Blockchain Incentive Design and Analysis

Jianyu Niu and Chen Feng

Abstract The Bitcoin white paper introduced the blockchain technology to realize a decentralized electronic cash system that does not rely on a central authority. A major novelty behind the technology is the incentive design, in which participating nodes obtain rewards by creating blocks in a longest chain. The incentive design is paramount for a secure blockchain system as shown in many recent works. In this chapter, we take a close look at the incentive design of three influential blockchain protocols including Bitcoin, Ethereum, and Bitcoin-NG. For each protocol, we present the potential incentive-based attacks and go through several theoretical results to characterize the impact of these attacks. We hope that at the end of our journey, our readers can have a deeper understanding of blockchain incentive designs and analysis.

1 Introduction

In 2008, Nakamoto invented Bitcoin in the seminal paper titled "Bitcoin: A Peer-to-Peer Electronic Cash System" [24]. Bitcoin is aimed to realize a decentralized electronic cash system that does not rely on a central authority for currency issuance and transaction processing. One year later, the Bitcoin network was launched based on the open-source code developed by Nakamoto. By now, as the biggest cryptocurrency, Bitcoin has a market value of more than 1 trillion US dollars by the dollar/bitcoin exchange rate. The huge success of Bitcoin is largely owing to two technological innovations: the so-called *Nakamoto consensus* (NC) protocol, which can realize a

J. Niu (✉)
Southern University of Science and Technology, Shenzhen, China
e-mail: niujy@sustech.edu.cn

C. Feng
The University of British Columbia (Okanagan Campus), Kelowna, Canada
e-mail: chen.feng@ubc.ca

D. A. Tran et al. (eds.), *Handbook on Blockchain*, Springer Optimization and Its Applications 194, https://doi.org/10.1007/978-3-031-07535-3_4

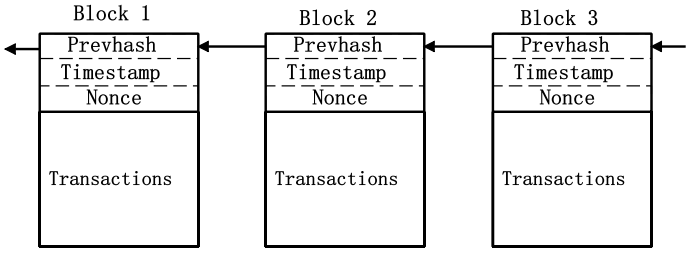

Fig. 1 An illustration of the blockchain data structure. Each block contains a batch of transactions, a hash value of its previous block, and other metadata

public, immutable, and distributed ledger widely known as a blockchain today, and the incentive design, by which nodes obtain rewards from participation. NC protocol has been discussed in other chapters. In this chapter, we will focus on the blockchain incentive design and analysis. This chapter is based closely on our previous work of incentive analysis for Ethereum [12] and Bitcoin-NG [27] as well as the first author's Ph.D. dissertation.

In a blockchain, users can create and exchange transactions to modify their monetary amounts, and new transactions can be batched into blocks for processing. Particularly, each block is linked to its previous block with a cryptographic hash, forming a chain of blocks accepted by the users (which explains the name of blockchain). A simple illustration of the blockchain data structure in Bitcoin is provided in Fig. 1. In addition to transactions and hash reference, each block also contains a timestamp, nonce (i.e., a random string), and other metadata. Here, the nonce is produced by solving computational puzzles, which is often known as Proof of Work (PoW). The searching process for the valid nonce is also called mining, and the participants are called miners. Miners follow the longest chain rule (LCR) to reach an agreement on an increasing sequence of blocks. That is, miners always extend the longest chain that they have received. Informally speaking, LCR together with PoW form the Nakamoto consensus. Despite the simplicity, NC can make blockchains securely work as long as more than half of the computation power is controlled by honest miners who follow the protocol.

When participating in NC-based blockchains, miners have to pay for the computing hardware (e.g., CPU, GPU, or ASIC), electricity, and other fees. The economic cost makes it impractical for any miners to voluntarily support the protocol. To solve this dilemma, a (public) blockchain system often relies on its incentive design, by which miners can receive a block reward, i.e., some amount of self-issued tokens, for every block included in the blockchain. In addition, miners can also get the transaction fees for all the transactions that are contained in the block [24]. These rewards can pay for the mining cost, making mining profitable and incentivizing miners to contribute as much computation power as possible. Here blockchain protocol adopts an implicit assumption that all the miners are individually rational [16, 24]. As stated previously, the security of NC relies on the assumption that the majority of computation power is

controlled by honest miners. To guarantee the rationality of being honest, NC should ensure *incentive compatibility*, i.e., miners will suffer from economic loss whenever they deviate from the protocols. In other words, without incentive compatibility, rational miners would deviate from the protocol to obtain higher revenue such that the above assumption does not hold, and, consequently, the system security will be threatened. All of these show the importance of the blockchain incentive design. In this chapter, we take a journey into blockchain incentive design and analysis, with a focus on three influential blockchain protocols: Bitcoin, Ethereum, and Bitcoin-NG. Ethereum and Bitcoin-NG are two variants of Bitcoin; both of them are designed based on NC and the incentive design of Bitcoin. For each protocol, we present the potential incentive-based attacks and go through several theoretical results to show the impact of these attacks. We hope that at the end of our journey, readers can have a deeper understanding of blockchain incentive designs and analysis.

The rest of this chapter is organized as follows. In Sect. 2, we present incentive design and analysis of Bitcoin, particularly on the selfish mining analysis from [11]. We then provide incentive analysis of Ethereum with a focus on comparing its reward design with Bitcoin's in Sect. 3 and conduct a comprehensive incentive analysis of Bitcoin-NG in Sect. 4. We provide further readings on blockchains' incentive design and analysis in Sect. 5 and conclude this chapter in Sect. 6.

2 Incentive Design and Analysis in Bitcoin

In the Bitcoin white paper, miners without coordination are suggested to follow the protocol, i.e., obeying NC to mine blocks and immediately publishing their blocks [24]. In this way, all miners are believed to be fairly rewarded; they obtain revenue in proportion to their computation power. This also encourages miners to devote more computation power to protocol. However, in reality, miners join in mining pools and share rewards to maintain stable incomes. Nowadays, the top six mining pools in Bitcoin control almost 75% of the total computation power.[1] The trend of centralization in computation power is not only against the goal of Bitcoin (i.e., a decentralized electronic cash system) but also raises concerns for incentive issues. Particularly, many studies show that a set of colluding miners who deviate from the protocol may obtain a revenue larger than their fair share from honest mining [11, 15, 17, 25, 33]. Such behavior is called *selfish mining* and the corresponding miners are called the selfish miners.

[1] Mining pool statistics: https://btc.com/stats/pool.

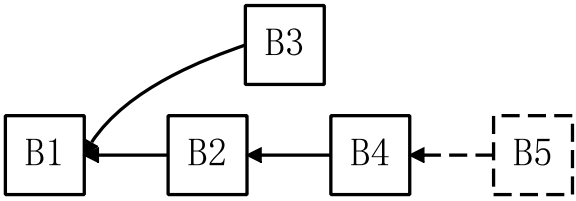

Fig. 2 An illustration of the chain structure stored in a miner. The miner tries to produce a block on the longest chain it observes

2.1 Overview of Bitcoin

Bitcoin relies on Nakamoto Consensus (NC) to make a group of distributed and mutually distrusting participants reach an agreement on a transparent and immutable ledger, also known as a blockchain. A blockchain is a list of blocks linked by hash values with each block containing a batch of ordered transactions. To make all participants agree on the same chain of blocks, NC leverages two components: the Proof-of-Work (PoW) mechanism and the longest chain rule (LCR). Each participant in NC (also referred to as a miner) collects valid and unconfirmed transactions from the network, then orders and packs these transactions into a block. A valid block needs to contain a proof of work, i.e., its owner needs to find a value of the nonce (i.e., a changeable data filed) such that the hash value of this block has required leading zeros [24]. The length of leading zeros is also known as the mining difficulty, which determines the probability to find a valid nonce of each computation try. Besides, the mining difficulty can be tuned by the system to make sure new blocks are mined every 10 min on average.

Once a new block is produced, it will be immediately broadcasted to the entire network. Ideally, the block should be accepted by all participants before the next block is produced. In reality, two new blocks might be mined around the same time, leading to a fork in which two “child” blocks share a common “parent" block, see Fig. 2 for an illustration. To resolve such a fork, an honest miner always accepts the longest chain as the valid one and mines after the last block of it. Particularly, if multiple chains have the same length, miners choose to mine after the one that it has first received. The forks do not last forever because the longest branch will win the competition and be accepted by all miners. The common prefix of all the longest chains of miners is called the *main chain*. In Bitcoin, a block miner will receive a block reward (if its block is eventually included in the system main chain) as well as transaction fees as another type of reward. These rewards encourage miners to devote their computational resources to the system.

2.2 Selfish Mining in Bitcoin

The selfish mining idea was first proposed in the Bitcoin forum [15]. Later, Eyal and Sirer formally describe and analyze the selfish mining attack [11]. In such an

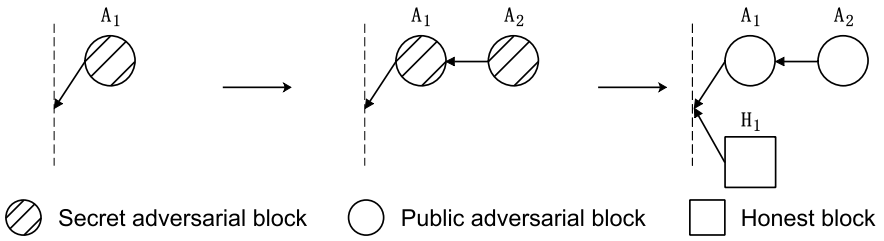

Fig. 3 The first example of the selfish mining attack in Bitcoin

attack, selfish miners first keep their newly discovered blocks private, creating a hidden branch. When some honest miners mine a new block, the selfish miners will immediately publish some private blocks, trying to make its branch the longest one. In this way, some honest miners' blocks will be abandoned (for not included in the longest chain), and the selfish miners could obtain a higher fraction of the block rewards than that from honest mining. Let us use two examples to better illustrate how the attack works. Figure 3 shows a case in which selfish miners first successfully mine two consecutive blocks A_1 and A_2 on the longest chain, and then it publishes these two blocks when receiving a block H_1 from an honest miner. In this way, the selfish miners will not only get two-block rewards (for blocks A_1 and A_2) but also make the honest block H_1 abandoned by all the miners. If the selfish miners can always replay this case, they certainly can obtain a higher fraction of block rewards than that from honest mining.

However, the selfish miners don't always take such a lead of two blocks and make their blocks accepted by all miners like the above case. Figure 4 shows a case in which the selfish miners successfully mine a block A_1 and then publish this block to match an honest block H_1. These two blocks form two forking branches of the same length, and miners cannot decide which one is the longest. To resolve the forking competition, miners shall continue mining and wait until a longer one wins. As said previously, under such a case, honest miners will mine after the first branch that it has received, whereas the selfish miners will mine after their branch. Therefore, the next block can be further divided into three subcases:

(a) The selfish miners produce the next blocks A_2 after their block and publish it. By LCR, all selfish miners' blocks are accepted by honest miners, whereas honest block H_1 is abandoned.
(b) Some honest miners produce the next block H_2 after the previous honest block H_1. If the selfish miners accept these two honest blocks H_1 and H_2 and mine after them, its block A_1 will be abandoned.
(c) Some honest miners produce the next block H_2 after the selfish miners' block A_1. Both blocks A_1 and H_2 will be accepted, whereas the block H_1 will be abandoned by LCR.

From this example, we can find that selfish miners' first block A_1 has the risk to be abandoned (shown in Fig. 4c). In other words, the selfish miners may suffer the loss of

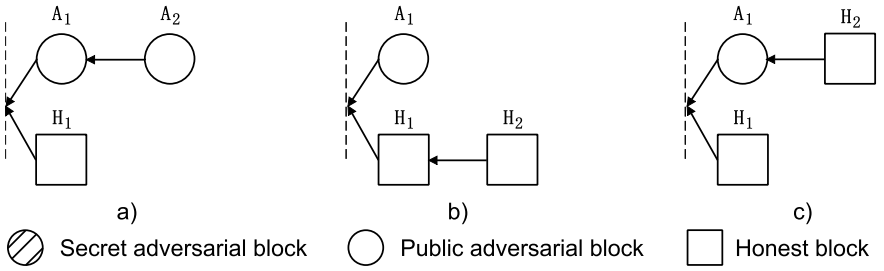

Fig. 4 The second example of the selfish mining attack in Bitcoin

block rewards in some cases, and so the attack is not always profitable. Moreover, we can see that the success of making the selfish mining attack profitable is determined by two parameters: the fraction of computation power owned by the selfish miners (denoted by α) and the fraction of honest computation power working on the selfish miners' blocks during a tie (denoted by γ). When α increases, the possibility for the selfish miners to take a lead of two blocks and to win the forking race during a tie also increases (in which the selfish miners have no loss). On the other hand, when γ is larger, the possibility that the selfish miners' blocks are abandoned is smaller. Particularly, when $\gamma = 1$ (all honest miners first receive the selfish miners' block A_1), the selfish miners' block A_1 is always accepted by all miners.

The above examples just illustrate several cases in the selfish mining attack. We can further design a complete selfish mining strategy covering all cases. In the following section, we will go through the selfish mining strategy in [11] and present the associated theoretical results. Also, note that the above strategy is not optimal under different α and γ. For example, in Fig. 4b, when α is small, the optimal strategy for the selfish miners is to accept these two honest blocks. But, when α is large, the optimal strategy for the selfish miners may be mining on its block A_1, trying to catch up (also called stubborn mining [25]). We refer our readers who are interested in the optimal selfish mining strategy to [17, 25, 33] for more details. Note that since the block generation rate is kept at a block every 10 min due to the mining difficulty adjustment, the number of mined blocks (and block rewards) is fixed during a period with or without attacks. Therefore, the higher fraction of rewards, the higher the mining revenue.

2.3 Theoretical Results on Selfish Mining in Bitcoin

Let us begin with the system model of selfish mining in Bitcoin. The system contains two types of miners: the honest, who follow the protocol, and the selfish, who may deviate from the protocol to maximize their profit. Let α denote the fraction of computation power controlled by selfish miners and β denote the fraction of total computation power controlled by honest miners. Without loss of generality, all selfish

miners are assumed to be controlled by a single selfish miner. The mining process is modeled as a Poisson process with rate f. Accordingly, the selfish miner generates blocks at rate αf, and the honest miners generate blocks at rate βf. Blocks produced by honest miners are referred to as *honest* blocks, and blocks produced by the selfish miner are referred to as *adversarial* blocks.

The time an honest block takes to arrive at all miners is assumed to be negligible in the following analysis. This is because the block interval (i.e., 10 min on average in Bitcoin) is much larger than the propagation delay (usually tens of seconds). As introduced previously, γ is used to denote the ratio of honest miners that are mining on blocks produced by the selfish miner (rather than by the honest miners) whenever they observe a fork of two branches of equal length. Here, γ captures the selfish miner's communication capability and is assumed to be in the range $[0, 1]$. What is more, since transaction fees in a block are usually much smaller than the block reward, they are not considered in the following analysis.

Next, we present the selfish mining strategy in [11], as shown in Algorithm 1. We use $L_s(t)$ (resp., $L_h(t)$) to denote the length of the private branch (resp., public branches) seen by the selfish miner (resp., honest miners) at time t. In the beginning, we assume that the selfish miner and honest miners have the consensus of the same chain (lines 1). When the selfish miner mines a new block (see lines 2 to 8), it will

Algorithm 1 The selfish mining strategy in bitcoin

on Consensus
1: $(L_s, L_h) \leftarrow (0, 0)$

on The selfish miner mines a new block
2: $L_s \leftarrow L_s + 1$
3: **if** $(L_s, L_h) = (2, 1)$ **then**
4: publish its private branch
5: $(L_s, L_h) \leftarrow (0, 0)$ (since all the miners achieve a consensus)
6: **else**
7: keep mining on its private branch
8: **end if**

on Some honest miners mine a new block
9: $L_h \leftarrow L_h + 1$
10: **if** $L_s < L_h$ **then**
11: keep mining on this new block
12: $(L_s, L_h) \leftarrow (0, 0)$
13: **else if** $L_s = L_h$ **then**
14: publish the block of the private branch
15: $(L_s, L_h) \leftarrow (1, 1)$
16: **else if** $L_s = L_h + 1$ **then**
17: publish its private branch
18: $(L_s, L_h) \leftarrow (0, 0)$ (since all the miners achieve a consensus)
19: **else**
20: publish first unpublished block in its private branch
21: set $(L_s, L_h) = (L_s - L_h, 0)$
22: **end if**

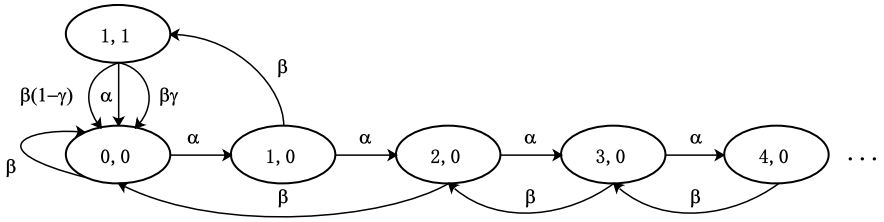

Fig. 5 The Markov process of the selfish mining in Bitcoin

keep this block private and continue mining on its private branch until its advantage is very limited (i.e., $(L_s, L_h) = (2, 1)$) which will be discussed later.

When some honest miners mine a new block, the length of a public branch will be increased by 1. We have the following cases. Case (1) If the new public branch is longer than the private branch, the selfish miner will adopt the public branch and mine on it. (That is why the selfish miner will set $(L_s, L_h) = (0, 0)$.) Case (2) If the new public branch has the same length as the private branch, the selfish miner will publish its private block immediately hoping that as many honest miners will choose its private branch as possible (since honest miners will see two branches of the same length when the private branch is published). Case (3) If the new public branch is shorter than the private branch by just one, the selfish miner will publish its private branch so that all the honest miners will adopt the private branch. Case (4) If the new public branch is shorter than the private branch by at least two, the selfish miner will publish the first unpublished block since the selfish miner still has a clear advantage. In addition, as the selfish miner will eventually win the competition (all its blocks are in the longest chain), (L_s, L_h) is set to $(L_s - L_h, 0)$.

With the above selfish mining strategy, we can use $(L_s(t), L_h(t))$ to capture the system state at time t. It is easy to verify that $(L_s(t), L_h(t))$ evolves as a Markov process under our selfish mining strategy, as illustrated in Fig. 5.[2] Moreover, the process $(L_s(t), L_h(t))$ is positive recurrent and so it has a unique stationary distribution. Let $\{\pi_{i,j}\}$ be the steady-state distribution of the Markov process $(L_s(t), L_h(t))$. By solving the above Markov process, we can compute the stationary distribution as shown in the following lemma.

Lemma 1 ([11]) *Given the selfish mining strategy in Algorithm 1, the stationary distribution of the states are*

[2] In Eyal and Sirer [11], the system states of the selfish mining process is denoted by using only $L_s(t)$. Here, we use $(L_s(t), L_h(t))$ as system states to keep consistent with that in Ethereum.

$$\begin{aligned}
\pi_{0,0} &= \frac{1-2\alpha}{2\alpha^3-4\alpha^2+1}, \\
\pi_{1,0} &= \frac{\alpha-2\alpha^2}{2\alpha^3-4\alpha^2+1}, \\
\pi_{1,1} &= \frac{(1-\alpha)(\alpha-2\alpha^2)}{1-4\alpha^2+2\alpha^3}, \\
\pi_{i,0} &= \left(\frac{\alpha}{1-\alpha}\right)^{k-1}\frac{\alpha-2\alpha^2}{1-4\alpha^2+2\alpha^3}, \ \textit{for } i \geq 1,
\end{aligned} \tag{1}$$

With the stationary distribution of the state space, we next can analyze revenues obtained by the selfish miner and by the honest miners, respectively. To achieve this, we first need to determine the rewards for published blocks in each state transition. For example, given the state transition from $(2, 0) \rightarrow (0, 0)$, there are one honest block and two adversarial blocks that are published. By LCR, the selfish miner will obtain two-block rewards, whereas honest miners obtain nothing (see the case in Fig 3). For brevity, we don't provide the reward analysis and refer interested readers to [11] for more details. Finally, we can derive the relative revenue for the selfish miner in the following theorem.

Theorem 1 ([11]) *Given the selfish mining strategy in Algorithm 1, the relative revenue for the selfish miner is* $\frac{\alpha(1-\alpha)^2(4\alpha+\gamma(1-2\alpha))-\alpha^3}{1-\alpha(1+(2-\alpha)\alpha)}$.

When the selfish miner's revenue given in Theorem 1 is larger than α, the selfish miner will earn more from the selfish mining strategy than that from honest mining. Here, we usually assume $0 \leq \alpha < 1/2$. Since when $\alpha \geq 1/2$, the selfish miners may launch the famous double-spending attack, by which it can gain more [24]. We can derive the range of α for making selfish mining profitable in the following corollary.

Corollary 1 ([11]) *For a given γ, the selfish miner with α fraction of computation power can obtain a revenue larger than its honest mining in the following range:*

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

To better understand Theorem 1 and Corollary 1, we first plot the selfish miner's relative revenue for different γ with α ranging from 0 to 0.5 in Fig. 6a. As stated previously, the selfish miner has the risk of losing block rewards when it has only one private block and publishes this block to match an honest block. In this case, if the selfish miner always propagates its blocks more quickly than honest miners (i.e., $\gamma = 1$), all miners will mine on the selfish miner's block. This means that a selfish miner takes no risk when launching the selfish mining strategy and so the selfish miner with any fraction of computation power can benefit by launching the selfish mining strategy. The minimum fraction of computation power (referred to as

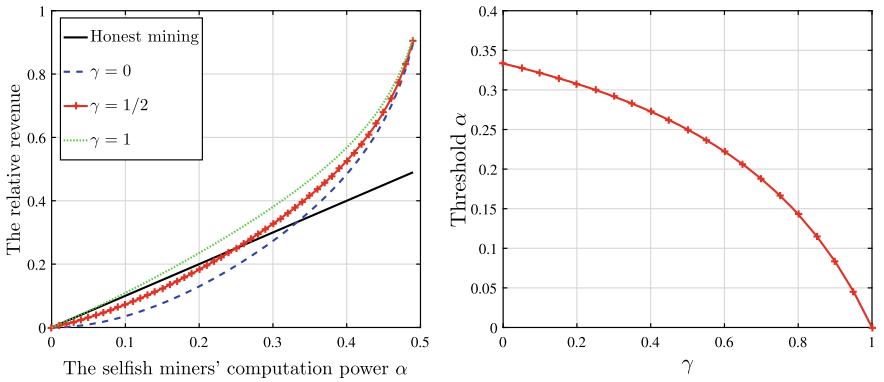

Fig. 6 The results of the selfish mining attack in Bitcoin. **a** The selfish miner's relative revenue with different α. **b** The threshold of the selfish minimg attack with different γ

threshold) making the selfish mining profitable is zero. contrary, when $\gamma = 0$ (honest miners always publish and propagate their block first), the threshold is at $1/3$. Hence, a selfish miner with more than one-third fraction of computation power can gain more revenue from the selfish mining attack. The threshold of selfish mining strategy is shown in Fig. 6b.

3 Incentive Design and Analysis in Ethereum

Ethereum is the second largest cryptocurrency by market capitalization and today's biggest decentralized platform that runs smart contracts. Ethereum currently uses a variant of NC as its underlying consensus but has a different reward design with Bitcoin by providing two additional uncle and nephew rewards.[3] In this section, we focus on studying the selfish mining attack in Ethereum, especially on analyzing the impact of these two rewards.

3.1 Overview of Ethereum

Ethereum is a distributed blockchain-based platform that runs smart contracts. Roughly speaking, a smart contract is a set of functions defined in a Turing-complete environment. The users of Ethereum are called clients. A client can issue transactions to create new contracts, to send Ether (internal cryptocurrency of Ethereum) to contracts, to other clients, or to invoke some functions of a contract. Clients' transactions are then collected into blocks by miners. Miners in Ethereum adopt a variant of NC

[3] Ethereum plans to gradually replace PoW with Proof of Stake (PoS).

Table 1 Mining rewards in Ethereum and Bitcoin

	Ethereum	Bitcoin	Purpose
Block reward	✓	✓	Compensate for miners' mining cost
Uncle reward	✓	×	Reduce centralization trend of mining
Nephew reward	✓	×	Encourage miners to reference uncle blocks
Transaction fee (gas cost)	✓	✓	Transaction execution; Resist network attack

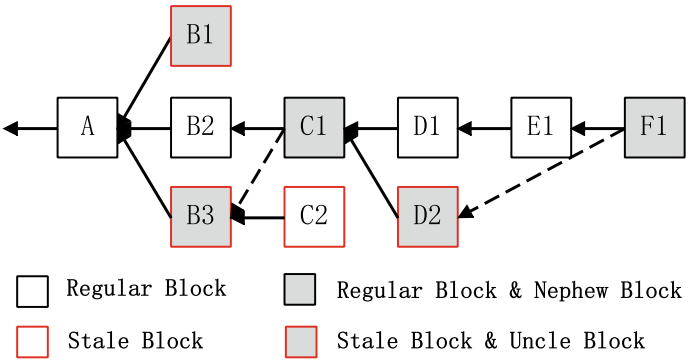

Fig. 7 Different block types in Ethereum. Here, regular blocks include $\{A, B2, C1, D1, E1, F1\}$ and stale blocks include $\{B1, B3, C2, D2\}$. Similarly, uncle blocks are $\{B1, B3, D2\}$ and nephew blocks are $\{C1, F1\}$. Uncle block $B3$ (uncle block $D2$, resp.) is referenced with distance one (two, resp.)

to reach an agreement on a growing sequence of blocks, i.e., blockchain.[4] Besides, Ethereum differs from Bitcoin in parameter settings; Ethereum sets its block size to about 20 KB and block interval to 10–20 s rather than 1 MB block size and 10 min block interval in Bitcoin for higher transaction throughput.

There are four types of rewards in Ethereum, namely, gas cost, block reward, uncle reward, and nephew reward [5, 9], as outlined in Table 1. On the one hand, Ethereum provides gas cost and block rewards, which are similar to rewards in Bitcoin. Specifically, the gas cost is used to reward miners to include and execute transactions in blocks, which is equivalent to transaction fees in Bitcoin. Also, like Bitcoin, miners in Ethereum can get block rewards, once their blocks are included in the blockchain accepted by all miners. On the other hand, Ethereum introduces two new rewards: uncle and nephew rewards.

[4] Although Ethereum claimed to apply the heaviest subtree rule [35], it seems to apply the longest chain rule instead to choose the main chain [17].

To explain these rewards, we shall introduce the concepts of *regular* and *stale* blocks. A block is called regular if it is included in the main chain, and is called stale block otherwise. In addition, an uncle block is a stale block that is a "direct child" of the main chain. In other words, the parent of an uncle block is always a regular block. An uncle block receives certain rewards if it is referenced by some future regular block, called a nephew block, through the use of reference links. See Fig. 7 for an illustration of uncle and nephew blocks. The values of uncle rewards depend on the "distance" between the uncle and nephew blocks. This distance is well defined because all the blocks form a tree. For instance, in Fig. 7, the distance between uncle block $B3$ (uncle block $D2$, resp.) and its nephew block is 1 (2, resp.). In Ethereum, if the distance is 1, the uncle reward is $\frac{7}{8}$ of the (static) block reward; if the distance is 2, the uncle reward is $\frac{6}{8}$ of the block reward; and so on. Once the distance is greater than 6, the uncle reward will be zero. By contrast, the nephew reward is always $\frac{1}{32}$ of the block reward.

3.2 Reward Design and Its Impact on Selfish Mining

In Ethereum, uncle and nephew rewards are initially designed to solve the mining centralization bias—miners form or join in some big mining pool. The mining pools with huge computation power are less likely to generate stale blocks and can be more profitable for mining. Thus, rewarding stale blocks can reduce the mining pools' advantage [8] and make them less attractive for small miners.

Unfortunately, uncle and nephew rewards also reduce the cost of launching selfish mining, and consequently lower the system security level. To see this, let us use one example to illustrate how the selfish miner can benefit from these rewards. Recall the case in which the selfish miner first produces one block, and then honest miners produce two subsequent blocks, as shown in Fig. 8a. In Bitcoin, the selfish miner will accept honest blocks H_1 and H_2 and lose the reward of block A_1 by the selfish mining strategy in Algorithm 1. By contrast, in Ethereum, this adversarial block A_1 can be referenced by the subsequent honest block A_2, by which the selfish miner can receive an uncle reward ($7/8$ of the block reward since the block distance is one), as shown in Fig. 8b. The additional uncle rewards reduce the loss of the selfish miner. Furthermore, the selfish miner can also obtain nephew rewards by referencing uncle blocks. In the next section, we will go through analytical results in [12], which systematically demonstrate the impact of these rewards on the selfish mining attack.

3.3 Theoretical Results on Selfish Mining in Ethereum

We follow the model introduced in Sect. 2.3. Let α denote the fraction of computation power controlled by selfish miners and β denote the fraction of total computation

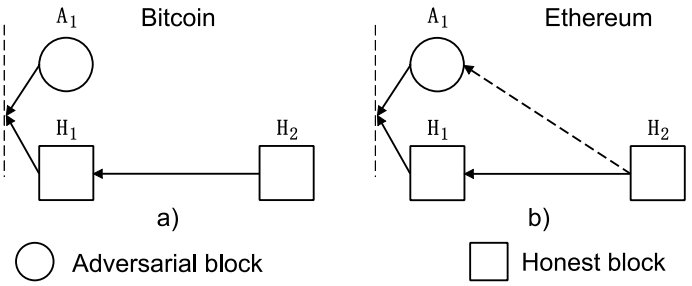

Fig. 8 A simple comparison of rewards obtained by the selfish miner in Bitcoin and Ethereum

power controlled by honest miners. Without loss of generality, all selfish miners are assumed to be controlled by a single selfish miner. The mining process is modeled as a Poisson process with rate f. Accordingly, the selfish miner generates blocks at rate αf, and the honest miners generate blocks at rate βf. We use K_s, K_u, and K_n to denote static, uncle, and nephew rewards, respectively. Without loss of generality,

Algorithm 2 A selfish mining strategy in ethereum

on Consensus
1: $(L_s, L_h) \leftarrow (0, 0)$

on The selfish miner mines a new block
2: reference all (unreferenced) uncle blocks based on its private branch
3: $L_s \leftarrow L_s + 1$
4: **if** $(L_s, L_h) = (2, 1)$ **then**
5: publish its private branch
6: $(L_s, L_h) \leftarrow (0, 0)$ (since all the miners achieve a consensus)
7: **else**
8: keep mining on its private branch
9: **end if**

on Some honest miners mine a new block
10: The miner references all (unreferenced) uncle blocks based on its public branches
11: $L_h \leftarrow L_h + 1$
12: **if** $L_s < L_h$ **then**
13: $(L_s, L_h) \leftarrow (0, 0)$
14: keep mining on this new block
15: **else if** $L_s = L_h$ **then**
16: publish the last block of the private branch
17: **else if** $L_s = L_h + 1$ **then**
18: publish its private branch
19: $(L_s, L_h) \leftarrow (0, 0)$ (since all the miners achieve a consensus)
20: **else**
21: publish first unpublished block in its private branch
22: set $(L_s, L_h) = (L_s - L_h + 1, 1)$ if the new block is mined on a public branch that is a prefix of the private branch
23: **end if**

we assume that $K_s = 1$ so that K_u (K_n, resp.) represents the ratio of uncle reward (nephew reward, resp.) to the static reward. In particular, $K_n < K_u < 1$ and K_u is a function of the distance.

We now describe the mining strategies for honest and selfish miners in Algorithm 2, which is based on the selfish mining strategy of Bitcoin in Algorithm 1. Similarly, we use $L_s(t)$ ($L_h(t)$) to denote the length of the private branch (resp., public branches) seen by the selfish miner (resp., honest miners) at time t. The main difference between Algorithms 1 and 2 is that when mining blocks, miners will include as many reference links as possible to (unreferenced) uncle blocks such that they can gain as many uncle and nephew rewards as possible (see Line 2 and 10). In addition, to track uncle and nephew rewards won by the selfish miner and honest miners, the lengths of private and public branches are kept (see Line 21 and 22). We refer our readers to [26] for more details.

With the algorithm, we can use $(L_s(t), L_h(t))$ to capture the system state at time t. The states $(L_s(t), L_h(t))$ evolve as a Markov process under the selfish mining strategy, as illustrated in Fig. 9. By comparing with Markov process of the selfish mining attack in Fig. 5, it is easy to see the increased complexity in system states because of the uncle and nephew rewards. Similarly, the process $(L_s(t), L_h(t))$ is positive recurrent and so it has a unique stationary distribution. Let $\{\pi_{i,j}\}$ be the steady-state distribution of the Markov process $(L_s(t), L_h(t))$. By solving the Markov process, we have the stationary distribution of states in the following lemma.

Lemma 2 ([12]) *Give the selfish mining strategy in Algorithm 2, the stationary distribution of the states is*

$$\pi_{0,0} = \frac{1-2\alpha}{2\alpha^3 - 4\alpha^2 + 1}, \quad \pi_{1,1} = \left(\alpha - \alpha^2\right)\pi_{0,0}, \quad \pi_{i,0} = \alpha^i \pi_{0,0}, \text{ for } i \geq 1$$

$$\pi_{i,j} = \alpha^i (1-\alpha)^j (1-\gamma)^j f(i,j,j)\pi_{0,0} + \alpha^{i-j}\gamma(1-\gamma)^{j-1}\left(\frac{1}{(1-\alpha)^{i-j-1}} - 1\right)\pi_{0,0} -$$

$$\gamma(1-\gamma)^{j-1}\sum_{k=1}^{j}\alpha^{i-k}(1-\alpha)^{j-k} f(i,j,j-k)\pi_{0,0}, \text{ for } i \geq j+2, j \geq 1.$$

The function $f(x, y, z)$ is multiple summations and is defined as

$$f(x,y,z) = \begin{cases} \underbrace{\sum_{s_z=y+2}^{x}\sum_{s_{z-1}=y+1}^{s_z} \cdots \sum_{s_1=y-z+3}^{s_2}}_{z} 1, & z \geq 1, x \geq y+2, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Next, we conduct the reward analysis for each state transition. Here, a probabilistic way is used to track various block rewards [26]. Specifically, in each state transition, there is a new block (mined by an honest miner or the selfish miner). It is impossible to decide the number of rewards associated with this new block when it is just created because the "destiny" of this new block depends on the evolution of the system. Hence, the expected rewards for the new block are computed based on possible

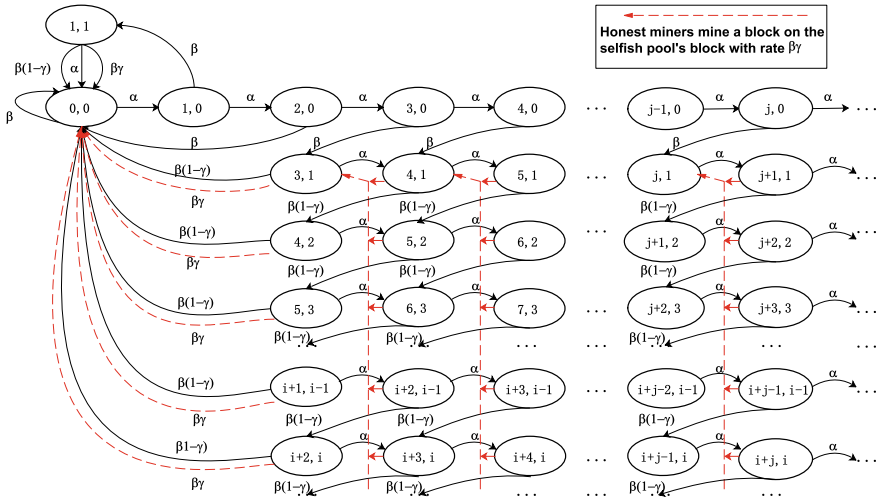

Fig. 9 The Markov process of the selfish mining in Ethereum

future events. In contrast, the selfish mining analysis in Bitcoin tracks published blocks associated with a state transition (whose destiny is already determined) rather than the new block and so it can compute the exact rewards [11]. This gives rise to the following two questions:

1. What is wrong with tracking published blocks?
2. How to compute the expected rewards for a new block at the time of its creation?

To answer the first question, one shall notice that tracking published blocks don't provide enough information to compute the uncle and nephew rewards. Recall from Sect. 3.1 that a published regular block can receive nephew rewards by referencing outstanding uncle blocks. The amount of nephew rewards depends on the number of outstanding uncle blocks. Therefore, it is necessary to keep track of all the outstanding uncle blocks in the system together with their depth information (which is needed to determine the number of uncle rewards). This greatly complicates the state space.

To answer the second question, one shall notice that it suffices to compute the expected rewards for a new block by using the following information: the probability that it becomes a regular block, the probability that it becomes an uncle block, the distance to its potential nephew block (if it indeed becomes an uncle block). Perhaps a bit surprisingly, all the information can be determined when this new block is generated for the selfish mining strategy in Algorithm 2.

To better understand this probabilistic method, we can see a simple example, as shown in Fig. 10. Assume that the selfish miner has already mined two blocks and kept them private at time t. Then, some honest miner generates a new block (case a in Fig. 10). According to Algorithm 2, the selfish miner publishes its private branch immediately. As such, this new block will become an uncle block with probability 1. Furthermore, we can see that this block will have a distance of 2 with its potential

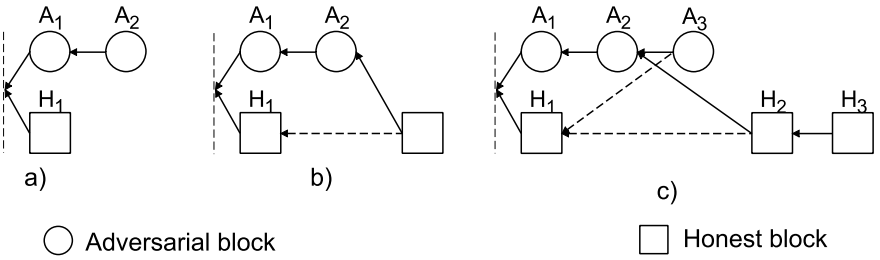

Fig. 10 A simple illustration of the probabilistic reward computing method

nephew block. Thus, this new block will receive an uncle reward of $K_u(2)$. Similarly, its potential nephew block will receive a nephew reward of $K_n(2)$. Moreover, this reward will belong to some honest miners with probability β (case b in Fig. 10) and $\alpha\beta^2(1-\gamma)$ (case c in Fig. 10). Clearly, the selfish miner obtains this reward with probability $1-\beta(1+\alpha\beta(1-\gamma))$. (See *Case 7* in Appendix B [12] for details.) Therefore, the expected rewards associated with this new block are $K_u(2)+K_n(2)$ in total among which $K_u(2)+K_n(2)\beta(1+\alpha\beta(1-\gamma))$ rewards will belong to honest miners (and the remaining will belong to the selfish miner). We refer interested readers to [12] for more details of the reward analysis.

With reward analysis, it is easy to compute the block rewards r_b^s (resp., r_b^s), uncle rewards r_u^s (resp., r_u^h), and nephew rewards r_n^s (resp., r_b^h) for the selfish miner (resp., honest miners) in the following theorem.

Theorem 2 ([12]) *Give the selfish mining strategy in Algorithm 2, the rewards for the selfish miner and honest miners are*

$$r_b^s = \frac{\alpha(1-\alpha)^2(4\alpha+\gamma(1-2\alpha))-\alpha^3}{2\alpha^3-4\alpha^2+1},$$

$$r_b^h = \frac{(1-2\alpha)(1-\alpha)(\alpha(1-\alpha)(2-\gamma)+1)}{2\alpha^3-4\alpha^2+1},$$

$$r_u^s = \frac{(1-2\alpha)(1-\alpha)^2\alpha(1-\gamma)}{2\alpha^3-4\alpha^2+1}K_u(1),$$

$$r_u^h = (\alpha\beta+\beta^2\gamma)K_u(1)\pi_{1,0}+\sum_{i=2}^{\infty}\beta K_u(i)\pi_{i,0}+\sum_{i=2}^{\infty}\sum_{j=1}^{\infty}\beta\gamma K_u(i)\pi_{i+j,j},$$

$$r_n^s = \alpha\beta K_s(1)\pi_{1,0}+\sum_{i=2}^{\infty}\sum_{j=1}^{\infty}\beta^{i-1}\gamma(\alpha-\alpha\beta^2(1-\gamma))K_s(i)\pi_{i+j,j},$$

$$r_n^h = \alpha\beta^2(1-\gamma)K_s(1)\pi_{0,0}+\beta^2\gamma K_s(1)\pi_{1,0}++\sum_{i=2}^{\infty}\sum_{j=1}^{\infty}\beta^i\gamma(1+\alpha\beta(1-\gamma))K_s(i)\pi_{i+j,j}.$$

With this theorem, it is easy to derive the absolute revenue $U_s(\alpha,\gamma)$ of the selfish miner in Eq. (4). Note that the absolute revenue is equivalent to the relative revenue (i.e., the share $R_s(\alpha,\gamma)$) in Bitcoin, but it is different from the relative revenue in

Ethereum due to the presence of uncle and nephew rewards. This is because Bitcoin adjusts the mining difficulty level so that the regular blocks are generated at a stable rate, say one block per time unit. Thus, the long-term average total revenue is fixed to be one block reward per time unit with or without selfish mining. This makes the absolution revenue equivalent to the relative revenue. The situation is different in Ethereum. Even if the regular blocks are generated at a stable rate, the average total revenue still depends on the generation rate of uncle blocks, which is affected by the selfish mining attack. Indeed, Ethereum didn't take into account the generation rate of uncle blocks when adjusting the difficulty level until the Byzantium hard fork. Therefore, there are two scenarios: (1) the regular block generation rate is 1 block per time unit and (2) the regular and uncle block generation rate is 1 block per time unit.

In the above analysis, the regular block generation rate is $r_b^s + r_b^h$, which is smaller than 1 as explained before. Thus, the time can be re-scaled to make the regular block generation rate be 1 block per time unit. In this scenario, the long-term absolute revenue for the selfish pool is

$$U_s(\alpha, \gamma) = \frac{r_b^s + r_u^s + r_n^s}{r_b^s + r_b^h}, \tag{4}$$

and the long-term absolute revenue for honest miners is

$$U_h(\alpha, \gamma) = \frac{r_b^h + r_u^h + r_n^h}{r_b^s + r_b^h}. \tag{5}$$

Similarly, the time can be scaled to make the regular and uncle block generation rate to be 1 block per time unit and define long-term absolute revenues for the selfish pool and honest miners accordingly. Finally, the threshold of the computation power to make selfish mining profitable in Ethereum can be derived. Specifically, if the selfish miner follows the mining protocol, its long-term average absolute revenue will be α, since the network delay is negligible (and so no stale blocks will occur). If the miner applies the selfish mining strategy in Algorithm 2, its long-term absolute revenue is given by $U_s(\alpha, \gamma)$, which can be larger than α. So the thresholds $\min_\alpha\{U_s(\alpha, \gamma) > \alpha\}$ $\min_\alpha$ and $\{R_s(\alpha, \gamma) > \alpha\}$ for both scenarios through numerical calculations can be derived, respectively.

To better the results, we plot the selfish miner's expected absolute revenue with α ranging from 0 to 0.5 in Fig. 11a. Here, due to the uniform tie-breaking policy, $\gamma = 1/2$. From it, we can see when the selfish miner controls more than 0.163 of the computation power, it can gain more from the selfish mining attack than that from honest mining. The threshold is lower than that (i.e., 0.25 when $\gamma = 1/2$) in Bitcoin. In other words, the additional uncle and nephew reward lowers the system security level. Figure 11b plots the threshold of the attack with different γ. Particularly, we compute the thresholds for the two scenarios: (1) the regular block generation rate is 1 block per time unit and (2) the regular and uncle block generation rate is 1 block per

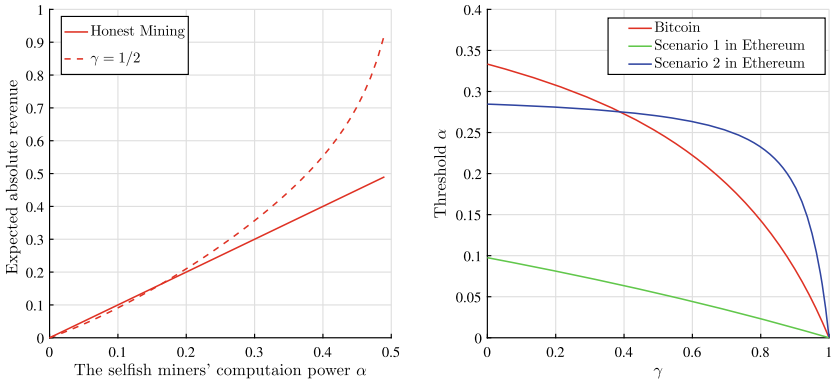

Fig. 11 The results of the selfish mining attack in Ethereum. **a** The selfish miner's relative revenue with different α. **b** The threshold of the selfish mining attack with different γ

time unit. From Fig. 11b, we can see that the higher γ is, the lower hash power needed for profitable selfish mining. Especially, when $\gamma = 1$, the selfish mining in Bitcoin and Ethereum can always be profitable regardless of their hash power. Besides that, the results show that the hash power thresholds of Ethereum in scenario 1 are always lower than in Bitcoin. By contrast, the hash power thresholds in scenario 2 are higher than Bitcoin when $\gamma \geq 0.39$. This is because the larger γ is, the more blocks mined by honest miners are uncle blocks. However, in scenario 2, the additional referenced uncle blocks will reduce the generation rate of the regular block, resulting in the decrease of selfish pools' block rewards. Thus, the selfish pool needs to have higher hash power to make selfish mining profitable. This suggests that Ethereum should consider the uncle blocks into the difficulty adjustment under the mining strategy given in Algorithm 2.

4 Incentive Design and Analysis in Bitcoin-NG

Bitcoin has suffered from low throughput (i.e., 7 transactions per second) and long latency (i.e., about one hour to confirm transactions) since inception. The poor performance significantly hinders blockchains' applications, and so a lot of scalable blockchain protocols are proposed [1, 10, 22, 29, 35, 42]. Among them, Bitcoin-NG (next generation) is among the first and the most prominent NC-based blockchains to approach the *near-optimal* throughput [10]. Bitcoin-NG creatively employs two types of blocks: (1) a *key block* that is very similar to a conventional block in Bitcoin except that it doesn't carry any transactions, and (2) a *microblock* that carries transactions. Every key block is generated through the leader election process (often known as the mining process) in NC. Each leader can issue multiple microblocks and receive the transaction fees until the next key block is generated. Unlike Bitcoin,

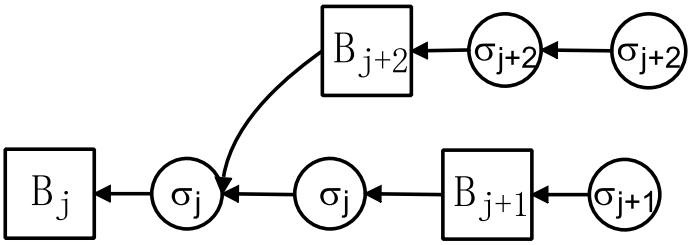

Fig. 12 An illustration of Bitcoin-NG. A square (respectively, circle) block denotes a key block (respectively, microblock). The microblocks are issued by the three key-block miners $\mathcal{B}_j, \mathcal{B}_{j+1}, \mathcal{B}_{j+2}$ with their signatures $\sigma_j, \sigma_{j+1}, \sigma_{j+2}$, respectively

Bitcoin-NG decouples leader election and transaction serialization. Intuitively, it is this decoupling that enables Bitcoin-NG to approach the near-optimal throughput, since the microblocks can be produced at a rate up to the network capacity. In this section, we focus on the incentive design of Bitcoin-NG, especially studying the impact of this decoupling idea on incentive-based attacks.

4.1 Overview of Bitcoin-NG

In Bitcoin, the mining of blocks has two functionalities: (1) electing leaders (i.e., the owners of valid blocks) by NC and (2) ordering and verifying transactions. By differentiating block functionalities, Bitcoin-NG decouples the leader election with the transaction serialization. Specifically, Bitcoin-NG uses key blocks mined through PoW to elect a leader at a stable rate (e.g., one key block per 100 s). Each leader can produce several microblocks containing unconfirmed transactions at another rate, often higher than the key-block rate (e.g., one microblock per 20 s). In a nutshell, a key block is very similar to a conventional block in Bitcoin except that it does not carry any transactions. On the other hand, microblocks contain transactions but do not contain any proof of work. Although the rate of microblocks is usually much larger than the key-block generation rate, it has to be bounded to prevent adversarial leaders from swamping the system with microblocks. This decoupling enables Bitcoin-NG to process many microblocks between two consecutive key blocks, which significantly increases its transaction throughput. Figure 12 illustrates these two types of blocks.

Bitcoin-NG adopts a similar fork choice rule as Bitcoin. In Bitcoin-NG, microblocks carry no weight, not even a secondary index for miners to choose which key block to mine. For instance, in Fig. 12, there are two forking branches with the same number of key blocks but different numbers of microblocks. Miners treat these two forking branches as equal, adopt a uniform tie-breaking rule to choose one branch, and then mine after the latest microblock in this branch [10]. To sum up, an honest miner still follows LCR to choose a "right" key block (i.e., the last key block in the longest chain consisting of key blocks only), and then mines on the

latest microblock produced by the key-block miner. Thus, without microblocks, the mining process of key blocks is the same as the one in Bitcoin. The selfish mining attack in Bitcoin can also be used here to attack key blocks in Bitcoin-NG. Similar to Bitcoin, Bitcoin-NG also provides two rewards, namely, key-block reward and transaction fee. Every miner obtains a key-block reward if it mines a key block by successfully solving a PoW puzzle and its key block ends up in the longest chain. After mining a key block, miners can also obtain transaction fees by including as many transactions as possible (up to the microblock size limit) in their microblocks.

4.2 Microblocks and Its Incentive-Based Attacks

In Bitcoin-NG, miners should include as many transactions as possible in their microblocks and publish these microblocks to win transaction fees. This is called the transaction inclusion rule. In addition, miners should accept as many microblocks issued by the previous key-block miner as possible and mine on the latest received microblock, i.e., obeying the longest chain extension rule. It is easy to see when all miners obey these two rules, more produced microblocks could be included in the blockchain, and Bitcoin-NG could achieve better transaction throughput. By contrast, a selfish miner could break the transaction inclusion and the longest chain extension rules to maximize its profit from transaction fees as explained below:

- **Transaction inclusion attack.** When the selfish miner publishes one key block and generates multiple microblocks, it keeps the last several microblocks private. That is, the selfish miner continues to mine on top of its latest microblock chain, while honest miners can only mine on top of the last published microblock. Figure 13 shows the case in which the selfish miner withholds some of its microblocks mined after the key block $\mathcal{B}_j$, and honest miners mine on the last public microblock of the selfish miner. This attack is incentivized if transaction fees in microblocks go primarily to the next key-block owner.
- **Longest chain extension attack.** When the selfish miner adopts an honest key block, it can reject some (or all) microblocks and mine directly on the last accepted microblock (or the last key block, respectively). In other words, the selfish miner rejects the transactions in these microblocks issued by the previous honest key-block miner. This attack is illustrated in Fig. 14. This attack is incentivized if transaction fees go primarily to the current key-block owner.

From the above cases, it is easy to see that the transaction fees in microblocks can neither go to the next key-block owner nor go primarily to the current key-block owner. Therefore, to resist these two attacks, Bitcoin-NG divides the transaction fees included in microblocks between two consecutive key-block miners into two parts. The first key-block miner gets the r fraction ($r \in [0, 1]$), while the second one

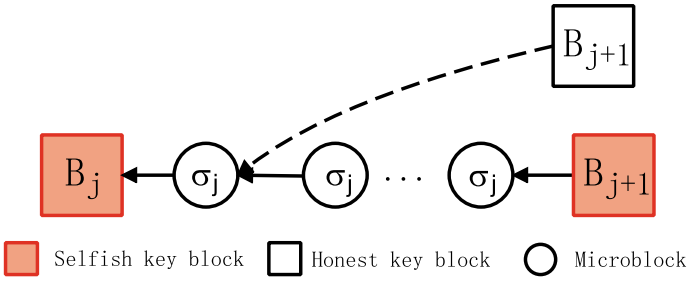

Fig. 13 An example of the transaction inclusion attack. The first two microblocks after the selfish $\mathcal{B}_j$ have been published and so they are public to honest miners. The other microblocks are kept private. A dashed square block denotes a future mined block

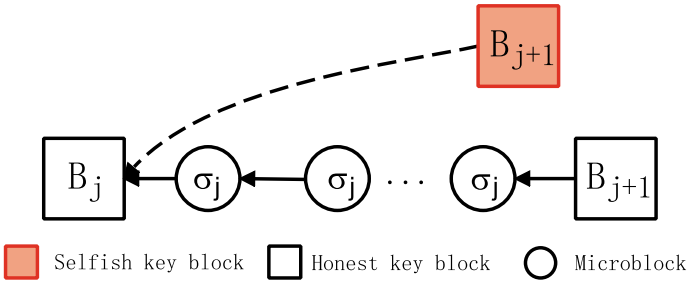

Fig. 14 An example of the longest chain extension attack. The selfish miner rejects all the microblocks and mines its key block on top of the honest $\mathcal{B}_j$. A dashed square block denotes a future mined block

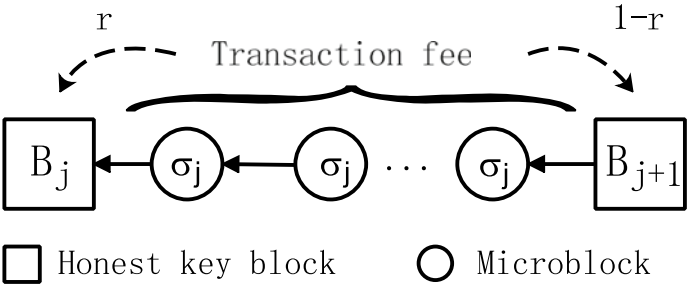

Fig. 15 Bitcoin-NG fee distribution rule

obtains the remaining $1 - r$ fraction. Figure 15 illustrates this fee distribution rule. The following analysis shows how to decide the value of r to resist the microblock mining attacks.

Resisting Transaction Inclusion Attack. In this attack, the selfish miner can withhold a microblock to avoid sharing its transaction fees with the subsequent key-block miner. (We refer readers to the original paper [10] for more details.) Note that the probability for the selfish (respectively, honest) miner mines a block is α (respec-

tively, $\beta = 1 - \alpha$). To guarantee the average revenue of the selfish miner launching the above attack is smaller than what it deserves, the distribution ratio r should satisfy

$$\overbrace{\alpha \times 100\%}^{\text{win 100\%}} + \overbrace{(1-\alpha) \times \alpha \times (100\% - r)}^{\text{Lose 100\%, but mine after txn}} < r, \tag{6}$$

therefore $r > 1 - \frac{1-\alpha}{1+\alpha-\alpha^2}$. This ratio requirement encourages the selfish miner to place a transaction in a public microblock.

Later, Yin et al. [41] found that the above computation neglects a case: the incumbent leader can be re-elected as the next leader and gain an extra $\alpha(1-r)$ fraction of the transaction fee. Thus, the distribution ratio r should satisfy

$$\overbrace{\alpha \times 100\%}^{\text{win 100\%}} + \overbrace{(1-\alpha) \times \alpha \times (100\% - r)}^{\text{Lose 100\%, but mine after txn}} < r + \alpha(1-r), \tag{7}$$

therefore $r > \frac{\alpha}{1-\alpha}$.

Resisting Longest Chain Extension Attack To increase revenue from some transactions, the selfish miner can ignore these transactions in an honest microblock and mine on a previous microblock. Later on, if the selfish miner mines a key block, it can place these transactions in its microblock. To resist this attack, the selfish miner's revenue in this case must be smaller than the revenue obtained by obeying the longest chain extension rule. Therefore, we have

$$\overbrace{\alpha \times r}^{\text{Mine next key block}} + \overbrace{\alpha^2 \times (100\% - r)}^{\text{Mine the third key Block}} < \overbrace{\alpha(100\% - r)}^{\text{Mine on microblock}}, \tag{8}$$

which leads to $r < \frac{1-\alpha}{2-\alpha}$. Taking the upper bound into consideration, the distribution ratio r satisfies $1 - \frac{1-\alpha}{1+\alpha-\alpha^2} < r < \frac{1-\alpha}{2-\alpha}$. In particular, when α is less than 25%, we obtain $37\% < r < 43\%$. Hence, $r = 40\%$ is chosen in the Bitcoin-NG [10].

Discussion. The above analysis provides theoretical results on the value of r, but it has several limitations. To better understand it, let us replay the longest chain extension attack, as shown in Fig. 16. Here, two simplifications are made to better illustrate the analysis limitation: 1) each leader is allowed to only create one microblock; 2) each microblock is allowed to only contain one transaction. Consider a scenario where an honest miner produces a key block B_j as well as a microblock containing a transaction tx. If the selfish miner obeys the longest chain extension rule and finds the next key block with probability α, it will get a $1 - r$ fraction of the transaction fee (which corresponds to the last item in Eq. (8)). However, the selfish miner can directly mine on the key block B_j, hoping to win a higher transaction fee of tx. If the selfish miner happens to create the next key block B_{j+1} with probability α, it can win r fraction of the transaction fee by including tx in its microblock (which corresponds to the first item in Eq. (8)). If the selfish miner is lucky to mine the next consecutive key block B_{j+2}, it will win the remaining $1 - r$ of the transaction fee. Combining all conditions leads to Eq. (8).

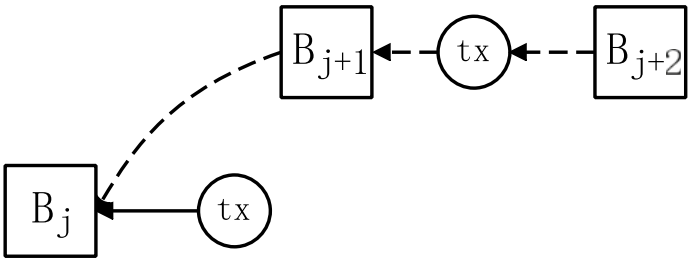

Fig. 16 A simple example to illustrate the limitation of the previous analysis. The selfish miner mines directly on block B_j and tries to include transaction tx in its future microblock

The analysis of the above example has two limitations. First, the simple analysis is quite reasonable if the selfish miner just hopes to get a higher fee from a targeted transaction tx. However, in reality, the selfish miner usually aims to increase its revenue from all transactions instead of just a targeted one. If the selfish miner applies the strategy to all the transactions rather than a single targeted one, it will quickly use up the space of its future microblocks. As a result, the selfish miner cannot include another transaction in its microblock, thereby losing the associated transaction fee. In other words, the above analysis ignores the impact of transaction size and microblock capacity, which magnifies the selfish miner's potential revenue from the attack. On the other hand, the above analysis works well for whale transactions with high fees, which are rare so that we don't need to worry about the space. But, most of the transactions in current blockchain systems have low fees.

Second, the selfish miner is assumed to always adopts honest miners' key blocks and immediately publishes its new key blocks (i.e., honest mining of key blocks). In other words, it does not consider the impact of key-block selfish mining. This assumption can only be justified when the selfish miner's computation power is less than the threshold of making key-block selfish mining profitable because the optimal mining strategy for key blocks is indeed honest mining [10, 11, 33]. However, once the selfish miner's computation power α is above the threshold, the selfish miner has the incentive to launch the key-block mining attack and so the impact of the key-block selfish mining cannot be ignored anymore. In the following, we will go through incentive analysis in [27], which has addressed the above two limitations.

4.3 Theoretical Results on Microblocks Mining

We use the mining models in Sect. 2.3. Let α denote the fraction of computation power controlled by selfish miners and β denote the fraction of total computation power controlled by honest miners. Without loss of generality, all selfish miners are assumed to be controlled by a single selfish miner. The key-block mining process is modeled as a Poisson process with rate f. Accordingly, the selfish miner generates key blocks at rate $f\alpha$, and the honest miners generate key blocks at rate $f\beta$. The

miner of each key block becomes a leader and can issue a series of microblocks containing as many transactions as possible (up to the maximum microblock size) at a constant rate v until the next key block is mined. Specifically, a block (including key block and microblock) mined by an honest (respectively, the selfish) miner is referred to as *honest* (respectively, *selfish*) block.

Following the network model of Bitcoin [11, 12], we assume that honest miners are fully connected through the underlying network, and an honest miner spends negligible time to broadcast a key block or microblock in Bitcoin-NG.[5] In addition, we assume that the selfish miner can broadcast its private blocks immediately after it sees a new honest key block.

We assume two types of transactions according to their transaction fees: "whale" transactions with a high fee and regular transactions with a low fee. Also, we assume that the vast majority of transactions are regular ones. We assume that the transaction size is fixed, and so the maximum number of transactions included in a microblock is also fixed. We also assume that miners have enough pending transactions to be included in microblocks.[6] We call a microblock regular if it contains only regular transactions. In addition, we refer to the total transaction fees included in a regular microblock as the microblock fee and use R_t to denote it. In addition, we use R_b to denote the key-block reward. Let $k = R_b/R_t$ denote the ratio of the block reward to the microblock fee. This ratio k ranges from $(0, \infty)$. When k approaches 0 (respectively, ∞), it implies that the transaction fee (respectively, key-block reward) dominates the reward. The different values of k exhibit the various impact of rewards on the Bitcoin-NG system. As whale transactions are so rare that they use little microblock space. For this reason, we can ignore their space requirement (even under the network capacity constraints) and apply the analysis in Sect. 4.2.

Next, we present the incentive analysis of microblocks. In particular, the analysis does not consider the selfish mining of key blocks. That is, it assumes that the selfish miner always adopts honest miners' key blocks and immediately publishes its new key blocks. This assumption is justified shortly and will be relaxed by considering the joint mining of microblocks and key blocks in Sect. 4.4. In addition, as the propagation delay of key blocks is negligible, forked key blocks are also not considered in the following analysis.

We consider the revenue of transaction fees in terms of regular transactions for the selfish miner and honest miners during a time interval $[0, t]$. Without loss of generality, we assume that there exists a block B_0 that the selfish miner and honest miners both agree to mine on at the starting time. (For example, B_0 can be the genesis block.) Let $M(t)$ be the number of key blocks mined during the time interval $[0, t]$. Let X_i ($i \in [0, M(t)]$) denote an indicator random variable which equals one if the i-th key block is a selfish key block, as described below:

[5] This assumption is reasonable for key blocks because the inter-arrival time of two consecutive key blocks is often much larger than the block propagation delay. On the other hand, this assumption can be relaxed for microblocks, as we will show later.

[6] This assumption is reasonable in Bitcoin and Ethereum-like public blockchains. For instance, a mempool visualization website [21] shows that the number of pending transactions is around 136k in May 2021.

$$X_i = \begin{cases} 1, \text{selfish key block} \\ 0, \text{honest key block.} \end{cases}$$

Without loss of generality, we assume block B_0 is an honest key block. For other key blocks, the possibility that it is a selfish key block is equal to α.

After mining a key block, its owner can issue a series of microblocks at a constant rate v until the next key block is mined. Here, the rate v captures the network capacity constraints. Let Y_i denote the interval between the i-th key block and $(i+1)$-th key block. Thus, the number of produced microblocks between i-th and $(i+1)$-th key blocks is vY_i. In addition, each microblock contains a total fee of R_t because we only consider regular transactions here. We are now ready to compute the suitable value of r to resist the two microblock attacks for regular transactions.

Resisting Transaction Inclusion Attack. In this attack, the selfish miner hides some of its microblocks generated after a key block but keeps mining on top of the microblock chain. Hence, honest miners directly mine on top of the selfish miner's last published block. Let ρ denote the fraction of the unpublished microblocks among all the selfish microblocks between two consecutive key blocks. In particular, $\rho = 1$ means that the selfish miner hides all the microblocks it has generated between two consecutive key blocks. Thus, if any two consecutive key blocks satisfy $(X_i, X_{i+1}) = (1, 0)$, there are $(1-\rho)vY_i$ microblocks between them from the view of an honest miner; otherwise, there are vY_i microblocks.

Let Z_i denote an indicator random variable equal to one if $\{X_i = 1, X_{i+1} = 0\}$, and equal to zero otherwise. Next, let $Z = \sum_{i=1}^{M(t)-1} Z_i$. Suppose $M(t) = m$. The following lemma will aid us to bound the value of Z with high probability.

Lemma 3 *For m consecutive key blocks, the number of block pairs $(X_i, X_{i+1}) = (1, 0)$ has the following Chernoff-type bound: For $0 < \delta < 1$,*

$$\Pr(|Z - \alpha\beta(m-1)| > \delta\alpha\beta(m-1)) < e^{-\Omega\left(\delta^2\alpha\beta m\right)}. \tag{9}$$

This lemma shows that as m increases, the number of key pairs $(X_i, X_{i+1}) = (1, 0)$ is between $(1-\delta)\alpha\beta m$ and $(1+\delta)\alpha\beta m$ with high probability.

Next, we compute the selfish miner's relative revenue for large m. On the one hand, the total amount of transaction fees for all the miners is given by $\sum_{i=1}^{m-1}(vY_iR_t - \rho vZ_iY_iR_t)$. To see this, note that there are $\sum_{i=1}^{m-1} vY_i$ microblocks produced with associated transaction fees $\sum_{i=1}^{m-1} vY_iR_t$. Note also that once $Z_i = 1$, there are ρvY_i microblocks not being included in the longest chain due to the transaction inclusion attack. Hence, the associated loss of transaction fees is $\sum_{i=1}^{m-1} \rho vZ_iY_iR_t$. On the other hand, the total transaction fees for the selfish miner is given by $\sum_{i=1}^{m-1}(\alpha vY_iR_t - r\rho vZ_iY_iR_t)$. To see this, note that without any attack, the selfish miner can get α fraction of the total transaction fees given by $\sum_{i=1}^{m-1} \alpha vY_iR_t$. Note also that with the transaction inclusion attack, the selfish miner will lose r

fraction of the total loss of transaction fees as the first leader. Combining the above analysis, we have the following lemma for large m.

Lemma 4 *The selfish miner's relative revenue u converges to $\frac{\alpha - r\alpha\beta\rho}{1-\alpha\beta\rho}$ with high probability as $m \to \infty$.*

This lemma says that for large m, the selfish miner's relative revenue is $\frac{(\alpha - r\alpha\beta\rho)}{(1-\alpha\beta\rho)}$. Recall that the key-block generation process is a Poisson process with rate f, and so $M(t)$ is a Poisson arrival process. Hence, when t tends to infinity, $M(t)/t \to f$ holds with high probability. Therefore, with high probability, the maximum relative revenue of the selfish miner during $[0, t]$ is

$$\begin{aligned} u &= \max_{0\le\rho\le 1} \frac{\alpha - r\alpha\beta\rho}{1-\alpha\beta\rho} \\ &= r + \max_{0\le\rho\le 1} \frac{\alpha - r}{1-\alpha\beta\rho}. \end{aligned} \tag{10}$$

If $r \le \alpha$, the optimal $\rho = 1$ and the corresponding

$$u = r + \frac{\alpha - r}{1-\alpha\beta}.$$

In this case, u is always larger than α since $1 - \alpha\beta < 1$. This means that the selfish miner can always have a relative revenue greater than its fair share by utilizing this attack. On the other hand, if $r > \alpha$, the optimal $\rho = 0$ and $u = \alpha$. This means that the maximum relative revenue that the selfish miner can obtain is honest mining (i.e., $\rho = 0$). Therefore, we should set $r > \alpha$ to guarantee the adversary cannot gain more from the transaction inclusion attack.

Resisting Longest Chain Extension Attack. In this attack, the selfish miner can bypass some honest microblocks and mines directly on an old honest block. Similarly, let ρ denote the rejected microblock fraction. In particular, $\rho = 1$ means that the selfish miner rejects all honest microblocks and mines directly on the last honest key block. More precisely, if two consecutive key blocks are $(X_i, X_{i+1}) = (0, 1)$, there are $(1 - \rho)vY_i$ honest microblocks accepted by the longest chain. Let K_i denote an indicator random variable equal to one if $\{X_i = 0, X_{i+1} = 1\}$, and equal to zero otherwise. Let $K = \sum_{i=1}^{m-1} K_i$. The following lemma will aid us to bound the expectation of K for m blocks.

Lemma 5 *For the m block sequence, the number of block pair $(X_i, X_{i+1}) = (0, 1)$ has the following Chernoff-type bound: For $0 < \delta < 1$,*

$$\Pr(|K - \alpha\beta(m-1)| > \delta\alpha\beta(m-1)) < e^{-\Omega(\delta^2\alpha\beta m)}. \tag{11}$$

Next, we compute the selfish miner's relative revenue for large m. On the one hand, the total amount of transaction fees for all the miners is given by $\sum_{i=1}^{m-1}(vY_iR_t - \rho vK_iY_iR_t)$. To see this, recall that there are $\sum_{i=1}^{m-1} vY_i$ microblocks

produced with associated transaction fees $\sum_{i=1}^{m-1} vY_iR_t$. Once $K_i = 1$, there are ρvY_i microblocks not being included in the longest chain due to the longest chain extension attack. Hence, the associated loss of transaction fees is $\sum_{i=1}^{m-1} \rho vK_iY_iR_t$. On the other hand, the total transaction fees for the selfish miner is given by $\sum_{i=1}^{m-1} (\alpha vY_iR_t - r\rho vZ_iY_iR_t)$. To see this, recall that without any attack, the selfish miner can get α fraction of the total transaction fees given by $\sum_{i=1}^{m-1} \alpha vY_iR_t$. With the longest chain extension attack, the selfish miner will lose $1-r$ fraction of the total loss of transaction fees as the second leader. Combining the above analysis, we have the following lemma for larger m.

Lemma 6 *The selfish miner's relative revenue μ converges to $\frac{\alpha-(1-r)\alpha\beta\rho}{1-\alpha\beta\rho}$ with high probability as $m \to \infty$.*

This lemma says that for large m, the selfish miner's relative revenue is $\frac{\alpha-(1-r)\alpha\beta\rho}{1-\alpha\beta\rho}$. Similar with the previous analysis, we can show that as $t \to \infty$, with high probability, the maximum relative revenue of the selfish miner during $[0, t]$ is

$$\begin{aligned} u &= \max_{0\le\rho\le 1} \frac{\alpha-(1-r)\alpha\beta\rho}{1-\alpha\beta\rho} \\ &= 1-r+\max_{0\le\rho\le 1} \frac{r-\beta}{1-\alpha\beta\rho}. \end{aligned} \tag{12}$$

If $r \ge \beta$, the optimal $\rho = 1$ and the corresponding

$$u = 1-r+\frac{r-\beta}{1-\alpha\beta}.$$

In this case, u is always larger than α since $1-\alpha\beta < 1$. This means that the selfish miner can always have a relative revenue greater than its fair share by launching this attack. On the other hand, if $r < \beta$, the optimal $\rho = 0$ and $u = \alpha$. This means that the maximum relative revenue that the selfish miner can obtain is honest mining (i.e., $\rho = 0$). Therefore, we should set $r < \beta$ to guarantee the adversary cannot gain more from the longest chain extension attack. Combining the two incentive sub-mechanisms of transaction inclusion and longest chain extension, the value of r needs to satisfy that

$$\alpha < r < \beta.$$

Discussion. The analysis in [10, 41] can be used to bound the split ratio r for whale transactions, while the above analysis can provide a new bound for the ratio r for regular transactions under network capacity constraints. These bounds are depicted in Fig. 17. The figure shows that the new bound $\alpha < r < \beta$ (for regular transactions) contains the previous two bounds $1-\frac{1-\alpha}{1+\alpha-\alpha^2} < r < \frac{1-\alpha}{2-\alpha}$ and $\frac{\alpha}{1-\alpha} < r < \frac{1-\alpha}{2-\alpha}$ (for whale transactions). This leads to several interesting implications.

First, introducing network capacity constraints doesn't make it harder to maintain the incentive compatibility of Bitcoin-NG. This is because the bounds for whale transactions are the same as the previous ones and the bound for regular transactions

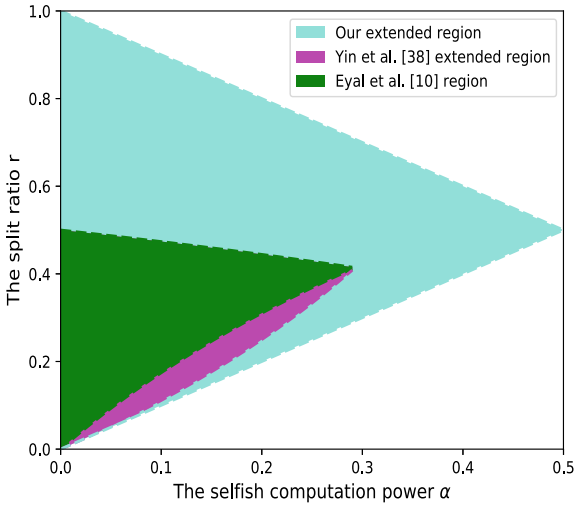

Fig. 17 The comparison of the transaction fee distribution ratio

contains the previous ones. Second, when α is smaller than 29%, we can find a value of r that satisfies all the bounds. This means that the incentive compatibility of Bitcoin-NG can be maintained for all types of transactions even under network capacity constraints in this regime. Third, when α is larger than 29%, we cannot find a value of r that satisfies all the bounds, because the two bounds for whale transactions both become invalid. This means that the incentive compatibility of Bitcoin-NG can be maintained only for regular transactions but not for whale transactions in this regime. In other words, the presence of whale transactions might cause instability of the whole system in this regime. As such, some defense mechanisms should be designed accordingly.

4.4 Theoretical Results on Microblocks and Key-Block Mining in Bitcoin-NG

In this section, we go through the analysis that jointly considers microblock and key-block mining. Particularly, the Markov decision process (MDP) is applied to model various selfish mining strategies. To make our analysis tractable, two simplifications are made. First, the key-block interval is assumed to be $1/f$ and so the number of microblocks produced between two consecutive key blocks is v/f. (Note that the key-block interval is assumed to follow the exponential distribution with mean $1/f$.) Second, only binary choices: publishing or hiding all selfish microblocks in the transaction inclusion attack and accepting or rejecting all honest microblocks in the longest chain extension attack are considered. (This is consistent with the fact that $\rho = 0$ or 1 in Sect. 4.3.)

The MDP model can be presented by a 4-tuple $\mathcal{M} := (S, A, P, R)$, where S is the state space, A is the action space, P is the stochastic state transition matrix, and R is the reward matrix. Specifically, S contains all possible states in the selfish mining process; A includes the available actions (e.g., publishing or hiding blocks by the selfish miner) at each state; P contains the transition probabilities from the current state to the next state according to the taken action; and R records how much the selfish miner obtains when there are some state transitions. Table 2 illustrates the MDP of selfish mining in Bitcoin-NG. Note that blocks are assumed to be transmitted without delay (see Sect. 2.3), and so forks are not considered in the analysis. Below we will discuss each component of the 4-tuple:

Actions. The selfish miner has eight available actions.

- **Adopt and include.** The selfish miner accepts all honest key blocks and the corresponding honest microblocks. In other words, the selfish miner will mine its key block on the last honest block and abandon its private chain. This action is referred to as adopt.
- **Adopt and exclude.** The selfish miner accepts all honest key blocks and microblocks except for microblocks produced after the last honest key block. Specifically, the selfish miner directly mines on top of the last honest key block, which is referred to as adoptE.
- **Override and publish.** The selfish miner publishes all its key blocks and corresponding microblocks whenever its private chain is longer than the honest one. The chain length is counted by the key block. This action is denoted as override.
- **Override and hide.** The selfish miner publishes all its key blocks and the microblocks except for these mined after the last selfish key block whenever its private chain is longer than the honest one. This action is denoted as overrideH.
- **Match and publish.** When an honest miner finds one new key block, the selfish miner publishes its key block of the same height and the microblocks built after this key block. This action is available when the selfish miner has one block in advance and is referred to as match.

Table 2 State transition and reward matrices for the optimal selfish mining (Prob.: probability; Cond.: condition)

State × Action	State	Probability	Reward	Condition
$(l_a, l_h, \cdot, S_h)$, adopt	$(1, 0, \text{noTie}, H_{\text{in}})$	α	$(l_h, l_h, 0, 0)$	–
$(l_a, l_h, \cdot, S_p)$, adopt			$(l_h, l_h - 1 + (1 - r), 0, r)$	
$(l_a, l_h, \cdot, \{H_{\text{in}}, H_{\text{ex}}\})$, adopt	$(0, 1, \text{noTie}, H_{\text{in}})$	$1 - \alpha$	$(l_h, l_h - 1, 0, 0)$	

Table 2 (continued)

State × Action	State	Probability	Reward	Condition
$(l_a, l_h, \cdot, S_h)$, adoptE	$(1, 0, \text{noTie}, H_{\text{ex}})$	α	$(l_h, l_h, 0, 0)$	–
$(l_a, l_h, \cdot, S_p)$, adoptE	$(0, 1, \text{noTie}, H_{\text{ex}})$	$1-\alpha$	$(l_h, l_h - 1 + (1-r), 0, r)$	
$(l_a, l_h, \cdot, \{H_{\text{in}}, H_{\text{ex}}\})$, adoptE			$(l_h, l_h - 1, 0, 0)$	
$(l_a, l_h, \cdot, H_{\text{ex}})$, override	$(l_a - l_h, 0, \text{noTie}, S_p)$	α	$(0, 0, l_h + 1, l_h + 1)$	$l_a > l_h$
$(l_a, l_h, \cdot, H_{\text{in}})$, override	$(l_a - l_h - 1, 1, \text{noTie}, S_p)$	$1-\alpha$	$(0, r, l_h + 1, l_h + (1-r))$	
$(l_a, l_h, \cdot, \{S_p, S_h\})$, override			$(0, 0, l_h + 1, l_h)$	
$(l_a, l_h, \cdot, H_{\text{ex}})$, overrideH	$(l_a - l_h, 0, \text{noTie}, S_h)$	α	$(0, 0, l_h + 1, l_h + 1)$	$l_a > l_h$
$(l_a, l_h, \cdot, H_{\text{in}})$, overrideH	$(l_a - l_h - 1, 1, \text{noTie}, S_h)$	$1-\alpha$	$(0, r, l_h + 1, l_h + (1-r))$	
$(l_a, l_h, \cdot, \{S_p, S_h\})$, overrideH			$(0, 0, l_h + 1, l_h)$	
$(l_a, l_h, \text{noTie}, \cdot)$, wait	$(l_a + 1, l_h, \text{noTie}, *)$	α	$(0, 0, 0, 0)$	–
	$(l_a, l_h + 1, \text{noTie}, *)$	$1-\alpha$		
$(l_a, l_h, noTie, H_{\text{in}})$, match $(l_a, l_h, \text{tie}, H_{\text{in}})$, wait	$(l_a + 1, l_h, \text{tie}, H_{\text{in}})$	α	$(0, 0, 0, 0)$	$l_a \geq l_h$
	$(l_a - l_h, 1, \text{noTie}, S_p)$	$\gamma(1-\alpha)$	$(0, r, l_h, l_h - 1 + (1-r))$	
	$(l_a, l_h + 1, \text{noTie}, H_{\text{in}})$	$(1-\gamma)(1-\alpha)$	$(0, 0, 0, 0)$	
$(l_a, l_h, noTie, H_{\text{ex}})$, match $(l_a, l_h, \text{tie}, H_{\text{ex}})$, wait	$(l_a + 1, l_h, \text{tie}, H_{\text{ex}})$	α	$(0, 0, 0, 0)$	$la \geq l_h$
	$(l_a - l_h, 1, \text{noTie}, S_p)$	$\gamma(1-\alpha)$	$(0, 0, l_h, l_h - 1)$	

Table 2 (continued)

State × Action	State	Probability	Reward	Condition
	$(l_a, l_h + 1,$ noTie, $H_{\text{ex}})$	$(1-\gamma)(1-\alpha)$	$(0,0,0,0)$	
$(l_a, l_h,$ noTie, $\{S_p, S_h\})$, match $(l_a, l_h,$ tie, $\{S_p, S_h\})$, wait	$(l_a+1, l_h,$ tie, $*)$	α	$(0,0,0,0)$	$la \geq l_h$
	$(l_a - l_h, 1,$ noTie, $S_p)$	$\gamma(1-\alpha)$	$(0,0,l_h,l_h)$	
	$(l_a, l_h + 1,$ noTie, $*)$	$(1-\gamma)(1-\alpha)$	$(0,0,0,0)$	
$(l_a, l_h,$ noTie, $H_{\text{in}})$, matchH $(l_a, l_h, tie', H_{\text{in}})$, wait	$(l_a + 1, l_h,$ $tie', H_{\text{in}})$	α	$(0,0,0,0)$	$la \geq l_h$
	$(l_a - l_h, 1,$ noTie, $S_h)$	$\gamma(1-\alpha)$	$(0, r, l_h, l_h - 1 + (1-r))$	
	$(l_a, l_h + 1,$ noTie, $H_{\text{in}})$	$(1-\gamma)(1-\alpha)$	$(0,0,0,0)$	
$(l_a, l_h, noTie, H_{\text{ex}})$, matchH $(l_a, l_h,$ tie′, $H_{\text{ex}})$, wait	$(l_a + 1, l_h,$ tie′, $H_{\text{ex}})$	α	$(0,0,0,0)$	$l_a \geq l_h$
	$(l_a - l_h, 1,$ noTie, $S_h)$	$\gamma(1-\alpha)$	$(0,0,l_h,l_h-1)$	
	$(l_a, l_h +$ $1,$ noTie, $H_{\text{ex}})$	$(1-\gamma)(1-\alpha)$	$(0,0,0,0)$	
$(l_a, l_h, noTie,$ $\{S_p, S_h\})$, matchH $(l_a, l_h,$ tie′, $\{S_p, S_h\})$, wait	$(l_a + 1, l_h,$ tie′, $*)$	α	$(0,0,0,0)$	$l_a \geq l_h$
	$(l_a - l_h, 1,$ noTie, $S_h)$	$\gamma(1-\alpha)$	$(0,0,l_h,l_h)$	
	$(l_a, l_h +$ $1,$ noTie, $*)$	$(1-\gamma)(1-\alpha)$	$(0,0,0,0)$	
$(l_a, l_h,$ tie′, $\cdot)$, revert	$(l_a, l_h,$ tie, $*)$	1	$(0,0,0,0)$	–
$(l_a, l_h, \cdot, S_h)$, revert	$(l_a, l_h, *, S_p)$	1	$(0,0,0,0)$	$l_h = 0$
$(l_a, l_h, \cdot, H_{\text{ex}})$, revert	$(l_a, l_h, *, H_{\text{in}})$	1	$(0,0,0,0)$	$l_a = 0$

$*$ denotes the state element remains the same in the state transition.

- **Match and hide.** When an honest miner generates a new key block, the selfish miner publishes its key block of the same height while hiding the microblocks built after this key block. This action is also available when the selfish miner has one block in advance. This action is denoted as matchH.
- **Wait.** In this action, the selfish miner does not publish any new key blocks and microblocks, while keeps mining on its private chain until a new key block and corresponding microblocks are found.
- **Revert.** The selfish miner reverts its previous actions. Specifically, the selfish miner can publish its hidden microblocks when there is no honest key block mined after its block; the selfish miner can include the honest microblocks (decided to exclude in the previous decision) or exclude the honest microblocks (decided to include in the previous decision) once there is no selfish key block mined on an honest block.

The adopt, override match, and wait actions include all possible actions on the selfish mining of key blocks, while hide, publish, and revert actions cover all possible actions on the transaction inclusion and longest chain extension attacks of microblocks. Note that in the match action, the selfish miner publishes its key block of the same height to match a key block produced by honest miners. Therefore, there are two forking branches of the same length. In Bitcoin-NG, honest miners adopt a uniform tie-breaking rule to choose which branch to mine on. In particular, the variable γ is introduced to denote the fraction of honest miners that mine on the selfish miner's branch.

State space. The state space S is also composed of 4-tuple $(l_a, l_h, \mathsf{fork}, \mathsf{lastMicroBlock})$.

- $\boldsymbol{l_a}$ accounts for the length of the chain mined by the selfish miner after the last common ancestor key block. More precisely, the last common ancestor key block is the last key block in the longest chain accepted by both the selfish miner and *all* honest miners and is updated once the selfish miner adopts the public chain or all honest miners adopt the selfish miner's chain. In addition, the chain length is counted by the selfish key blocks in this branch.
- $\boldsymbol{l_h}$ is the length of the public chain after the last common ancestor key block. This chain can be viewed by both the selfish miner and honest miners.
- **fork**. The field fork obtains three possible values, dubbed noTie, tie, and tie$'$. Specifically, tie means the selfish miner publishes l_h selfish key block and the corresponding microblocks; tie$'$ presents the selfish miner publishes l_h selfish key block and the corresponding microblocks except for these after the last selfish key block; noTie signifies that there are not two public branches with the equivalent length.
- **lastMicroBlock**. This field also includes four possible values, dubbed H_{in}, H_{ex}, S_p, and S_h. Specifically, H_{in} (respectively, H_{ex}) represents the common ancestor is an honest key block, and the corresponding microblocks are accepted (respectively, rejected) by the selfish miner. While S_p (or S_h) which stands for the common ancestor is a selfish key block, and the corresponding microblocks mined are published (or hidden, respectively) by the selfish miner.

State Transition and Reward. The rewards for the selfish miner and honest miners in the state transitions can be indicated by a 4-tuple (R_h, T_h, R_a, T_a). Specifically, R_h (respectively, R_a) is the key-block rewards for honest (respectively, the selfish) miners, while T_h (respectively, T_a) is the transaction fee for honest (respectively, the selfish) miners.

Recall that there are two types of transactions. Here, the analysis focuses on regular transactions and will discuss whale transactions later. Recall also that the microblock fee of a regular microblock is denoted by R_t. For convenience, instead of recording the number of rewards, each field only records the number of key-block rewards or transaction fees (the total transaction fee in v/f microblocks as one unit) won by miners. More importantly, the transaction fees included in the microblocks after the common ancestor key block are not assigned to miners until the next ancestor key block is decided. This is because these transaction fees are affected by some future actions of the selfish miner (see Sect. 4.3).

In adopt or adoptE actions, the selfish miner accepts l_h honest key blocks and the microblocks mined before these key blocks. Honest miners obtain $l_h R_b$ key-block rewards and $(l_h - 1)v/f R_t$ transaction fees. In override or overrideH actions, the selfish miner publishes $l_h + 1$ selfish key blocks. Honest miners accept these key blocks and the microblocks produced before the key blocks. Thus, the selfish miner obtains $(l_h + 1)R_b$ key-block rewards and $l_h v/f R_t$ transaction fees. In the match actions, the next state depends on whether the next key block is created by the selfish miner (w.p. α), by some honest miners working on the honest branch (w.p. $(1-\gamma)(1-\alpha)$), or by the left honest miners mining on the selfish branch (w.p. $\gamma(1-\alpha)$). In the latter case, the selfish miner effectively overrides the honest miners' branch. It can obtain $l_h R_b$ key-block reward and $(l_h - 1)v/f R_t$ transaction fees. Note that the value of γ is decided by the adopted fork solution (e.g., $\gamma = 0.5$ in the uniform tie-break policy).

Once the common ancestor key block is changed, the transaction fees in the microblocks produced after the previous ancestor key block will be assigned. There are two cases:

- The previous common ancestor key block is mined by an honest miner. This case can be further divided into two subcases: (1) the next key block is mined by honest miners, and honest miners get $v/f R_t$ transaction fees; (2) the next key block is mined by the selfish miner and **lastMicroBlock** $= H_{\mathsf{in}}$, honest miners get $rv/f R_t$ transaction fees and the selfish miner gets $(1-r)v/f R_t$ transaction fees.
- The previous common ancestor key block is mined by the selfish miner. This case can be further divided into two subcases: (1) the next key block is mined by the selfish miner, and the selfish miner gets $v/f R_t$ transaction fees; (2) the next key block is mined by some honest miners and **lastMicroBlock** $= S_{\mathsf{p}}$, the selfish miner gets $rv/f R_t$ transaction fees and honest miners get $(1-r)v/f R_t$ transaction fees.

Note that since whale transactions are rare and unpredictable, the microblock fee can be modeled as a random variable taking two values: R_t or R_t plus the fee of

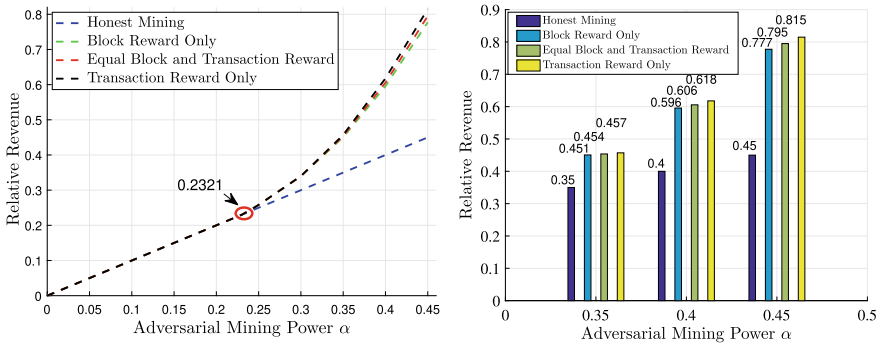

Fig. 18 The results of the selfish mining attack in Ethereum. **a** The selfish miner's relative revenue with different α. **b** The threshold of the selfish mining attack with different β

a whale transaction. Let $\bar{R}_t$ be the expected microblock fee. Clearly, $\bar{R}_t > R_t$. The long-term effect of whale transactions is to decrease the ratio k from R_b/R_t to $R_b/\bar{R}_t$. Such an effect slightly increases the relative revenue of the selfish miner.

By using MDP toolbox [7] to solve the above MDP model, we can numerically obtain the optimal policies under each scenario and obtain the selfish miner's relative revenue. To better illustrate the results, we first plot the relative revenues of the selfish miner when $r = 0.4$ (used in Bitcoin-NG [10]) with different α, as shown in Fig. 18a. Here, three reward settings: $k \to 0$, $k = v/f$, and $k \to \infty$ are considered. Specifically, in the first setting, the transaction fees dominate the miners' revenue; in the second setting, the transaction fees included in v/f microblocks between two consecutive key blocks have the same weight with one key-block reward; in the third setting, the key-block rewards dominate miners' revenue. Note that the key-block reward-dominated case has a similar reward distribution as Bitcoin, i.e., the microblock architecture does not impact the system. The figure shows that the thresholds of making selfish mining profitable in these three settings are all 23.21%, which is the same as the selfish mining threshold in Bitcoin. In other words, by adopting the suitable r (i.e., $\alpha < r < 1 - \alpha$), the microblock architecture in Bitcoin-NG does not affect the system security compared with Bitcoin. In addition, the selfish miner's revenues in the three settings are still the same even when $\alpha > 29\%$, which verifies the analysis in Sect. 4.3 and supports that Bitcoin-NG is as resilient as Bitcoin under suitable settings. When $\alpha > 35\%$, the differences between the selfish miner's revenues in the three settings and the honest revenue are exhibited in Fig. 18b. It's easy to see that the selfish miner can obtain the highest revenue in the transaction fee-dominated case. This implies that the microblock architecture can slightly increase the selfish miner's revenue.

5 Further Reading

In this section, we provide more works on the incentive design and analysis of blockchains for interested readers.

Bitcoin. Eyal and Sirer are among the first to formally analyze selfish mining in Bitcoin [11]. However, the proposed selfish mining strategy isn't optimal. Later, Sapirshtein et al. [33] and Nayak et al. [25] demonstrated that by adopting optimized strategies, the threshold of the computation power to make selfish mining profitable can be reduced to 23.2% (instead of 25% in [11]) when honest miners adopt the uniform tie-breaking defense. In [17], Gervais et al. further extended the analysis to several variants of Bitcoin including Dogecoin, Litecoin, and Ethereum. Different with these works on block rewards, the selfish mining strategy of transaction fees was studied in [6]. The results showed that even an attacker with small computation power and a poor network connection can still gain more profits from the attack. Based on this work, Tsabary and Eyal [37] additionally studied the Bitcoin gap game between block reward and transaction fee. In [19], the propagation delay is considered for the analysis of selfish mining.

In addition to the incentive analysis, there are several works on defending the selfish mining attack in Bitcoin. Heilman proposed a defense mechanism called Freshness Preferred [20], in which by using the latest unforgeable timestamp issued by a trusted party, the threshold can be increased to 32%. Bahack in [2] introduced a fork-punishment rule to make selfish mining unprofitable. Specifically, each miner in the system can include fork evidence in their block. Once confirmed, the miner can get half of the total rewards of the winning branch. Solat and Potop-Butucaru [34] proposed a solution called ZeroBlock, which can make selfish miners' block expire and be rejected by all the honest miners without using forgeable timestamps. In [43], a backward-compatible defense mechanism called weighted FRP was proposed, in which the weights of the forked chains instead of their lengths are considered. In [28], Pass and Shi proposed Fruitchains, which distributes rewards to all recent fruits that are parallel products of block mining. Similar with Fruitchains, Szalachowski et al. [36] proposed a new protocol, called StrongChain, which enables miners to publish weak solutions, i.e., solutions with higher mining difficulty targets. Miners can include weak solutions in their blocks and always select the chain with the largest weighted count of blocks and weak solutions to mine on. Bissias and Levine [3] proposed Bobtail, which enables miners to publish and collect all PoW solutions with a higher target until the mean of the k smallest hashes is below a certain target.

Ethereum. In [30], Ritz and Zugenmaier conducted extensive simulations to study selfish mining in Ethereum. Wang et al. [38] analyzed two kinds of stubborn mining in Ethereum. Yang et al. [40] analyzed the impact of imperfect network on selfish mining in Ethereum. Besides, Ethereum has updated its transaction fee mechanism in EIP1559 [4], which make it quite different with that in Bitcoin. Several studies on this new transaction mechanism were conducted [13, 23, 31, 32].

Other Blockchains. Yin et al. [41] have extended the incentive analysis of Bitcoin-NG by considering a situation that the original paper omits [10]. Later, Wang at

al. [39] considered advanced selfish mining strategies, i.e., stubborn mining strategies, when an attacker may manipulate the microblock chains between two honest parties. Fooladgaret al. [14] modeled the participation costs and rewards received within a strategic interaction scenario in Algorand [18]. They showed that the reward sharing approach in Algorand is not a Nash equilibrium and proposed a novel reward mechanism to fix it.

6 Conclusion

In this chapter, we revisit the incentive design and analysis of three influential blockchain protocols: Bitcoin, Ethereum, and Bitcoin-NG. In particular, compared with Bitcoin, Ethereum introduces two new rewards (i.e., uncle and nephew block rewards), while Bitcoin-NG redesigns the transaction fee distribution rule to accommodate the new consensus architecture. Throughout the studies, we first find that incentive designs are closely related with the system security. Hence, incentive design should be carefully evaluated before adoption. Second, we find that new consensus protocols also require new incentive designs. Therefore, consensus algorithms and incentive mechanisms should be jointly considered and evaluated for blockchain protocols. Third, we find that existing incentive analysis may not be feasible for new designs, and so tailored analysis is required. However, the process of modeling and theoretical analysis make it difficult to evaluate the incentive design for each blockchain protocol. Thus, AI-driven methods such as deep reinforcement learning (DRL) may help to automatically analyze incentive designs and to conduct the analysis. We notice that there are several works on this direction and leave discussion of these automatic analysis as future work.

References

1. Bagaria, V., Kannan, S., Tse, D., Fanti, G., Viswanath, P.: Deconstructing the blockchain to approach physical limits (2018). arXiv preprint arXiv:1810.08092
2. Bahack, L.: Theoretical Bitcoin attacks with less than half of the computational power (draft) (2013). arXiv preprint arXiv:1312.7013
3. Bissias, G., Levine, B.N.: Bobtail: improved blockchain security with low-variance mining. In: The Network and Distributed System Security Symposium (NDSS), NDSS '20 (2020)
4. Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I., Bakhta, A.: EIP-1559: fee market change for ETH 1.0 chain (2019). https://eips.ethereum.org/EIPS/eip-1559
5. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. White paper (2014)
6. Carlsten, M., Kalodner, H., Matthew Weinberg, S., Narayanan, A.: On the instability of Bitcoin without the block reward. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pp. 154–167. ACM (2016)
7. Chadès, I., Chapron, G., Cros, M.-J., Garcia, F., Sabbadin, R.: Mdptoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems. Ecography **37**(9), 916–920 (2014)
8. Ethereum. Design rationale: uncle incentivizatinon. github (2018)

9. Ethereum. Mining rewards (2018)
10. Eyal, I., Gencer, A.E., Sirer, E.G., Van Renesse, R.: Bitcoin-NG: a scalable blockchain protocol. In: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), pp. 45–59 (2016)
11. Eyal, I., Sirer, E.G.: Majority is not enough: bitcoin mining is vulnerable. Commun. ACM **61**(7), 95–102 (2018)
12. Feng, C., Niu, J.: Selfish mining in ethereum. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 1306–1316 (2019)
13. Ferreira, M.V.X., Moroz, D.J., Parkes, D.C., Stern, M.: Dynamic posted-price mechanisms for the blockchain transaction-fee market (2012). arXiv preprint arXiv:2103.14144
14. Fooladgar, M., Manshaei, M.H., Jadliwala, M., Rahman, M.A.: On incentive compatible role-based reward distribution in algorand. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 452–463. IEEE (2020)
15. Forum, Bitcoin: mining cartel attack, December 2010
16. Gervais, A., Karame, G.O., Capkun, V., Capkun, S.: Is Bitcoin a decentralized currency? IEEE Secur. Privacy **12**(3), 54–60 (2014)
17. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, pp. 3–16. ACM (2016)
18. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling Byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, SOSP '17, pp. 51–68. ACM (2017)
19. Göbel, J., Keeler, H.P., Krzesinski, A.E., Taylor, P.G.: Bitcoin blockchain dynamics: the selfish-mine strategy in the presence of propagation delay. Perform. Eval. **104**, 23–41 (2016)
20. Heilman, E.: One weird trick to stop selfish miners: fresh bitcoins, a solution for the honest miner. In: International Conference on Financial Cryptography and Data Security, pp. 161–162. Springer (2014)
21. Hoenicke, J.: Unconfirmed transaction count (mempool) (2020)
22. Kogias, E.K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Bryan, F.: Enhancing bitcoin security and performance with strong consistency via collective signing. In: 25th USENIX Security Symposium (USENIX Security 16), pp. 279–296 (2016)
23. Leonardos, S., Monnot, B., Reijsbergen, D., Skoulakis, S., Georgios, P.: Dynamical analysis of the EIP-1559 ethereum fee market (2021). arXiv preprint arXiv:2102.10567
24. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. Working paper (2008)
25. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: generalizing selfish mining and combining with an eclipse attack. In: 2016 IEEE European Symposium on Security and Privacy (EuroS P), pp. 305–320. IEEE (2016)
26. Niu, J., Feng, C., Dau, H., Huang, Y.-C., Jingge, Z.: Analysis of Nakamoto consensus, revisited (2019). arXiv preprint arXiv:1910.08510
27. Niu, J., Wang, Z., Gai, F., Feng, C.: Incentive analysis of Bitcoin-NG, revisited. In: Performance Evaluation: An International Journal, vol. 144, pp. 102–144. Elsevier (2020)
28. Pass, R., Shi, E.: Fruitchains: a fair blockchain. In: Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC '17, pp. 315–324. ACM (2017)
29. Pass, R., Shi, E.: Hybrid consensus: efficient consensus in the permissionless model. In: 31st International Symposium on Distributed Computing (DISC 2017), vol. 91, pp. 39:1–39:16 (2017)
30. Ritz, F., Zugenmaier, A.: The impact of uncle rewards on selfish mining in Ethereum. In: 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS PW), pp. 50–57. IEEE (2018)
31. Roughgarden, T.: Transaction fee mechanism design for the ethereum blockchain: An economic analysis of EIP-1559 (2020). arXiv preprint arXiv:2012.00854
32. Roughgarden, T.: Transaction fee mechanism design (2021). arXiv preprint arXiv:2106.01340
33. Sapirshtein, A., Sompolinsky, Y., Zohar, A.: Optimal selfish mining strategies in Bitcoin. In: International Conference on Financial Cryptography and Data Security, pp. 515–532. Springer (2016)

34. Solat, S., Potop-Butucaru, M.: Zeroblock: preventing selfish mining in Bitcoin. Technical report, Sorbonne Universites (2016)
35. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. In: Financial Cryptography and Data Security, pp. 507–527. Springer (2015)
36. Szalachowski, P., Reijsbergen, D., Homoliak, I., Sun, S.: Strongchain: transparent and collaborative proof-of-work consensus. In: Proceedings of the 28th USENIX Conference on Security Symposium, SEC'19, pp. 819–836 (2019)
37. Tsabary, I., Eyal, I.: The gap game. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS' 18, pp. 713–728 (2018)
38. Wang, Z., Liu, J., Qianhong, W., Zhang, Y., Hui, Yu., Zhou, Z.: An analytic evaluation for the impact of uncle blocks by selfish and stubborn mining in an imperfect ethereum network. Comput. Secur. **87**, 101581 (2019)
39. Wang, Z., Liu, J., Zhang, Z., Zhang, Y., Yin, J., Yu, H., Liu, W.: A combined micro-block chain truncation attack on Bitcoin-NG. In: Information Security and Privacy, pp. 322–339. Springer (2019)
40. Yang, R., Chang, X., Mišić, J., B Mišić, V.: Assessing blockchain selfish mining in an imperfect network: honest and selfish miner views. Comput. Secur. **97**, 101956 (2020)
41. Yin, J., Wang, C., Zhang, Z., Liu, J.: Revisiting the incentive mechanism of Bitcoin-ng. In: Information Security and Privacy, pp. 706–719. Springer (2018)
42. Yu, H., Nikolic, I., Hou, R., Saxena, P.: OHIE: Blockchain scaling made simple. In: Proceedings of the 41th IEEE Symposium on Security and Privacy, S&P. IEEE (2020)
43. Zhang, R., Preneel, B.: Publish or perish: a backward-compatible defense against selfish mining in Bitcoin. In: Cryptographers' Track at the RSA Conference, pp. 277–292. Springer (2017)