]. Concurrent to our work, Sapirshtein et al. extended Bahack's results and presented an algorithm to the find a near-optimal strategy for the selfish miner [19]. However, estimating the revenue share with simulations facilitates our discussion on the influence of different parameters and non-trivial network propagation speed.

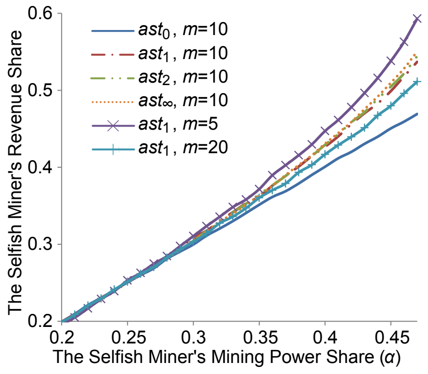
5.1 Choosing the Most Profitable Strategy from the st_k Family

In this part, we use Monte Carlo simulation to estimate the revenue share for the selfish miner with mining power share α, in order to choose the most profitable one from the st_k family. We assume the defense due to Eyal and Sirer [11] is implemented, namely honest miners choose randomly when ties happen. The results of this simulation will facilitate our discussion on choosing the most profitable strategy from the ast_k family in Sect. 5.2.

In this simulation we consider the mining process as a stationary Markov chain, and omit all propagation delay. There is one honest miner and one selfish



(a) st_k



(b) ast_k

Fig. 2. The expected revenue share of the st_k and ast_k strategy families

miner. Two transitions are possible from each state: the selfish miner finds a block with probability α, or the honest miner finds a block with probability $1-\alpha$. The longest-chain rule and heaviest-branch rule are identical in this setting since it is impossible for more than one block to be found by the honest miner at the same time. We simulate all α values between 0.20 and 0.48 with interval 0.01 for strategies st_0 to st_{10} and st_∞. For every pair of α and k we run the simulation to generate 10^7 blocks and compute the revenue share of each miner.

Figure 2(a) shows the result of four strategies within the family. The optimal strategy according to our simulation is:

$$st_{OptSim}(\alpha) = st_{\lambda(\alpha)}$$

where the value of $\lambda(\alpha)$ is defined as follows:

α	$[0.2, 0.24]$	$[0.25, 0.31]$	$[0.32, 0.35]$	$\{0.36, 0.37\}$
$\lambda(\alpha)$	0	1	2	3
α	$[0.38, 0.41]$	$\{0.42, 0.43\}$	0.44	$[0.45, 0.48]$
$\lambda(\alpha)$	4	5	8	∞

.

5.2 Choosing the Most Profitable Strategy from the ast_k Family

The selfish miner's expected revenue share differs according to the value of m, and it is infeasible to enumerate all m values. However, that does not render the problem insoluble. We have the following conjecture to simplify the problem:

Conjecture 1. *The optimal strategy from the ast_k family should follow almost the same rule as that from the st_k family.*

The reason is that as soon as the first block of the public branch is mined during a block race, the behaviors of Alg. 1 and Alg. 4 with the same k value become identical. Therefore the probability of the selfish miner winning the block race becomes the same. Although the expected revenue is different, that does not affect the comparison result among strategies.

We simulate the mining process for $m = 5, 10$ and 20 within our defense to verify our conjecture. In this simulation, four transitions are possible from each state: the selfish miner finds a block with probability α/m, the selfish miner finds an in-block with probability $\alpha \cdot (m-1)/m$, the honest miner finds a block with probability $(1-\alpha)/m$, and the honest miner finds an in-block with probability $(1-\alpha) \cdot (m-1)/m$. Miners choose the predecessors and in-predecessors of their blocks and in-blocks according to their respective strategies. Since there is no propagation delay we set $\tau = 0$. For every (m, k, α) combination we simulate 3×10^5 blocks and compute the revenue share of each miner.

Figure 2(b) shows the results of six pairs of (m, k) values. For a given m, the difference in revenue share between two strategies from the ast_k family is much smaller than that from the st_k family, which makes it harder to identify the optimal strategy. However, we can still conclude with strong confidence that $ast_{\lambda(\alpha)}$ serves as a good estimation of the optimal strategy. In most situations we have $ast_{OptSim}(m, \alpha) = ast_{\lambda(\alpha)\pm1}$. For all (m, α) pairs, the difference in revenue share between $ast_{OptSim}(m, \alpha)$ and $ast_{\lambda(\alpha)}$ never exceeds 1%. Therefore for the rest of this section we use $ast_{\lambda(\alpha)}$ as an approximation of the optimal strategy for the selfish miner within our defense. The optimal strategy of the three m values in our simulation can be found in Appendix B.

5.3 Choosing m and τ

As we have seen, as m grows, the proof-of-work of each branch is more accurately revealed, and the secret branch is less likely to win a block race. Meanwhile, as the number of in-blocks grows, the expected in-block interval decreases. The next conjecture shows the optimal value of τ when m is not too large:

Conjecture 2. *As long as the average interval between in-blocks is larger than or on the same order as the propagation delay, $\tau = 0$ yields the best performance in thwarting the selfish miner.*

The reason behind this conjecture is as follows. To the honest miners' advantage, a larger τ value would reduce unintentional rejection of in-blocks mined by honest miners. To the selfish miner's advantage, a larger τ value encourages her to further delay the publication of in-blocks so that these in-blocks have a larger probability of contributing only to the selfish miner's blocks. However, the frequency of these events differs. When the propagation delay is comparable with the interval between two consecutive honest in-blocks, the former case happens relatively rarely: if all honest miners behave correctly, only when another in-block is found at roughly the same time of the "unlucky" in-block and more than τ in-blocks are mined during the propagation of the in-block may result in its rejection by some honest miners. Whereas the selfish miner can always delay publication of her in-blocks until it would be invalidated by an honest block or in-block. Moreover, as the mining power of the selfish miner grows, the probability of the former case will decrease further, because the total number of in-blocks found by honest miners will decrease, thus increasing the expected interval and decreasing the expected overlap of in-block propagation. Therefore the smallest τ would best thwart a resourceful selfish miner.

We incorporate network propagation delay in the simulation to verify this conjecture. One simulation round represents 1 second in real time. The probability that a block is found by the network in a round is $1/600$ so that on average it takes 10 minutes to find a block. We choose $m = 20$ so that on average an in-block is found every 30 seconds, in line with P2Pool [3]. The honest mining power share is split into 1000 identical honest miners. Every honest miner manages her own blockchain, and may reject in-blocks that do not reach her in time. In terms of propagation speed for the honest miner's blocks and in-blocks, there are two different settings. First, in *slow propagation*, a block or in-block mined by an honest miner reaches all honest miners in 120 seconds at an even rate. This setting exaggerates the network connectivity advantage of the selfish miner. Second, in *real propagation*, a block or in-block mined by an honest miner reaches half of the honest miners in 5 seconds at an even rate, and the other half in the next 20 seconds at an even rate. This is a rough approximation of reality [2]. We simulate τ values ranging from 0 to 5. One simulation represents 250 days in real time. The results of our simulation can be found in Appendix C. In both settings, $\tau = 0$ performs the best in thwarting the selfish miner.

At last, we note that the conclusion may not hold when the selfish miner suffers the same network propagation delay or the expected interval between honest in-blocks is much smaller than the propagation delay. We leave this investigation to future work.

5.4 Comparison with Other Defenses

Two mining strategies are simulated with no defense as the base lines for our comparison: the st_0 strategy, namely the honest mining strategy, and the st_∞ strategy. We implement two defense strategies besides our own. The first one requires miners to choose a random branch when ties happen [11], which we refer to as *Random Branch*. The second one is *Freshness Preferred*, which requires miners to choose the block with fresher unforgeable timestamp [13] when two competing blocks are received within 120 seconds. A timestamp is released every minute as the author suggested. We do not consider the defense due to Bahack [6] and Schultz [20] because they are not backward compatible. For our own defense we use parameters $m = 10$, $\tau = 0$ so that the expected interval between two consecutive publications of in-blocks or blocks equals the interval of timestamps in Freshness Preferred. The set of parameters $m = 20$, $\tau = 0$ are also simulated. Each simulation represents 500 days in real time. Block and in-block propagation speed is the same with the real propagation mode in Sect. 5.3.

The selfish miner follows the $st_{\lambda(\alpha)}$ strategy under Random Branch. The optimal strategy under Freshness Preferred is non-trivial, we use st_1 as the base strategy and implemented the attack proposed by Schultz [20]: when a consensus block is found by honest miners, the selfish miner keeps mining on its preceding block when a fresher timestamp is available but within the 120-second window.

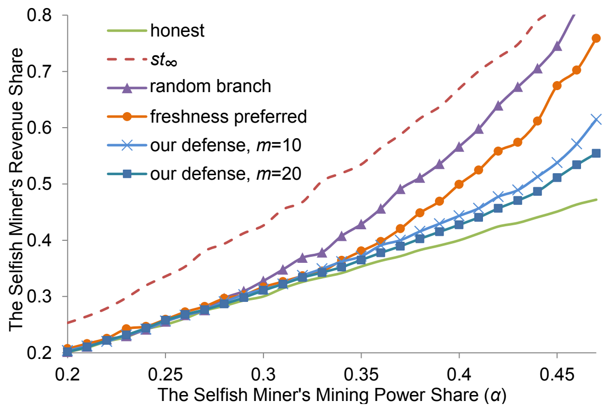


Fig. 3. Comparison with Other Defenses

Upon success, the earlier block, now stale, would be discarded by honest miners. Besides, when $\alpha \leq 0.33$ and the lead of the private branch is no more than one, she publishes the whole branch when a new timestamp is released to avoid losing a tie; when $\alpha \geq 0.34$ she bets on winning block races by having a longer chain.

The simulation results are shown in Fig. 3. It can be seen that our $m = 10$ defense outperforms the two other defenses in almost every data point. The only exception is when $\alpha = 0.27$, where Freshness Preferred earns 0.2% less block rewards than in our defense. The advantage of our defense becomes more obvious against stronger selfish miners, since under the two other defenses, the selfish miner can always win a block race when she has a longer chain.

6 Conclusion

The selfish mining attack disproves the fairness assumption in Bitcoin designer's original analysis. Furthermore, since the revenue of a malicious miner rises super-linearly with her computational power, rational miners would prefer to join their forces for a higher input-output ratio, and the decentralized structure of Bitcoin would be damaged. We observed that it is easier for a selfish miner to create a private branch with a longer chain than more amount of work. Based on the observation, this paper proposed a defense that demands miners to broadcast the by-products of their mining, so that block forks can be resolved by comparing the amount of work of each branch. This mechanism has the benefit of very low communication and computation overhead, in addition to accelerating block propagation and mitigating double-spending attack on fast payments. We provided a simple rule on choosing the parameters according to the network synchronization speed. Our simulation showed that even the most profitable mining strategy within our defense obtains only marginal unfair block rewards.

References

1. Bitcoin core version 0.10.1 released, https://bitcoin.org/en/release/v0.10.1
2. Bitcoin stats - data propagation, http://bitcoinstats.com/network/propagation/
3. P2Pool, http://p2pool.in/
4. Andresen, G.: Re: Faster blocks vs bigger blocks, https://bitcointalk.org/index.php?topic=673415.msg7658481♯msg7658481
5. Babaioff, M., Dobzinski, S., Oren, S., Zohar, A.: On Bitcoin and red balloons. In: 13th ACM Conference on Electronic Commerce. pp. 56–73. ACM (2012)
6. Bahack, L.: Theoretical Bitcoin attacks with less than half of the computational power (draft). arXiv preprint arXiv:1312.7013 (2013)
7. Bamert, T., Decker, C., Elsen, L., Wattenhofer, R., Welten, S.: Have a snack, pay with Bitcoins. In: 13th IEEE International Conference on Peer-to-Peer Computing (P2P). pp. 1–5. IEEE (2013)
8. btchris, Bytecoin, mtgox, RHorning: Mining cartel attack (2010), https://bitcointalk.org/index.php?topic=2227
9. Decker, C., Wattenhofer, R.: Information propagation in the Bitcoin network. In: 13th IEEE International Conference on Peer-to-Peer Computing (P2P) (2013)
10. Dmitrienko, A., Noack, D., Sadeghi, A., Yung, M.: On offline payments with Bitcoin (poster abstract). In: 1st Workshop on Bitcoin Research, affiliated with Financial Cryptography and Data Security. pp. 159–160 (2014)
11. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security, pp. 436–454. Springer (2014)
12. Garay, J.A., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: Analysis and applications. In: EUROCRYPT. pp. 281–310 (2015)
13. Heilman, E.: One weird trick to stop selfish miners: Fresh Bitcoins, a solution for the honest miner. Cryptology ePrint Archive, Report 2014/007 (2014), https://eprint.iacr.org/2014/007
14. Johnson, B., Laszka, A., Grossklags, J., Vasek, M., Moore, T.: Game-theoretic analysis of ddos attacks against Bitcoin mining pools. In: 1st Workshop on Bitcoin Research, affiliated with Financial Cryptography and Data Security. Springer (2014)
15. Karame, G.O.: Two Bitcoins at the price of one? double-spending attacks on fast payments in Bitcoin. In: 19th ACM Conference on Computer and Communications Security (2012)
16. Möser, M., Böhme, R.: Trends, tips, tolls: A longitudinal study of Bitcoin transaction fees. In: 2nd Workshop on Bitcoin Research, affiliated with Financial Cryptography and Data Security. pp. 19–33 (2015)
17. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008), http://www.bitcoin.org/bitcoin.pdf
18. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. Cryptology ePrint Archive, Report 2015/796 (2015), https://eprint.iacr.org/2015/796
19. Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in Bitcoin. arXiv preprint arXiv:1507.06183 (2015)
20. Shultz, B.L.: Certification of witness: Mitigating blockchain fork attacks (2015), http://bshultz.com/paper/Shultz_Thesis.pdf
21. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. Financial Cryptography and Data Security (2015)
22. Todd, P.: Near-block broadcasts for proof of tx propagation, http://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-September/003275.html

A Mitigating Double-Spending on Fast Payments

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. The difficulty of securing fast payments from double-spending attacks is highlighted by Karame and Androulaki [15], as well as Bamert et al. [7]. Guidelines are provided in these studies for merchants to lower their risks. Dmitrienko et al. designed an offline payment system to address the problem [10]. We present here our suggestion on how to utilize in-block broadcasting to mitigate the attack.

Assume the adversary wants the merchant to accept transaction A, meanwhile trying to have a conflicting transaction B eventually accepted by the network. Intuitively, the merchant can accept A and release the goods as long as she receives a certain number of in-blocks containing A, as suggested by Todd [22]. However this approach has two disadvantages: first, the adversary can perform a network-level attack on the merchant so that she only receives in-blocks containing A, as pointed out by Bamert et al. [7]; second, the adversary can set a higher transaction fee to B and publish it only after the goods are released, in the hope that rational miners would abandon A for the extra transaction fee.

Therefore we propose to add two extra rules to defend against the attack. First, a miner should stick to the first conflicting transaction she receives, regardless of the transaction fee. Admittedly, this rule might not be rational if the only goal of a miner is to maximize her rewards; however, following this rule would strengthen the security of fast payments, thus might help stabilize the future exchange rate of Bitcoin, therefore be consistent with the long-term rationality of miners. Second, a cryptographic proof on the identity of the miner should be incorporated in every block and in-block. For example, the miner can sign the merkle root of the transaction set excluding the coinbase transaction with her private key, and include this signature to the coinbase transaction. With these rules enforced, an in-block can be seen as a commitment on a certain version of a transaction by the miner. A merchant can maintain a blacklist of public keys of miners who do not stick to their commitments. Once the merchant has confirmed that the overwhelming majority of mining power is honest and has committed on A, she can deliver the goods. The mining power share of a miner associated with a public key can be calculated as the number of blocks of the last 100 blocks with the public key.

B The Optimal Strategy for $m = 5$, 10 and 20 in Our Simulation

The most profitable strategies in our simulation for all three m values are:

$$ast_{OptSim}(m, \alpha) = ast_{\hat{\lambda}(m,\alpha)}$$

where

α	$[0.2, 0.24]$	$\{0.25, 0.27, 0.3, 0.31\}$	$\{0.26, 0.28, 0.29, 0.33, 0.38\}$
$\hat{\lambda}(\ 5, \alpha)$	0	1	2

α	$\{0.34, 0.36\}$	$\{0.32, 0.37\}$	$\{0.35, 0.39, 0.41, 0.42\}$	$[0.43, 0.46]$
$\hat{\lambda}(\ 5, \alpha)$	3	4	5	8

α	0.4	$\{0.47, 0.48\}$
$\hat{\lambda}(\ 5, \alpha)$	9	10

α	$[0.2, 0.25]$	$[0.26, 0.34]$	$\{0.36, 0.37\}$	$\{0.35, 0.38, 0.39, 0.41, 0.42\}$
$\hat{\lambda}(10, \alpha)$	0	1	3	4

α	0.4	0.45	$\{0.43, 0.44\}$ and $[0.46, 0.48]$
$\hat{\lambda}(10, \alpha)$	5	6	7

α	0.21	$\{0.2, 0.26, 0.3, 0.32, 0.35, 0.36\}$ and $[0.22, 0.24]$
$\hat{\lambda}(20, \alpha)$	0	1

α	$\{0.25, 0.34, 0.37, 0.39\}$ and $[0.27, 0.29]$	$\{0.31, 0.33\}$	$\{0.4, 0.41\}$
$\hat{\lambda}(20, \alpha)$	2	3	4

α	0.38	0.42	0.45	$\{0.43, 0.44\}$ and $[0.46, 0.48]$
$\hat{\lambda}(20, \alpha)$	5	6	9	∞

We believe that the unordered data points should be attributed to random error.

C The influence of τ in Different Network Settings

The experimental setting can be found in Sect. 5.3.

Table 1. The Selfish Miner's Expected Revenue Share in Slow Propagation

$\alpha\backslash\tau$	0	1	2	3	4	5
0.25	0.2697	0.2741	0.2700	0.2788	0.2802	0.2756
0.30	0.3227	0.3248	0.3260	0.3317	0.3290	0.3373
0.35	0.3869	0.3902	0.3941	0.3939	0.3997	0.3992
0.40	0.4521	0.4579	0.4598	0.4644	0.4705	0.4755
0.45	0.5312	0.5420	0.5529	0.5563	0.5581	0.5586

Table 2. The Selfish Miner's Expected Revenue Share in Real Propagation

$\alpha\backslash\tau$	0	1	2	3	4	5
0.25	0.2574	0.2583	0.2564	0.2562	0.2452	0.2583
0.3	0.3100	0.3086	0.3101	0.3115	0.3156	0.3102
0.35	0.3648	0.3693	0.3682	0.3731	0.3917	0.3698
0.4	0.4275	0.4340	0.4332	0.4319	0.4373	0.4333
0.45	0.5056	0.5127	0.5197	0.5246	0.5276	0.5146