An Eye for an Eye: Economics of Retaliation in Mining Pools

Yujin Kwon*, Hyoungshick Kim†, Yung Yi*, Yongdae Kim*

*KAIST
{dbwls8724,yiyung,yongdaek}@kaist.ac.kr

†Sungkyunkwan University
hyoung@skku.edu

ABSTRACT

Currently, miners typically join *mining pools* to solve cryptographic puzzles together, and mining pools are in high competition. This has led to the development of several attack strategies such as *block withholding* (BWH) and *fork after withholding* (FAW) attacks that can weaken the health of PoW systems and but maximize mining pools' profits. In this paper, we present strategies called Adaptive Retaliation Strategies (ARS) to mitigate not only BWH attacks but also FAW attacks. In ARS, each pool cooperates with other pools in the normal situation, and adaptively executes either FAW or BWH attacks for the purpose of retaliation only when attacked. In addition, in order for rational pools to adopt ARS, ARS should strike to an adaptive balance between retaliation and selfishness because the pools consider their payoff even when they retaliate. We theoretically and numerically show that ARS would not only lead to the induction of a no-attack state among mining pools, but also achieve the adaptive balance between retaliation and selfishness.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; **Economics of security and privacy**;

KEYWORDS

Bitcoin; Mining; Fork After Withholding Attack; Block Withholding Attack; Repeated Game

1 INTRODUCTION

Numerous cryptocurrencies based on a peer-to-peer network now exist, utilizing an open ledger called a *blockchain*. In such blockchain systems, network nodes called *miners* verify the transactions collected through the network, generate a block consisting of valid transactions, and propagate the block to the entire network. In general, (public) blockchain systems offer financial incentives to encourage miners to participate in this process. For example, in Bitcoin, a miner is currently rewarded with 12.5 BTC for a new block creation.

AFT '19, October 21–23, 2019, Zurich, Switzerland

ACM ISBN 978-1-4503-6732-5/19/10…$15.00
https://doi.org/10.1145/3318041.3355472

The security and reliability of blockchain systems depend on their consensus algorithm. Most popular cryptocurrency systems such as Bitcoin and Ethereum adopt a *proof-of-work* (PoW) mechanism in order to agree on the same blockchain [27]. In such PoW mechanisms, miners must solve cryptographic puzzles (i.e., *proofs-of-work*) showing that a certain amount of computational resources (e.g., time and memory) was spent in order to generate a new block. The mining difficulty is adjusted automatically to maintain an average mining rate of one block at a fixed time interval with changes in the total computational power of the blockchain system.

The significant increase in mining difficulty led the formation of *mining pools* in which miners gather to mine together; these pools perform mining as a single node in a network. Most pools consist of a manager and miners, and the manager assigns a puzzle to miners at the beginning of every round. The miners then find partial proofs-of-work (PPoWs) and full proofs-of-work (FPoWs) for a given puzzle and submit them to the manager. PPoWs are needed for assessing each miner's contribution to share the reward in the pool. If a miner fully solves the puzzle, the manager generates and propagates a block based on the submitted PoW. The manager then earns the reward for one block and splits it among miners in proportion to their number of submitted PPoWs.

However, previous studies [17, 28] demonstrated how existing mining pools' protocols can be vulnerable to *block withholding* (BWH) [28] and *fork after withholding* (FAW) attacks [17]. Moreover, pools in high competition can launch these attacks against each other by infiltrating a part of their mining power (i.e., computational power) into other pools. A BWH attacking pool first infiltrates its mining power into other pools (i.e., victim pools) and submits only PPoWs but not FPoWs to the victim pools. The FAW attack is an extended attack of the BWH attack. Similar to the BWH attack, a FAW attacking pool also infiltrates its mining power into other pools, and submits only PPoWs but not FPoWs to the victim pools except for the case when an external honest miner (i.e., neither the attacker nor a miner in the victim pool) propagates a block. Unlike the BWH attack, in the case, the attacker *intentionally* generates forks through the victim pools.

To analyze the strategic interaction between miners for those two attacks, we can use game-theoretic models: the BWH game and the FAW game. Eyal [9] showed that the BWH game between two pools where they can execute BWH attacks is similar to the prisoner's dilemma. For the FAW game between two pools where they can execute FAW attacks, Kwon et al. [17] showed that mining pools' strategies form a Nash equilibrium in which a larger pool among two pools earns extra profit. Thus, FAW attacks between pools can cause more centralization of the system because miners

would join in large pools that earn the extra reward through the FAW attack. *The objective of this study is to find strategies inducing a no-attack state where FAW and BWH attacks do not occur.*

System model: To achieve this goal, we take a more macroscopic view of the system by modeling long-term interactions between two pools as a notion of repeated game, which we call a *repeated FAW-BWH game*, and by considering both FAW and BWH attacks together in one framework. In the repeated game, the *FAW-BWH game* is repeated as a one-stage game over time, where in every stage each rational pool makes a decision to cooperate or execute FAW or BWH attacks (see Section 3 for a full description of our model). Unlike previous studies [9, 17] focusing on a single stage game, we model a game with multiple stages to analyze the effects of long-term interactions among pools. Also, we considered both FAW and BWH attack strategies while they considered a single attack strategy (i.e., FAW or BWH attack) only.

A novel strategy ARS: In game theory, the *iterated prisoner's dilemma* (i.e., a repeated version of the prisoner's dilemma) has been extensively studied in terms of how rational players can cooperate through retaliation [1]. Using this retaliation concept, we find how rational pools can cooperate in the repeated FAW-BWH game. Unlike the iterated prisoner's dilemma, the repeated FAW-BWH game leads to a situation where a larger pool always wins the game (i.e., the pool size game). This occurs when two pools execute FAW attacks against each other as in the *FAW game*. Therefore, it is relatively under-explored to find a cooperation-inducing strategy in such a situation. In this paper, we propose strategies called Adaptive Retaliation Strategies (ARS), which can lead to the state of no-attack between two pools. In ARS, a pool cooperates at first and continues to cooperate, but executes attacks for retaliation only when attacked. Here, ARS must achieve a good balance between retaliation and selfishness because rational pools would consider their payoff even when they retaliate. In other words, if retaliation is costly, they would not follow ARS, which implies that we should find a *credible retaliation*. This is done by ARS' adaptive retaliation, i.e., adaptively deciding the amount of infiltration power for retaliation against FAW or BWH attacks.

We formally describe ARS and prove that 1) ARS leads rational pools to cooperate and 2) pools are likely to adopt ARS, by using a popular concept of equilibrium in repeated game theory, called subgame perfect Nash equilibrium (see Section 4). Furthermore, our numerical analysis demonstrates that ARS makes the FAW and BWH attacks unprofitable (see Section 5).

Practical requirements for ARS: To apply ARS in real-world settings, pools should be able to monitor other pools' information about whether an attacker executes FAW or BWH attacks, and if so, how much the attacker's infiltration power is used. To monitor these information, victim pool's some statistical properties can be used. For example, a pool can detect FAW and BWH attacks by observing its fork rate and the ratio between the number of submitted PPoWs and FPoWs, respectively. It can also determine the attacker's infiltration power in the victim pool through this detection method. Another prerequisite for ARS is to identify which pools have attacked a victim pool, and this may take a long time. Therefore, we propose investigating the variance of the reward densities of pools through moles to reduce the identifying time (see Section 6). The benign pool's reward density is not at all related with block rewards of other pools. Meanwhile, when a pool executes either FAW or BWH attacks, its reward density would be correlated with block rewards of other pools because the attacker earns part of its reward from the victim. This implies that we can identity the pools executing attacks based on this correlation information (see Section 6).

Multiple pools: In addition, for generalization, we extend ARS to that for multiple pools beyond two pools. Through a case study, we show that ARS still makes FAW and BWH attacks unprofitable in the game among multiple pools (see Section 7).

2 BACKGROUND

In current cryptocurrencies, peers verify transactions issued by clients. Peers record the verified transactions in a "blockchain." To maintain the blockchain, many cryptocurrencies including Bitcoin adopt the PoW mechanism. In this section, we describe the mining process, focusing on Bitcoin. Further, we demonstrate FAW and BWH attacks against mining pools.

2.1 Bitcoin Basics

Mining Process: For mining, miners first gather issued transactions, which are not yet recorded in the blockchain, into their local storage. Then, miners place the transactions into a block and find a PoW, spending their computational power to generate a valid block. The header of a block includes a Merkle root [23] of transactions in the corresponding block and the hash value of the header of the previous block. The block header also includes a nonce, which is a key component necessary to become a valid block.

To be a valid block, the hash value of block header must be less than a given target number T_1. In particular, Bitcoin uses the double SHA256 hash function. The hash value of a block header is obtained as an output of double SHA256 for an input containing a concatenation of block contents including a nonce. Miners increment a nonce to find a valid nonce, which makes the hash value less than the target number T_1. If a miner finds a valid nonce as a PoW, the miner generates the valid block including the nonce and propagates the block to a peer-to-peer network. Finally, the block is appended to the blockchain, and the above mining process is repeated.

In Bitcoin, the target number T_1 is adjusted every 2016 blocks to keep the average period of one block generation (i.e, the average period of one round) at 10 mins. The smaller the value of T_1, the more difficult the mining process will be.

Forks: When a miner A propagates a block, another block can also be generated and propagated by a miner B who has not yet received A's block. Therefore, miners receive two blocks and regard the first received block as the blockchain head. This situation is called a *fork*. When a fork occurs, only one block becomes valid. Moreover, an attacker can generate intentionally forks to execute double-spending [15, 25] and selfish mining [11, 13, 24, 29].

Mining Pools: As the mining difficulty has been increasing, mining pools were introduced, in which miners gather to mine together. Major pools consist of a manager and many miners. These pools run as one node in the Bitcoin network, and pool miners only need to connect to the manager and create IDs. The manager forms and distributes a potential block to miners, and then miners spend their

computational power to generate a valid nonce based on the block form provided by the manager. Moreover, there are *open* and *closed* pools, depending on the policy whether any miner can join or not.

Pools' reward systems are different from the block reward system in Bitcoin. Miners in pools receive rewards for nonces, which make the hash value of the block header less than a new target number T_2. The number T_2 is greater than the original target number T_1. We refer to a nonce for the target T_2 and T_1 as a partial proof-of-work (PPoW) and a full proof-of-work (FPoW), respectively. When a miner finds a PPoW or an FPoW, the miner submits it (called *shares*) to the manager, where PPoWs are needed to assess each miner's contribution in order to share the reward in the pool. When the submitted share is FPoW, the manager generates a valid block and earns the block reward. The manager then distributes the block reward to miners in proportion to the number of submitted shares.

2.2 Existing Attacks on Pools

Block Withholding: To execute the BWH attack [7, 28], an attacker splits her computational power into two parts, which are used for solo mining and malicious mining in the victim pool, respectively. In the malicious mining, she submits only a PPoW to the victim without submitting an FPoW. Although she undermines the victim by withholding blocks in the victim pool, she still earns a portion of the reward through PPoWs submitted to the victim. In addition, her solo mining has a higher efficiency compared to the case where she does not attack because the block reward is gained in proportion to each pool's *relative* computational power in Bitcoin. In other words, by undermining the victim pool, her relative computational power would increase. As a result, this point allows the attacker to earn an extra reward. Naturally, the BWH attack can be executed in many proof-of-work cryptocurrencies including Bitcoin, Ethereum, and Litecoin.

In 2015, Eyal [9] developed a BWH game between two pools. In the game, each pool can launch the BWH attack against an opponent by infiltrating a part of the computational power into the opponent. Eyal found that the BWH game results in *the miner's dilemma*. In other words, there is only one Nash equilibrium in which two pools execute BWH attacks against each other and both pools suffer losses.

Fork After Withholding: For the FAW attack proposed in 2017 [17], similar to the BWH attack, an attacker splits her computational power into two parts, which are used for her solo mining and malicious mining in a victim pool. However, when the attacker finds an FPoW in the victim pool, she submits the FPoW to the manager unlike in the BWH attack. This occurs only if an external honest miner (i.e., neither the attacker nor a miner in the victim pool) propagates a block. Therefore, the attacker intentionally generates a fork through pools. In the FAW game, pools can launch FAW attacks against each other by infiltrating a portion of their computational power into the other pools. There is unique Nash equilibrium, where two pools execute the FAW attack against each other. In the equilibrium, a larger pool earns an extra reward (unlike in the BWH game) while a smaller pool suffers a loss. In other words, the game leads to a *pool size game*. This fact can make the decentralization level of Bitcoin decrease when occurring FAW attacks among pools.

Countermeasure: Because no viable countermeasure against FAW and BWH attacks [9, 17] exists, the attacks can be launched in practice. Indeed, the BWH attack was executed against the "Eligius" pool in 2014. To detect the attack, a pool's manager can investigate the ratio of FPoWs to PPoWs. If the ratio is low enough, the manager can speculate that the BWH attack has occurred in his pool. However, identification of the attacker is known to be difficult if she executes BWH attacks using many Sybil nodes (IDs) in the pool.

For the FAW attack, miners can detect it because the attack will cause a high fork rate. However, it is still difficult to identify the attacker because she indirectly generate intentional forks through pools instead of generating them by herself. Of course, in the victim pool, the manager can expel any miners suspected of causing forks. Nonetheless, the attacker's reward can be unaffected by this manager's behavior by planting many Sybil IDs in the victim pool. In other words, even though an ID that generates a fork is expelled by the manager, the attacker still earns the extra reward though other IDs. Eventually, the manager would be unable to prevent FAW attacks with a simple blacklist of suspects.

Even though many papers [8, 10, 28] proposed a new PoW mechanism that can prevent BWH and FAW attacks, they might be impractical. These protocols make pool miners not know whether their found nonce can generate a valid block, and thus the miners cannot execute BWH and FAW attacks. However, this point causes another withholding attack of a manager, where she can stealthily withhold blocks found by a pool miner and earn extra profit through her solo mining. In addition, the protocols increase the pool operation cost. The above facts make the protocols impractical [17].

3 MODEL AND FORMULATION

3.1 System Model

Block generation in PoW: In PoW mechanisms, miners attempt to generate valid blocks by finding an inverse image of a hash function satisfying a certain condition in each round, where one round is defined as the time during which a new task is generated and a valid block is found by a miner. Due to the pseudorandomness of hash functions, we assume that the number of blocks found by a miner for one round follows a Poisson distribution with the miner's relative computational power. Then, the number of blocks found by a pool also follows a Poisson distribution because the sum of Poisson random variables is a Poisson random variable. For simplicity, we assume that natural forks do not occur in the block generating process, as the probability (≈0.004 [13]) of natural forks occurring is significantly low in practice [9, 13, 17].

Computation power and reward: We let $\mathcal{I}$ be the set of all pools and solo miners[1], and denote by α_i the computational power of $i \in \mathcal{I}$. For analytical convenience, we normalize the total computation power with 1, and thus $\sum_{i\in\mathcal{I}} \alpha_i = 1$. We assume that a node cannot possess more than 50% computational power (i.e., $\alpha_i \leq 0.5$) as in the previous works. A reward for one block is assumed to be 1, implying that the total reward of a node i in a round is simply the total number of blocks found by i in that round. When a pool finds a block and earns the reward for the block, the pool manager

[1] A solo miner directly conducts mining, not joining pools.

distributes the reward to miners in proportion to the number of shares submitted for the time duration over which the pool finds the block. We also assume that the manager honestly distributes the reward among the pool's miners. We define node i's *reward density* at round r as $\frac{R_i^r}{\alpha_i}$ where R_i^r is i's reward earned for round r.

Attacks: We consider a case in which only two types of attacks, FAW and BWH, can be executed by pools. In addition, because most pools are open pools, we consider only open pools, not closed pools. Closed pools will be discussed in Section 8. For the worst case analysis, we assume that the FAW attack is executed under the best network capability[2], meaning that the attacker's blocks always become valid after forks caused by the FAW attack. During an attack, we assume that an attacking pool executes either FAW or BWH attacks, but not both at the same time. Indeed, an attack combining FAW and BWH is equivalent to the FAW attack under some network capability. An attacking pool infiltrates a part of its computation power into "victim" pools. Note that such a choice on the attack strategy can be time-varying (see **stage** in the next paragraph). We consider a regime in which there are a sufficient number of miners in each pool, so as to assume that each pool's infiltration power used for attacks is a real number. Moreover, an attacker infiltrates its partial computational power into a victim pool using Sybil IDs in order to make it more difficult to identify who the attacker is. Nonetheless, we assume that the victim can trace the attacker and know how much infiltration power the attacker has used for attacks because the attacking pool is an open pool. In Section 6, we describe how this becomes possible in practice.

Stage: Our interest lies in investigating how pools interact over a long-term period. To this end, the entire time is divided into a sequence of *stages*, where over each stage a pool can know whether an attack occurs against itself, how much infiltration power is being used, and who the attacker is. This notion of stage is the one that is popularly used in repeated games (see Section 3.2). Therefore, at the end of each stage, a victim identifies an attacker when an attack was executed against the victim during the stage. Note that a stage consists of multiple rounds. At the start of each stage, pools can change their actions based on other pools' actions. For analytical tractability, we assume that stages are synchronized among pools.

3.2 Repeated FAW-BWH Game

Primer on repeated game: We aim at modeling how multiple pools interact to achieve their own objectives over a *long-term* period. To this end, we use the theory of *repeated games*, popularly used to understand the long-term interactions among players. In repeated games, the interactions among players repeat for multiple stages, and the players become aware of other players' past behaviors and their future benefits, accordingly adapting their strategies over time. The main idea of the theory of repeated games is that a player may be deterred from exploiting her short-term advantage by the threat of punishment that reduces her long-term payoff.

The basic component of a repeated game is a (simultaneous-move) stage game G played for each stage. The stage game G is represented by $< \mathcal{N}, (A_i)_{i\in\mathcal{N}}, (U_i)_{i\in\mathcal{N}} >$, where $\mathcal{N}$ is the set of players, A_i is the set of actions of the player i, and $U_i(a)$ is a player i's payoff function when the players' action profile is $a \in (A_i)_{i\in\mathcal{N}}$. We denote by G^T the T-period repeated game of G with perfect information, where the players play the same stage game G for T stages, possibly $T = \infty$. We use the superscript t to express the stage t in all notations, and let $a^t := (a_i^t)_{i\in\mathcal{N}}$ denote the action profile at stage t, i.e., the actions by all players at stage t. Also, we denote by $a_i := (a_i^t)_{0\le t\le T}$ the actions of player i for T stages. For $t \ge 1$, let $h^t = (a^0, a^1, \ldots, a^{t-1})$ denote the history up to stage $t-1$, where H^t is the space of all possible stage-t histories. Depending on whether $T < \infty$ or $T = \infty$, we call each case as a finitely or infinitely repeated game. At each stage t, each player i knows all past actions h^{t-1} and chooses the next action a_i^t according to i's strategy, thus $a^t = (a_i^t)_{i\in\mathcal{N}}$ is determined. Here, a *strategy* s_i for player i in the repeated game is a sequence of maps s_i^t—one for each stage t—that map a $(t-1)$-history h^{t-1} to an action a_i^t in A_i. By perfect information, we mean that at the end of each stage game, players are able to know other players' actions and their payoffs. In this paper, we focus on the infinitely repeated game, in which given the whole action profiles $\boldsymbol{a} = (a_1, a_2, \ldots)$, the payoff $\mathcal{U}_i(\boldsymbol{a})$ of player i in the corresponding repeated game is the discounted average payoff, i.e.,

$$\mathcal{U}_i(\boldsymbol{a}) = \sum_{t=1}^{\infty} \delta^{t-1} U_i(a^t), \tag{1}$$

where $0 < \delta < 1$ is the discount factor. The discount factor indicates how much a player discounts in the future. Next, we introduce the concept of the subgame perfect Nash equilibrium.

Definition 3.1 (Subgame Perfect Nash Equilibrium (SPNE)). For a given history h^t and player i's strategy s_i, we denote the discounted average payoff of player i in the subgame given the history h^t as $U_i(s_i, \boldsymbol{s_{-i}} \mid h^t)$. Then the strategy profile $\boldsymbol{s^\star} = (s_i^\star)_{i\in\mathcal{N}}$ is a subgame perfect Nash equilibrium if

$$\mathcal{U}_i(s_i^\star, \boldsymbol{s_{-i}^\star} \mid h^t) = \max_{s_i' \in S_i} \mathcal{U}_i(s_i', \boldsymbol{s_{-i}^\star} \mid h^t) \text{ for } \forall i \in \mathcal{N}, \forall h^t \in H^t, \forall t > 0,$$

where S_i is a space of player i's strategies.

In the repeated game, SPNE is a stronger version of Nash equilibrium, roughly meaning a strategy profile, which is a Nash equilibrium in *every* subgame. Thus, a SPNE is regarded as a mathematically-proved strategy vector that rational players are likely to follow when players interact over a long-term time period. There is also Folk Theorem [12] that states the existence of SPNE outcomes under a certain condition. More specifically, it represents that if there is a *credible retaliation*, it is likely to achieve cooperation among rational players, where a credible retaliation indicates that is not costly for a retaliator.

Repeated FAW-BWH game: As in the previous studies [9, 17], for simplicity we consider a game only between two pools (Pool_1 and Pool_2). As mentioned in Section 3.1, we define a stage as the duration of time when each pool is able to trace other pools' attacking behaviors, which enables us to obtain the condition of perfect information in our analytical model.

In defining the repeated FAW-BWH game, it is crucial to define a stage game G, for which we now describe how we model the set of actions $(A_i)_{i=1,2}$ and the payoff function $U_i(\cdot)$. First, each pool i's action is defined in terms of which attack is performed and the

[2] A variable $0 \le c \le 1$ represents the network capability [17], where we assume $c = 1$ in this study. This assumption, which is made for the sake of simplicity, can be relaxed readily.

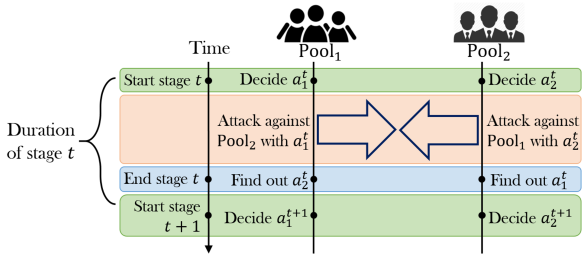


Figure 1: At the start of stage t, two pools can decide their strategy a_i^t. They then know the opponent's strategy a_{-i}^t at the end of stage t. This process is repeated at a new stage $t+1$.

amount of infiltration power used. We assume that a pool's attack is homogeneous, i.e., it executes either FAW attack or BWH attack. However, a pool is able to change its attack across stages. Then, in this stage game, each pool i's action space is no-attack, FAW, or BWH, with a choice of some infiltration power for each attack. More formally, at stage t, each pool i's action can be expressed as a vector $a_i^t = (f_i^t, b_i^t)$, where f_i^t and b_i^t are the infiltration powers for each FAW and BWH attack, respectively ($0 \le f_i^t, b_i^t \le \alpha_i$). It is clear that the case of $f_i^t = b_i^t = 0$ corresponds to no-attack, in which case we simply denote it by $a_i^t = \overline{0}$. Moreover, because we assume homogeneity in the attack, always only one of f_i^t or b_i^t is positive, i.e., $f_i^t \times b_i^t = 0$. Fig. 1 depicts a model of our repeated FAW-BWH game between two pools. When a stage t ends, each pool i is aware of its opponent pool's strategy (a_{-i}^t) for this stage game. Then, pools can change their action, depending on the opponent's action at the previous stage, when a new stage game starts. To complete the definition of the repeated FAW-BWH game, it remains to define the payoff function $U_i(a^t)$ in (1) at each stage game, which we will delay until Section 4.

4 ANALYSIS OF REPEATED FAW-BWH GAME

In this section, we study how two pools choose their strategies in the equilibrium of the repeated FAW-BWH game, to understand the pools' behaviors in terms of their interaction over a long time.

4.1 Payoff Function in the Stage Game

Because pools play the same game at each stage, we remove the stage index t in this section for simplicity. For an action profile $a = (a_1, a_2)$ of two pools, it seems natural to define the payoff U_i as its extra reward density with respect to 1, i.e., $U_i(a_1, a_2) = \frac{R(a_1,a_2)}{\alpha_i} - 1$, where $R(a_1, a_2)$ is the average earned reward in each round. As described in Section 3.2, each pool's action is expressed as its infiltration power for either FAW or BWH attack, i.e., $a_1 = (f_1, b_1)$ and $a_2 = (f_2, b_2)$ where either f_i or b_i is 0. Then, for convenience, we separately present four possible cases as follows: for a given profile $a = (a_1, a_2)$,

$$U_i\Big((f_i, b_i), (f_{-i}, b_{-i})\Big) = \begin{cases} U^{FF}(f_i, f_{-i}) & b_i = 0 \text{ and } b_{-i} = 0, \\ U^{FB}(f_i, b_{-i}) & b_i = 0 \text{ and } f_{-i} = 0, \\ U^{BF}(b_i, f_{-i}) & f_i = 0 \text{ and } b_{-i} = 0, \\ U^{BB}(b_i, f_{-i}) & b_i = 0 \text{ and } f_{-i} = 0, \end{cases}$$

where we henceforth provide the forms of four functions: U^{FF}, U^{BB}, U^{BF}, and U^{FB}. According to the definition of payoff U_i, if two pools do not attack, their payoffs are 0. If U_i is positive, Pool_i earns an extra reward. Otherwise, Pool_i suffers a loss.

Homogeneous attacks: The case in which two pools execute the same attack, FAW or BWH, has been studied in two related studies [9, 17]. For the FAW-FAW attack, the following payoff function can be obtained from Kwon's work [17].

$$\begin{aligned} U^{FF}(f_i, f_{-i}) = {} & \frac{\alpha_i - f_i}{(1 - f_i - f_{-i})(\alpha_i + f_{-i})} + \frac{f_{-i}(1 - \alpha_i - \alpha_{-i})}{(1 - f_{-i})(\alpha_i + f_{-i})} \\ & + \frac{f_i f_{-i}}{2}\left(\frac{1}{1 - f_i} + \frac{1}{1 - f_{-i}}\right)\frac{1 - \alpha_i - \alpha_{-i}}{(1 - f_i - f_{-i})(\alpha_i + f_{-i})} \\ & + \frac{(U^{FF}(f_{-i}, f_i) + 1) \cdot f_i}{\alpha_i + f_{-i}} - 1. \end{aligned} \tag{2}$$

In (2), the first term is obtained from the honest mining of each pool, achieved with the computational power remaining after deducting the infiltration power. Note that Pool_i gets a reward of $\frac{\alpha_i - f_i}{1 - f_i - f_{-i}}$ from the honest mining because each node earns a mining reward based on how many blocks it generated relative to others. The second term represents the extra reward density that is earned in the case where the opponent generates an intentional fork and Pool_i does not generate any block. In this case, both an external honest miner and the infiltration power of Pool_{-i} generate a block, and the probabilities of these events are $\frac{1-\alpha_i-\alpha_{-i}}{1-f_{-i}}$ and f_{-i}, respectively. This derives the second term. The third term is from intentional forks caused by both Pool_1 and Pool_2. In this case, an external honest miner and infiltration powers of Pool_1 and Pool_2 find blocks, and then a fork with three branches occurs. If the infiltration power of Pool_i finds a block faster than that for Pool_{-i}, its probability would be $f_i \cdot \frac{f_{-i}}{1-f_i} \cdot \frac{1-\alpha_i-\alpha_{-i}}{1-f_i-f_{-i}}$. On the other hand, if the infiltration power of Pool_{-i} generates a block faster than that for Pool_i, its probability would be $f_{-i} \cdot \frac{f_i}{1-f_{-i}} \cdot \frac{1-\alpha_i-\alpha_{-i}}{1-f_i-f_{-i}}$. Considering these facts, the third term is derived. Lastly, the fourth term is from its infiltration mining into the opponent and is derived from that the opponent distributes the reward of $(U^{FF}(f_{-i}, f_i) + 1) \cdot f_i$ to Pool_i. Note that Pool_i infiltrates the computational power of f_i into the opponent.

Next, we consider when two pools execute BWH attacks against each other. In this case, we have the following form of the payoff function from Eyal's work [9], where forks are not intentionally generated so the second and third terms in (2) disappear in (3).

$$U^{BB}(b_i, b_{-i}) = \frac{\alpha_i - b_i}{(1 - b_i - b_{-i})(\alpha_i + b_{-i})} + \frac{(U^{BB}(f_{-i}, f_i) + 1) \cdot b_i}{\alpha_i + b_{-i}} - 1. \tag{3}$$

Heterogeneous attacks: As opposed to the payoff functions in homogeneous attacks borrowed from previous studies [9, 17], it still remains to establish the payoff functions for when each of two pools execute FAW and BWH attacks. We first consider the case when Pool_i and Pool_{-i} execute FAW and BWH attacks, respectively. Then, the payoff $U^{FB}(f_i, b_{-i})$, which quantifies the extra reward density, turns out to be given by:

$$U^{FB}(f_i, b_{-i}) = \frac{\alpha_i - f_i}{(1 - f_i - b_{-i})(\alpha_i + b_{-i})} + \frac{(U^{BF}(b_{-i}, f_i) + 1) \cdot f_i}{\alpha_i + b_{-i}} - 1. \tag{4}$$

This payoff can be easily derived. First, the first term represents the earned reward density of Pool_i through its honest mining with the computational power remaining after deducting the infiltration power. The second term is obtained from Pool_i's infiltration mining into the opponent, Pool_{-i}. Note that (4) does not have any reward

density term earned from generated forks in Pool_i because Pool_{-i} does not generate forks in Pool_i.

Now, when Pool_i and Pool_{-i} execute BWH and FAW attacks with infiltration power b_i and f_{-i}, respectively, we have:

$$U^{BF}(b_i, f_{-i}) = \frac{\alpha_i - b_i}{(1 - b_i - f_{-i})(\alpha_i + f_{-i})} + \frac{f_{-i}}{1 - b_i} \times \frac{1 - \alpha_i - \alpha_{-i}}{(1 - b_i - f_{-i})(\alpha_i + f_{-i})} + \frac{(U^{FB}(f_{-i}, b_i) + 1) \cdot b_i}{\alpha_i + f_{-i}} - 1. \quad (5)$$

Because only Pool_{-i} executes the FAW attack, forks are generated by Pool_{-i} in only Pool_i. Thus, (5) is the addition of a similar form of (4) to the reward density (the second term) earned when forks occur in Pool_i.

4.2 Equilibrium at the Stage Game

We now discuss how two pools would behave at the equilibrium for each stage game, before we study how rational pools behave through long-term interactions in the repeated game. This step is of significant importance because (i) it clearly shows how much a near-sighted view of pools' interaction in each stage game (as in prior work [9, 17]) differs from a far-sighted one in the repeated games, and (ii) understanding the per-stage equilibrium behaviors is a key to understanding what happens if such stage games are repeated among pools. This per-stage equilibrium is stated in Theorem 4.1.

Theorem 4.1 (Nash equilibrium for stage game). *There exists a unique Nash equilibrium (NE) $a^\star = (a_i^\star, a_{-i}^\star)$ in the stage game; it is characterized as:*

$$(a_i^\star, a_{-i}^\star) = \Big((f_i^\star, 0), (f_{-i}^\star, 0)\Big), \quad \text{where} \quad f_i^\star > 0 \text{ and } f_{-i}^\star > 0. \quad (6)$$

Further, the following payoff values are obtained for different cases of two pools' computational powers α_i and α_{-i}:

$$U_i(a_i^\star, a_{-i}^\star) > 0,\ U_{-i}(a_i^\star, a_{-i}^\star) < 0 \quad \text{if} \quad \alpha_i > \alpha_{-i}, \quad (7)$$

$$U_i(a_i^\star, a_{-i}^\star) = U_{-i}(a_i^\star, a_{-i}^\star) = 0 \quad \text{if} \quad \alpha_i = \alpha_{-i}. \quad (8)$$

(see the full version [19] for our proof of the theorem.) Theorem 4.1 states the existence and the uniqueness of the Nash equilibrium, which is technically meaningful in the sense that per-stage equilibrium is predictably interpretable from a mathematical perspective. The major messages of Theorem 4.1 are: (i) when both pools are allowed to execute FAW and BWH attacks, at the Nash equilibrium, the two pools execute only FAW attacks (see (6)), and (ii) the larger pool always earns an extra reward, whereas the smaller pool always suffers a loss (see (7)). However, when they possess the same computational power, no additional reward is provided to both pools (see (8)). This is in stark contrast to the previous game where only BWH is allowed [9]. Note that Theorem 4.1 provides the *first* analysis of a scenario in which both BWH and FAW attacks are possible. Also, the actions at the Nash equilibrium and their resulting payoffs differ markedly from those in classical games such as the prisoner's dilemma.

4.3 ARS (Adaptive Retaliation Strategies)

We now propose strategies that induce cooperation among two pools, i.e., no-attack, which is provably verified in the framework of repeated games. In the classical repeated game theory, it is well-known that "threat of future punishment" induces cooperation. We

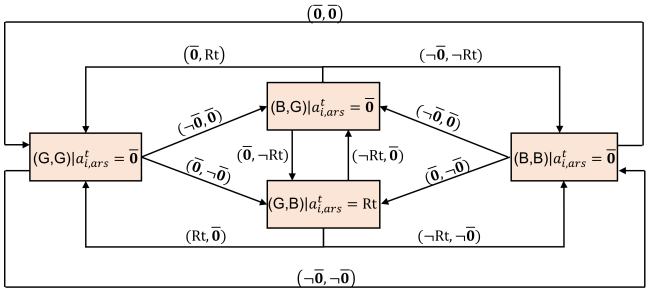


Figure 2: ARS_K has four states, depending on the two pools' standings. In each state box, (G or B,G or B) represents $(\texttt{stnd}_i, \texttt{stnd}_{-i})$, and Pool_i's action $a^t_{i,ars}$ is presented for the given pools' standings according to ARS_K. The tuple at each edge represents (a_i^t, a_{-i}^t). The action tuple at each edge results in a stage change.

inherit such a rationale in our study; however, the following key differences are noted: (i) As mentioned in Section 4.2, the prisoner's dilemma is played repeatedly in many studies, whereas our stage game significantly differs from the prisoner's dilemma, and (ii) our stage game is also defined for a continuous action space, and thus, in punishing other pools deviating from cooperation, it is critically important to adaptively determine the amount of infiltration power for retaliation. As a result, considering the above facts, we should find a *credible retaliation*, which is necessary for inducing cooperation according to Folk Theorem.

4.3.1 Strategy description. In this paper, we denote by (a_i^t, a_{-i}^t) the resulting actions of two pools at stage t. A given strategy of both pools would produce the sequence of actions $(a_i^t, a_{-i}^t)_{t=0}^{\infty}$. We now describe special strategies, named *ARS (Adaptive Retaliation Strategies)*, which call a subroutine **Retaliate** of Algorithm 2. Here **Retaliate** has infinitely many versions, depending on a parameter K that we will describe in **Retaliate subroutine** paragraph. Therefore, we denote by ARS_K a strategy belonging to ARS, and ARS_K is represented in Algorithm 1. When playing ARS_K, an internal variable $\texttt{stnd}_i$ representing the *standing* of Pool_i is maintained by each pool; $\texttt{stnd}_i$ represents whether Pool_i has followed ARS_K well or not at the previous stage. We use the notation $a^t_{i,ars}$ to refer to the action when ARS_K is played, in order to differentiate an action a_i^t from a different strategy. Thus, $a_i^t = a^t_{i,ars}$ when Pool_i plays ARS_K.

ARS_K: In ARS_K, Pool_i starts to cooperate with no-attack when $t = 0$, and initializes its standing variable $\texttt{stnd}_i$ to GOOD. At each stage t, Pool_i first sets its standing $\texttt{stnd}_i$, depending on whether its stage $t-1$ action a_i^{t-1} matches $a^{t-1}_{i,ars}$ from ARS_K (***S1***). Thus, if Pool_i deviates from what ARS_K does at the stage $t-1$, its standing at the stage t is set to BAD. Then, different standing values of both pools lead to the following combinations: $(\texttt{stnd}_i, \texttt{stnd}_{-i}) = (\texttt{G}, \texttt{G})$, $(\texttt{G}, \texttt{B})$, $(\texttt{B}, \texttt{G})$, $(\texttt{B}, \texttt{B})$, where G = 'GOOD' and B = 'BAD'. To help readers better understand ARS_K, we present a state diagram of ARS_K in Fig. 2, where four states exist, depending on two pools' standings; in each state, we also present Pool_i's action $a^t_{i,ars}$ at stage t according to ARS_K. In the figure, Rt is the output of **Retaliate** with K (denoted by $\textbf{Retaliate}_K$ in Algorithm 1), and $\neg(*)$ denotes an action value that differs from $(*)$. The action tuple at each edge (which may deviate from ARS_K) is what results in a state change, and we did

ARS_K for each pool i

Start when $t = 0$:

Start the stage game with no-attack, (i.e., $a_i^0 = \overline{\mathbf{0}}$), and set a variable $\texttt{stnd}_i = \texttt{GOOD}$.

At each stage $t \geq 1$:

S1. *Set the standing of this stage.*
If $(a_i^{t-1} == a_{i,ars}^{t-1})$, $\texttt{stnd}_i = \texttt{GOOD}$, else $\texttt{stnd}_i = \texttt{BAD}$.

S2. *Estimate the infiltration power.*
If $(\texttt{stnd}_i == \texttt{GOOD})$ and $(\texttt{stnd}_{-i} == \texttt{BAD})$
$a_i^t = \textbf{Retaliate}_K\left(\alpha_i, a_i^{t-1}, \alpha_{-i}, a_{-i}^{t-1}, a_{-i,ars}^{t-1}\right)$
else $a_i^t = \overline{\mathbf{0}}$.

S3. Output a_i^t.

Algorithm 1: ARS (Adaptive Retaliation Strategies) for two pools.

not present the action tuples that do not change a state (e.g., in (G,G), the action $(\overline{\mathbf{0}}, \overline{\mathbf{0}})$ does not incur the state change).

To summarize ARS_K, Pool_i starts with cooperation, and then retaliates when the opponent deviates from cooperation. However, if, as a response to Pool_i's retaliation, the opponent goes back to cooperation, Pool_i stops retaliating and resumes cooperation with its opponent. If the opponent is not back to cooperation (thus not following ARS_K) and keeps executing attacks, Pool_i, which follows ARS_K, also keeps retaliating. When two pools deviate from ARS_K simultaneously, both of them turn out to cooperate at the next stage. Retaliation phase is presented in Fig. 2, where Pool_i retaliates against its opponent (***S2***) only when the standing is (G, B) (i.e., when Pool_i follows ARS_K but the opponent deviates from ARS_K). Considering this fact, at least one of a_i^{t-1} and $a_{-i,ars}^{t-1}$, which are two inputs of **Retaliate**, should be $\overline{\mathbf{0}}$ (see the actions at edges toward (G, B)). This is because (i) if $a_i^{t-1} \neq \overline{\mathbf{0}}$, it indicates that the opponent's standing at stage $t-1$ was BAD, and thus Pool_{-i} should not attack according to ARS_K, i.e., $a_{-i,ars}^{t-1}$ is $\overline{\mathbf{0}}$, or (ii) if $a_{-i,ars}^{t-1} \neq \overline{\mathbf{0}}$, a_i^{t-1} should be $\overline{\mathbf{0}}$. After Pool_i's retaliation, the opponent goes back to cooperation as a contrite behavior; the contrite phase is represented as a change from (G,B) to (G,G), where two pools cooperate (Fig. 2). In the (B,B) state where two pools deviate from ARS_K, both of them cooperate at the next stage, making the transition to (G,G).

Note that ARS assumes that Pool_i has values of a_{-i}^{t-1}, $\bar{a}_{-i}^{t-1}$, $\texttt{stnd}_{-i}$, and α_{-i} of its opponent; we will discuss how that information is available to Pool_i in Section 6. Indeed, there exists a strategy, *contrite tit-for-tat* (CTFT) [3], which uses standings similar to (not the same as) that for ARS and induces cooperation in the iterated prisoner's dilemma. However, CTFT is studied as a strategy for the iterated prisoner's dilemma with a discrete action space including only two actions, where ARS significantly differs from CTFT.

Retaliate subroutine: Prior to explaining **Retaliate**, we first introduce the notion of an *infiltration power candidate set* (or simply, *infiltration set*) as follows: for given pools' local and opponent actions a_i^{t-1}, a_{-i}^{t-1}, and $a_{-i,ars}^{t-1}$, we define the infiltration set with respect to either FAW or BWH as the set of Pool_i's infiltration powers that makes Pool_{-i}'s attack unprofitable as a retaliating response to Pool_{-i}'s deviation from cooperation. Formally, Pool_i's

Input: *Local.* computation power α_i, previous action a_i^{t-1}
Input: *Opponent.* computation power α_{-i}, previous action a_{-i}^{t-1}, previous ARS_K action $a_{-i,ars}^{t-1}$
Output: Pool_i's action a_i^t

S.1 Retaliation with FAW

S.1.1 Construct the infiltration set for FAW $IP_{\text{faw}} = IP_{\text{faw}}(a_i^{t-1}, a_{-i}^{t-1}, a_{-i,ars}^{t-1})$. If $(IP_{\text{faw}} == \emptyset)$, goto ***S.2***.

S.1.2 Find the following two sets F_1 and F_2 as follows:

$$F_1 = \min\Big\{f_i \in IP_{\text{faw}} \mid U_i(a_i^{t-1}, a_{-i}^{t-1}) - U_i(a_i^{t-1}, a_{-i,ars}^{t-1}) \geq U_{-i}((f_i, 0), \overline{\mathbf{0}})\Big\},$$
$$F_2 = \underset{f_i \in IP_{\text{faw}}}{\arg\min} |f_i - M_i^F(\alpha_i, \alpha_{-i})|, \tag{9}$$

S.1.3 Compute the infiltration power f_i for retaliating with FAW as $f_i = \min\{F_1 \cup F_2\}$, and set $a_i^t = (f_i, 0)$. Goto ***S.3***.

S.2 Retaliation with BWH

S.2.1 Construct the infiltration set for BWH $IP_{\text{bwh}} = IP_{\text{bwh}}(a_i^{t-1}, a_{-i}^{t-1}, a_{-i,ars}^{t-1})$.

S.2.2 Find the following two sets F_1 and F_2 as follows:

$$B_1 = \min\Big\{b_i \in IP_{\text{bwh}} \mid U_i(a_i^{t-1}, a_{-i}^{t-1}) - U_i(a_i^{t-1}, a_{-i,ars}^{t-1}) \geq U_{-i}((0, b_i), \overline{\mathbf{0}})\Big\},$$
$$B_2 = \underset{b_i \in IP_{\text{bwh}}}{\arg\min} |b_i - M_i^B(\alpha_i, \alpha_{-i})|, \tag{10}$$

S.2.3 Compute the infiltration power b_i for retaliating with BWH as $b_i = \min\{B_1 \cup B_2\}$, and set $a_i^t = (0, b_i)$. Goto ***S.3***.

S.3 Terminate. Output a_i^t.

Algorithm 2: Retaliate Subroutine where $M_i^F(\alpha_i, \alpha_{-i})$ and $M_i^B(\alpha_i, \alpha_{-i})$ are given in (13) and (14), respectively.

infiltration set IP_{faw} for the FAW attack is given as:

$$IP_{\text{faw}}(a_i^{t-1}, a_{-i}^{t-1}, a_{-i,ars}^{t-1}) := \Big\{f_i \mid U_{-i}(a_i^{t-1}, a_{-i}^{t-1}) + K \cdot U_{-i}((f_i, 0), \overline{\mathbf{0}}) < U_{-i}(a_i^{t-1}, a_{-i,ars}^{t-1}), 0 \leq f_i \leq \alpha_i\Big\}, \tag{11}$$

where K is an arbitrary number in $[0, 1)$. As K gets close to 1, the retaliator tries to use FAW attacks as much as possible rather than BWH attacks. Similarly, we define IP_{bwh} for the BWH attack as:

$$IP_{\text{bwh}}(a_i^{t-1}, a_{-i}^{t-1}, a_{-i,ars}^{t-1}) := \Big\{b_i \mid U_{-i}(a_i^{t-1}, a_{-i}^{t-1}) + U_{-i}((0, b_i), \overline{\mathbf{0}}) < U_{-i}(a_i^{t-1}, a_{-i,ars}^{t-1}), 0 \leq b_i \leq \alpha_i\Big\}. \tag{12}$$

The main goal of **Retaliate** is to determine which attack to perform and how much infiltration power is needed to retaliate against the deviating opponent while maximizing the retaliator's (long-term) payoff. Thus, the crux of **Retaliate** is to strike a good balance between retaliation and selfishness. To this end, we first prioritize FAW over BWH, simply because the FAW attack is known to be more profitable than the BWH attack [17] (see ***S.1*** and ***S.2***, where ***S.1*** is first attempt). We henceforth focus on the steps for retaliation with FAW (***S.1***), which is quite similar to that with BWH, where we first construct the FAW-infiltration set IP_{faw} as in (11). In fact, it is possible for IP_{faw} to be empty, and this occurs when the FAW attack has no effect of retaliation, in which case the retaliation with BWH is then tried (***S.2***). Note that the BWH-infiltration set (***S.2.1***) is provably non-empty. Intuitively, this is because BWH is known to

have more strength in damaging the opponent more severely [17]. In the full version [19], we prove the non-emptiness of IP_{bwh}.

Next, in balancing between retaliation and selfishness, we construct two filtered sets of infiltration powers, F_1 and F_2 (***S.1.2***), which consider retaliation and selfishness, respectively. In F_1, Pool_i following ARS_K computes the set of infiltration powers in proportion to the degree of Pool_{-i}'s attack, i.e., generating the same amount of loss to Pool_{-i} as that to Pool_i from Pool_{-i}'s attack, which we call "equal retaliation". In F_2, the set of infiltration powers is constructed so as to maximize Pool_i's payoff for the FAW attack, where $f_i \in F_2$ is chosen to be closest to the infiltration power $M_i^F(\alpha_1, \alpha_2)$ maximizing Pool_i's payoff U_i, expressed as:

$$M_i^F(\alpha_1, \alpha_2) = \frac{-1+\sqrt{1-\alpha_i(1+\alpha_{-i})-(1-\alpha_i-\alpha_{-i})(1+\alpha_{-i}-\alpha_i\alpha_{-i})}}{\alpha_i(1-\alpha_i-\alpha_{-i})}, \quad (13)$$

which is obtained from [17]. Finally, in ***S.1.3***, Pool_i decides to retaliate by deciding between equal retaliation (F_1) and selfishness (F_2). **Retaliate** chooses the minimum infiltration power for FAW in F_1 and F_2, which is the minimum amount of power to achieve retaliation while considering its own payoff. Therefore, if Pool_i must use a significant portion of its computational power to infiltrate for equal retaliation, it instead maximizes its payoff rather than pursuing equal retaliation. Sets B_1 and B_2, which are similar to F_1 and F_2 for the FAW attack, are constructed for the BWH attack, where $M_i^B(\alpha_1, \alpha_2)$ is derived from [9], given as:

$$M_i^B(\alpha_1, \alpha_2) = \frac{-\alpha_{-i}(1-\alpha_i)+\sqrt{-\alpha_{-i}^2(-1+\alpha_i+\alpha_i\alpha_{-i})}}{1-\alpha_i-\alpha_{-i}} \quad (14)$$

Note that **Retaliate** outputs $\overline{\mathbf{0}}$ if $0 \in F_1$, in which case Pool_i does not need to retaliate. This occurs when the opponent did not attack at stage $t-1$, even though the opponent would retaliate against Pool_i at stage $t-1$ according to ARS_K. In this case, because the opponent did not follow ARS_K, the opponent's standing would be BAD, where Pool_i would call **Retaliate**. However, **Retaliate** would usually output $\overline{\mathbf{0}}$ in this case, and Pool_i would not attack for retaliation because IP_{faw} and F_1 would include 0 in most cases.

4.3.2 Equilibrium Analysis. Next, we prove that ARS is a subgame perfect Nash equilibrium for a sufficiently large δ.

Theorem 4.2. *There exists a function $F_K(\alpha_1, \alpha_2)$ such that, for all discount factor $\delta \geq F_K(\alpha_1, \alpha_2)$, the two-pool strategy vector (ARS_K, ARS_K) is a subgame perfect Nash equilibrium. Function $F_K(\alpha_1, \alpha_2)$ is always less than 1, and $F_K(\alpha_1, \alpha_2)$ is an increasing function of K and $|\alpha_1 - \alpha_2|$ for given α_1. Moreover, (ARS_K, $\text{ARS}_{K'}$) is a Nash equilibrium for all $\delta \geq F_{\max(K,K')}(\alpha_1, \alpha_2)$.*

A proof of Theorem 4.2 appears in the full version [19]. In the proof of Theorem 4.2, we show that it is not more profitable for each player to deviate ARS at the start of any subgame when compared to the case where it follows ARS. This implies that ARS is a subgame perfect Nash equilibrium according to *one-time deviation property*.

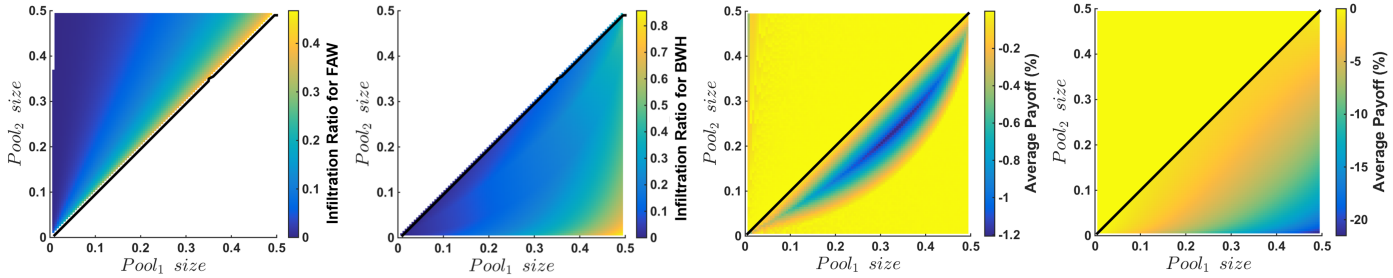
If two pools use one of ARS (their strategies need to be not necessarily the same), the strategy vector is a Nash equilibrium. Especially, if two pools use the same strategy, the strategy vector is a subgame perfect Nash equilibrium. As described in Section 3.2, a subgame perfect Nash equilibrium refines a Nash equilibrium by eliminating *non-credible threats*, which is a strategy vector that rational pools are actually unlikely to follow. In addition, a large value of δ implies a condition in which pools consider future payoffs significantly, or the probability that pools are patient enough to stay inside the system for a long time. Indeed, most pools, including Slush, Eligius and F2Pool, are operated for a long time in practice. A large value of δ is also better satisfied when the duration of one stage is short compared to the pools' entire operation period. In Section 6, we explain that the duration of a stage period can be short, which supports the practical value of our analytical result.

Indeed, there are many other subgame perfect Nash equilibria (from Folk Theorem in repeated games [12]) in the repeated FAW-BWH game, which trivially include the one that two pools always execute FAW attacks against each other. From a manager's perspective, the manager would want to increase its pool size while earning extra rewards, until the pool increases to a size that does not seriously threaten the system. This is a good reason for the manager to execute the FAW attack. Meanwhile, it is unknown whether the subgame perfect Nash equilibria have cooperation between pools because the existence of *credible retaliation* in the repeated FAW-BWH game has not been studied to the best of our knowledge. Our results imply that cooperation can be stable when ARS is used even though the FAW-BWH game has a certain winner; i.e., a larger pool. Moreover, ARS includes infinitely many strategies with K in (11). As such, there are infinitely many ways to achieve cooperation between pools. In addition, ARS restores cooperation even if FAW and BWH attacks impulsively occur, which is another advantage of ARS. For example, if Pool_i impulsively attacks, the opponent would retaliate. After that, Pool_i does not attack, being contrite, and two pools achieve the no-attack status.

5 NUMERICAL ANALYSIS

In this section, we use a numerical analysis to demonstrate how much infiltration power each pool would use for retaliation in ARS_K in response to the opponent's action. We simulate the repeated FAW-BWH game with varying Pool_1 and Pool_2's sizes. We consider a case in which Pool_1 deviates from ARS_K to attack Pool_2 while maximizing its payoff U_1 during one stage. As a result, Pool_2 would retaliate against Pool_1 according to ARS_K. In this section, we consider a strategy ARS_{1^-}, where K is close to 1.

Fig. 3 represents when Pool_1 optimally executes the FAW attack to maximize its payoff. Then, Pool_2 following ARS_{1^-} retaliates at the next stage. The x and y-axes are Pool_1 and Pool_2's sizes, respectively. Moreover, we define infiltration ratios $r_i = (r_i^F, r_i^B)$, where r_i^F and r_i^B are proportions of infiltration power f_i and b_i for Pool_i's computational power, respectively (i.e., $r_i^F = \frac{f_i}{\alpha_i}$, $r_i^B = \frac{b_i}{\alpha_i}$). Fig. 3a represents Pool_2's infiltration ratio r_2^F for retaliation using the FAW attack. In the white region of Fig. 3a, Pool_2 cannot retaliate against Pool_1 with the FAW attack. Thus, Pool_2 should retaliate using the BWH attack. Fig. 3b represents Pool_2's infiltration ratio r_2^B for retaliation using the BWH attack. Here, we can see that all cases are covered with the colored regions in Fig. 3a and 3b. Considering two stages where Pool_1 first executes the FAW attack and then Pool_2 retaliates, Fig. 3c and 3d represent average payoffs of Pool_1 and Pool_2 for two stages, respectively. That is, these figures show $\frac{U_i(a^0)+U_i(a^1)}{2}$ when we denote each of two stages by stage 0 and 1. As shown in Fig. 3c, Pool_1's average payoff is always negative,



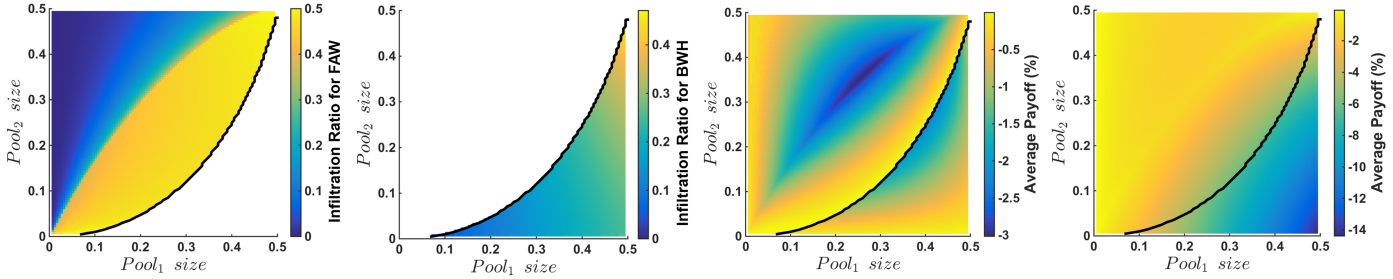
(a) $Pool_2$'s infiltration ratio for retaliation using the FAW attack.

(b) $Pool_2$'s infiltration ratio for retaliation using the BWH attack.

(c) Average relative extra reward (%) of $Pool_1$ for two stages.

(d) Average relative extra reward (%) of $Pool_2$ for two stages.

Figure 3: $Pool_1$ optimally executes the FAW attack.



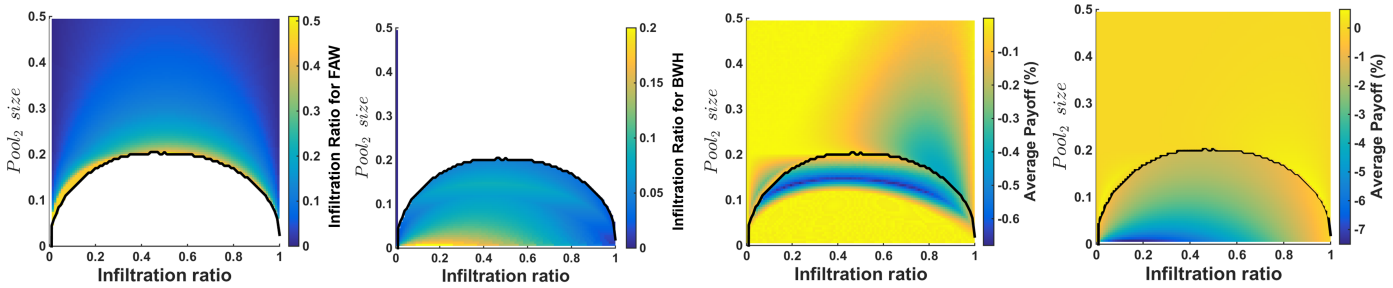
(a) $Pool_2$'s infiltration ratio for retaliation using the FAW attack.

(b) $Pool_2$'s infiltration ratio for retaliation using the BWH attack.

(c) Average relative extra reward (%) of $Pool_1$ for two stages.

(d) Average relative extra reward (%) of $Pool_2$ for two stages.

Figure 4: $Pool_1$ optimally executes the BWH attack.



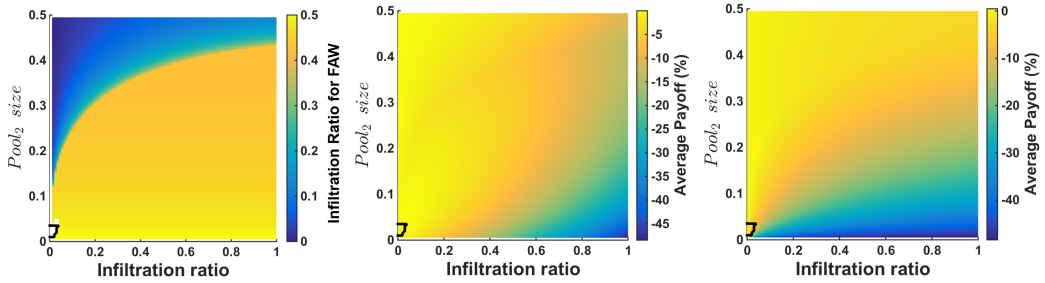
(a) $Pool_2$'s infiltration ratio for retaliation with the FAW attack against $Pool_1$'s FAW attack.

(b) $Pool_2$'s infiltration ratio for retaliation with the BWH attack against $Pool_1$'s FAW attack.

(c) Average relative extra reward (%) of $Pool_1$ for the two stages when $Pool_1$ executes the FAW attack.

(d) Average relative extra reward (%) of $Pool_2$ for the two stages when $Pool_1$ executes the FAW attack.

Figure 5: $Pool_1$ with computational power of 20% executes the FAW attack.



(a) $Pool_2$'s infiltration ratio for retaliation using the FAW attack against $Pool_1$'s BWH attack.

(b) Average relative extra reward (%) of $Pool_1$ for the two stages when $Pool_1$ executes the BWH attack.

(c) Average relative extra reward (%) of $Pool_2$ for the two stages when $Pool_1$ executes the BWH attack.

Figure 6: $Pool_1$ with computational power of 20% executes the BWH attack.

meaning that ARS_{1^-} makes FAW attacks unprofitable. Moreover, Fig. 3d shows that $Pool_2$ can completely recover a loss from $Pool_1$'s attack in the case where $Pool_2$ retaliates with the FAW attack.

Fig. 4 represents when $Pool_1$ optimally executes the BWH attack to maximize its short-term payoff. Similar to Fig. 3, Fig. 4a and 4b represent $Pool_2$'s infiltration ratio r_2^F and r_2^B, respectively. Fig. 4c and 4d respectively represent the average payoffs of $Pool_1$ and $Pool_2$ for two stages. Fig. 4c shows that $Pool_1$ always suffers a loss from the retaliation of $Pool_2$ when $Pool_1$ executes the BWH attack. Therefore, it shows that ARS makes BWH attacks unprofitable.

As a representative scenario, we simulate the repeated FAW-BWH game in terms of various $Pool_1$'s infiltration ratio used for attacks, assuming that $Pool_1$'s size is 0.2 (20%). Fig. 5 and 6 represent $Pool_1$'s execution of FAW and BWH attacks, respectively. The x and y-axes are $Pool_1$'s infiltration ratio used for attack and $Pool_2$'s sizes, respectively. Fig. 5a and 5b show the infiltration ratio r_2^F and r_2^B for retaliation against $Pool_1$, respectively. Because the extent of retaliation by ARS depends on the loss caused by the opponent's attack, $Pool_2$'s infiltration ratio for retaliation depends on $Pool_1$'s attack infiltration ratio. Fig. 5c and 5d represent the average payoffs of $Pool_1$ and $Pool_2$, respectively, for two stages in which $Pool_1$ executes the FAW attack and then $Pool_2$ retaliates. $Pool_1$ always suffers a loss by deviating from ARS because all colors in Fig. 5c indicate negative values. Meanwhile, there are some cases in which $Pool_2$ can earn extra profit in the process of retaliation, as shown in Fig. 5d. Similar to Fig. 5, Fig. 6 shows $Pool_2$'s infiltration ratio r_2^F for retaliation, and the attacker's and victim's average payoffs for two stages when $Pool_1$ executes the BWH attack. In most cases, $Pool_2$ chooses the FAW attack rather than the BWH attack for retaliation. Even though there exist some cases to execute the BWH attack in response to ARS_{1^-}, we omit $Pool_2$'s infiltration ratio r_2^B for retaliation with the BWH attack because the region of such cases is very small (see small areas bounded by black bold lines at left-bottom corners in Fig. 6). As a result, BWH attacks become unprofitable by ARS.

Table 1: Considering the current power distribution [2] and assuming that BTC.com is an attacker, this table lists infiltration ratios $r_i = (r_i^F, r_i^B)$ that four pools use for retaliation according to ARS_{1^-}.

Name	(r_i^F, r_i^B)(%) against FAW	Total Payoff	(r_i^F, r_i^B)(%) against BWH	Total Payoff
AntPool	(0, 14.33%)	-1.89%	(46.2%, 0)	-0.78%
ViaBTC	(0, 13.7%)	-0.54%	(47.2%, 0)	-0.15%
DPool	(0, 17.71%)	-0.004%	(0, 13.14%)	-1.1%
Bixin	(0, 21%)	-0.025%	(0, 13%)	-0.63%

Also, we consider the current power distribution obtained from Blockchain.info [2]. We assume that BTC.com, which is the largest pool as of Jan 2019 and has a computational power of about 25%, optimally executes FAW and BWH attacks against each of four pools (AntPool, ViaBTC, DPool, and Bixin), which have respective computational powers of 15%, 10%, 3.5%, and 2%. In this case, four pools would retaliate against BTC.com according to ARS. Table 1 represents the infiltration ratio $r_i = (r_i^F, r_i^B)$, which $Pool_i$ uses for retaliation with FAW and BWH against BTC.com, respectively. The second and fourth columns show how each pool should retaliate against BTC.com's FAW and BWH attacks, respectively. The third and fifth columns represent the attacker's total payoff for each victim pool when the attacker executes FAW and BWH attacks, respectively. As shown in Table 1, by retaliating according to ARS_{1^-}, the four pools make the attacks of BTC.com unprofitable.

6 IDENTIFYING THE OPPONENT'S ATTACK

To follow ARS, $Pool_i$ needs to know seven parameters in Table 2: α_i, $\texttt{stnd}_\texttt{i}$, a_i^{t-1}, α_{-i}, $\texttt{stnd}_{-\texttt{i}}$, a_{-i}^{t-1}, and $a_{-i,ars}^{t-1}$. In this section, we describe how the pool can obtain these seven parameters, which make it possible for pools to adopt ARS. Among these parameters, $Pool_i$ already knows α_i, $\texttt{stnd}_\texttt{i}$, and a_i^{t-1}, which are referred to as *internal variables* in this paper. Also, $Pool_i$ can easily obtain α_{-i} because the computational power of pools can be approximately calculated from the mined block information [2]. Among the remaining parameters, $\texttt{stnd}_{-\texttt{i}}$, a_{-i}^{t-1}, and $a_{-i,ars}^{t-1}$, $Pool_i$ needs to know a_{-i}^{t-1} and $a_{-i,ars}^{t-1}$ because $\texttt{stnd}_{-\texttt{i}}$ is determined by a_{-i}^{t-1} and $a_{-i,ars}^{t-1}$. Moreover, the value $a_{-i,ars}^{t-1}$ is $\overline{\mathbf{0}}$ if $t-1$ is 0. If $t-1$ is positive, the value can be obtained from pools' actions at stage $t-2$. In other words, $Pool_i$ can determine $a_{-i,ars}^{t-1}$ by obtaining the opponent's action a_{-i}^{t-2} at stage $t-2$. As a result, $Pool_i$ only needs to know the opponent's previous action in order to determine $\texttt{stnd}_{-\texttt{i}}$, a_{-i}^{t-1}, and $a_{-i,ars}^{t-1}$.

To guess the opponents' actions, $Pool_i$ can plant moles in other pools. Through the moles, $Pool_i$ can observe other pools' average reward densities and stochastically determine other pools' actions from their observed average reward densities. However, it may take a long time to find out other pools' actions with their average reward densities. Note that, if the time duration of a stage increases, the discount factor δ would be decreased because pools might focus on the increase of short-term advantages rather than long-term value. This implies that it is important to shorten the time duration of a stage. In the following section, we describe how to achieve this.

Table 2: List of parameters.

Notation	Definition
α_i	Computational power of $Pool_i$
$\texttt{stnd}_\texttt{i}$	Standing of $Pool_i$
a_i^{t-1}	The action of $Pool_i$ at time $t-1$
α_{-i}	Computational power of the opponent
$\texttt{stnd}_{-\texttt{i}}$	Standing of the opponent
a_{-i}^{t-1}	The action of the opponent at time $t-1$
$a_{-i,ars}^{t-1}$	Output of ARS of the opponent at time $t-1$

Detection of Attacks: First, $Pool_i$ must determine whether it is being attacked. Indeed, $Pool_i$ can easily detect FAW attacks because the fork rate increases [17]. Moreover, the manager of victim pool can detect BWH attacks by investigating the ratio between the number of submitted shares and the number of found blocks [9, 22].

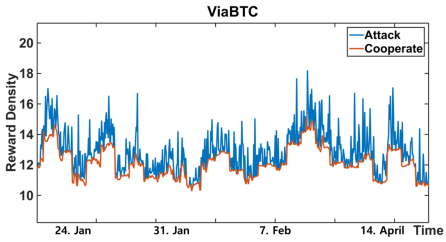
Furthermore, we found that the period of attack detection can be reduced if the manager can focus only on the shares submitted by *unlucky miners*[3] and *short-term miners*[4], rather than all shares. For example, if a victim pool's size is 20% and a BWH attacking pool infiltrates 0.5% into the pool, the victim pool would find 20.1%

[3]Miners who submitted a relatively small number of FPoWs
[4]Miners who stayed in the pool for relatively short period of time

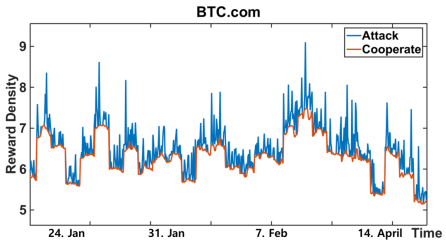
(0.2/0.995) of all blocks on average. Note that the victim pool would find 20.5% of all blocks on average in the case that the infiltrating power of 0.5% belongs to benign miners. When the manager of the victim pool considers entire 2000 blocks (about two weeks in Bitcoin), there is an approximately 35.82% probability that the ratio of the victim pool's found blocks to all blocks is 0.201, under the assumption that an attack had not occurred. Therefore, it is difficult for the manager to determine whether an attack has occurred. However, if the manager considers the shares submitted by only unlucky and short-term miners, he would be able to find these shares that account for about 0.5% of the entire computational power. Assuming that the attack had not occurred, the probability that these shares do not contain FPoWs is about 0.0045%. Therefore, the manager can successfully detect *attacks* and estimate the *infiltration power* used for attacks. If a pool does not detect attacks, other pools' actions are set as $\overline{\mathbf{0}}$. Otherwise, the pool needs to identify the attacking pool.

Identification of Attackers: The FAW or BWH attacker receives a portion of the reward earned by the victim pool. Therefore, the attacker's reward is related to the number of blocks generated from the victim pool. We denote by P the period in which the attacker finds one block in its pool. At the end of P, miners in the attacker's pool receive a portion of the reward for the one block and the rewards gained from the victim pool for period P. For ease of presentation, we use α and β to represent the attacker's size and the victim's size, respectively, instead of α_{-i} and α_i. We also suppose the attacker infiltrates a fraction γ of the attacker's computational power into the victim pool. Moreover, we denote by Rd_p the earned reward density of miners belonging to the attacking pool for the period P. If the number of generated blocks from the victim pool for the period P is N, Rd_p would be $\frac{1}{\alpha} + \frac{N\gamma}{\beta+\gamma\alpha}$. We can easily check that N has a geometric distribution with a parameter $\frac{(1-\gamma)\alpha}{\beta+\gamma\alpha(1-\alpha-\beta)+(1-\gamma)\alpha}$ for FAW attack, and $\frac{(1-\gamma)\alpha}{\beta+(1-\gamma)\alpha}$ for BWH attack (see the full version [19] for details).

If α does not change and the pool does not attack, the reward density Rd_p for the period P is fixed at $\frac{1}{\alpha}$. Meanwhile, if the pool attacks, Rd_p would continuously change depending on N. Therefore, the victim can identify the attacking pool immediately by observing their variances in Rd_p after planting moles in pools. However, even if attacks do not occur, the reward density as well as α and β changes in practice. If α usually has a large variance, it would indeed be difficult to identify the attacker by investigating the variance in the reward density. Thus, to determine this, we should find out how much is the variance in pools' computational powers in real world when attacks do not occur. To this end, we collected hash rates from two pools (ViaBTC [31] and BTC.com [4]) by monitoring their hash rates over one month (Jan. 21, 2019~Feb. 18, 2019). Two pools publicly provide their average hash rate (PH/s) for one hour. Using the data, we first normalized their computational power and then calculated the reward density when assuming that these pools are benign, as a reciprocal number of computational power (e.g., if a pool's computational power is 0.2, its reward density Rd_p would be 5). The reward density at time t indicates how much of the reward per computational power a pool miner can earn when the corresponding pool finds one block at time t. Note that the value of



(a) ViaBTC's reward densities



(b) BTC.com's reward densities

Figure 7: Reward densities of two pools. In each figure, the blue and orange lines represent each pool's reward density when the pool executes FAW attacks and when the pool cooperates, respectively.

$\frac{Rd_p}{E[P]}$ for each miner is the same regardless of pool when all miners are honest, where $E[P]$ is the mean of P.

We simulated two scenarios with the FAW attack: (1) first scenario (attack pool: ViaBTC) and (2) second scenario (attack pool: BTC.com) in which the attack pool executes the FAW attack with an infiltration power of 0.005 (0.5%) against the victim pool with a computational power 0.2 (20%). Fig. 7 shows the reward densities Rd_p of two pools. Blue and orange lines represent reward densities observed when each pool executes the FAW attack and when they are benign, respectively. In Fig. 7, blue lines fluctuate noticeably more than the orange lines. By running the simulation several times, we observed that the variation in the blue lines was usually about 99~152 and 24~32 times the variation in the orange lines for ViaBTC and BTC.com, respectively.

We also simulated two scenarios with the BHW attack: (1) first scenario (attack pool: ViaBTC) and (2) second scenario (attack pool: BTC.com) in which the attack pool executes the BWH attack with an infiltration power of 0.005 against the victim pool with a computational power 0.2. Similarly, in two cases, the variance in reward densities of the attacker when the attack is executed increases about 85~177 and 22~36 times that for when the attack is not executed, respectively. For space reasons, we omit figures representing the simulation results similar to Fig. 7. *These results show that if a pool is a FAW or BWH attacker, the pool's reward density Rd_p would usually increase (or decrease) when N increases (or decreases).* Therefore, the victim can identify the attacker by observing the variance in reward densities in other pools and by comparing it with the number of blocks found in his pool. Probably, the attacker would try several methods to reduce the variance in the attacker's reward density to hide the attack evidence. However, we note that the victim can still observe relatively high variance in the attacker's reward density even when such evasion methods are used (see the full version [19]).

ARS_K for each pool i against pool $j \neq i$

Start when $t = 0$**:**

Start the stage game with no attack, (i.e., $a_{ij}^0 = \overline{\mathbf{0}}$), and set a variable $\texttt{stnd}_{\texttt{ij}} = \texttt{GOOD}$.

At each stage $t \geq 1$:

S1. *Set the standing of this stage.*
If $(a_{ij}^{t-1} == a_{ij,ars}^{t-1})$, $\texttt{stnd}_{\texttt{ij}} = \texttt{GOOD}$, else $\texttt{stnd}_{\texttt{ij}} = \texttt{BAD}$.

S2. *Estimate the infiltration power.*
If $(\texttt{stnd}_{\texttt{ij}} == \texttt{GOOD})$ and $(\texttt{stnd}_{\texttt{ji}} == \texttt{BAD})$
$a_{ij}^t = \textbf{Retaliate}_K \left(\alpha_i, a_{ij}^{t-1}, \alpha_j, a_{ji}^{t-1}, a_{ji,ars}^{t-1}\right)$
else $a_{ij}^t = \overline{\mathbf{0}}$.

S3. Output a_{ij}^t.

Algorithm 3: ARS for multiple pools.

Summary: In summary, the victim first determines whether attacks occurred by investigating the fork rate and the ratio of found blocks to submitted shares. This method also allows the victim to estimate the infiltration power in the victim pool. The victim then identifies the attacker by investigating the variance in Rd_p in other pools. Using the above two methods, a pool can identify other pools' actions. Therefore, a pool can find out all seven parameters in Table 2 to run ARS.

7 MULTIPLE POOLS

In this section, we present ARS for n pools (Pool_i: $i = 1 \sim n$). To this end, we now specify, for all notations of Pool_i's standing and action, those against Pool_j for each $j \neq i$, such as $\texttt{stnd}_{ij}$, a_{ij}^t, and $a_{ij,ars}^t$. Then, Pool_i maintains $n-1$ dimensional standing vectors $(\texttt{stnd}_{ij})_{j\neq i}$ and action vectors $(a_{ij}^t)_{j\neq i}$, $(a_{ij,ars}^t)_{j\neq i}$. Note that $a_{ij}^t = (f_{ij}^t, b_{ij}^t)$, where f_{ij}^t and b_{ij}^t are the infiltration powers for FAW and BWH attacks, respectively. ARS_K of Pool_i against Pool_j is described in Algorithm 3 (similar to Algorithm 1). When Pool_i follows ARS_K, the pool retaliates against Pool_j only if $\texttt{stnd}_{ij}$ and $\texttt{stnd}_{\texttt{ji}}$ are GOOD and BAD, respectively, where **Retaliate** in Algorithm 3 is similar to that in Algorithm 2. The difference between **Retaliate** for n and two pools is that n-pool **Retaliate** simply replaces f_i and b_i in Algorithm 1 with f_{ij} and b_{ij}, and outputs a_{ij}^t.

For the identification of attackers, a mechanism similar to that described in Section 6 can be applied. Even if multiple attackers execute attacks against multiple victims in parallel, each victim can find out who the attacker is because the corresponding attacker's reward density fluctuates depending on *the number of blocks found by the corresponding victim.* In addition, when multiple pools target a victim, the victim can identify those attackers and estimate their infiltration rates because each attacker's reward density variance depends on its infiltration rate. More specifically, the larger infiltration rate is, the higher variance in the attacker's reward density is. To distinguish FAW from BWH, the victim can investigate a fork rate. Note that, because the FAW attack is to intentionally generate a fork with blocks generated by a victim and another miner, the fork rate from the FAW attack pool is relatively low compared with those of other pools.

It seems natural to expect that, similar to ARS for two pools, ARS for n pools will make attacks unprofitable and thus induce cooperation, despite the mathematical challenges in formally proving it due to complex inter-coupling among n pools. We numerically conduct this analysis through the simulation of the scenario in which BTC.com (Pool_1) possessing 25% computational power optimally executes FAW or BWH attacks against four other pools (AntPool (Pool_2), ViaBTC (Pool_3), DPool (Pool_4), and Bixin (Pool_5)) at the same time while maximizing its short-term payoff, and four other pools follow ARS_1- (see Table 3). In Table 3, we denote the infiltration ratios for FAW and BWH attacks of Pool_i against Pool_j by $r_{ij} = (r_{ij}^F, r_{ij}^B)$. The third and sixth columns show the optimal infiltration ratios r_{1j}^F and r_{1j}^B ($j = 2 \sim 5$) of BTC.com for FAW and BWH attacks, respectively; these ratios maximize the payoff of BTC.com. The fourth and seventh columns show each pool's infiltration ratio for retaliation against BTC.com's FAW and BWH attacks, respectively. Finally, the fifth (or eighth) column represents BTC.com's payoff for two stages in which BTC.com first executes the FAW (or BWH) attack and then four pools retaliate against BTC.com. For example, when BTC.com executes FAW attacks using infiltration ratios of 22.7%, 15.1%, 5.3%, and 3% against AntPool, ViaBTC, DPool, and Bixin, respectively, the four pools would retaliate against BTC.com at the next stage according to the fourth column. In this case, BTC.com suffers a loss of 5.4 % for two stages in the aggregate. As such, BTC.com's total payoff becomes negative when the pool executes FAW and BWH attacks. Considering these results, it becomes unprofitable to attack, and rational pools sustain cooperation without attacks when they follow ARS.

8 DISCUSSION

8.1 Closed Pools and Solo Miners

Even though we focused on attacks executed only by open pools in this paper, solo miners or closed pools can also execute FAW and BWH attacks in practice. If a solo miner or closed pool is an attacker, the victim cannot retaliate against the attacker because the victim cannot infiltrate its moles. This fact may lead to a rational solo miner or closed pool to execute FAW and/or BWH attacks. However, fortunately, it is widely known that solo miners and closed pools have limited computational power. To estimate the current size of solo miners or closed pools, we observed the hashrate distribution in websites given from BTC.com [26] at the time of writing (Jan. 2019), and we found the total of 22 pools. Among them, BitFury and 58COIN are only closed pools with about 3.1% and 1.3% computational powers, respectively. If each of them executes FAW attacks, separately, against BTC.com, which has about 25% computational power as the largest mining pool, BitFury and 58COIN earn extra reward densities of 0.74%, 0.32%, respectively. Moreover, BTC.com suffers losses of 0.09% and 0.016% from each attack. However, as we can see, the impacts of those attacks seem rather marginal.

8.2 Infiltration Power

Pool_i's infiltration power into other pools should be loyal to Pool_i [9, 17]. The loyal power can be the manager's own computational power or cloud mining [6], or the computational power of miners with a private relation to the manager. Moreover, pools' loyal power

Table 3: The simulation results of ARS for multiple pools. The third and sixth columns show the optimal infiltration ratio for BTC.com's FAW and BWH attacks, respectively. The fourth and seventh columns show each pool's infiltration ratio for retaliation against BTC.com's FAW and BWH attacks, respectively. Finally, the fifth (or eighth) column represents BTC.com's payoff for two stages in which BTC.com first executes the FAW (or BWH) attack and then four pools retaliate against BTC.com.

Name	Computational Power	r_{1j}^F	(r_{j1}^F, r_{j1}^B) against FAW	$\mathcal{U}_1$	r_{1j}^B	(r_{j1}^F, r_{j1}^B) against BWH	$\mathcal{U}_1$
AntPool	15%	22.7%	(0, 13.67%)	−5.4%	9.5%	(46.2%, 0)	−1.55%
ViaBTC	10%	15.1%	(0, 13.7%)		6.4%	(47.2%, 0)	
DPool	3.5%	5.3%	(0, 13.14%)		2.2%	(0, 13.14%)	
Bixin	2%	3%	(0, 13%)		1.3%	(0, 13%)	

ratios are trade secrets. In Section 5, there exist some cases in which the infiltration ratio for retaliation against the attacker is greater than 80%. However, these are extreme cases in which the victim pool's size is very small and the attacker's size is close to 50%. In the current computational power distribution, the infiltration power for retaliation is less than 50%, as shown in Table 1. However, it is possible that a pool has a loyal power of less than 50%. For example, if AntPool has a loyal power of less than 46.2%, the pool cannot prevent the BWH attack of BTC.com through retaliation with the FAW attack. Therefore, to retaliate against BTC.com, Antpool can execute the BWH attack instead of the FAW attack. In this case, Antpool needs only about 14.33% infiltration ratio to prevent the BWH attack through retaliation with the BWH attack of its own.

8.3 Sabotage Attack

We showed that rational pools can cooperate by making attacks unprofitable through ARS. However, FAW and BWH attacks can still be executed by a large pool to disable a small pool (called *Sabotage* attack) because a small pool's loss is greater than a large pool's loss when the large pool deviates from ARS. This situation is commonly known as the *Chicken Game*. If the small pool stops the operation, the large pool would be the winner despite of the loss in their reward because the small pool's miners may migrate into the large pool. However, if two pools among several pools launch attacks against each other, other pools not involved in the attack do not suffer losses stemming from the attack. Therefore, even if the small pool ceases the mining operation, the small pool's miners would move into other pools rather than the large pool. As a result, the large pool cannot earn a direct profit through sabotage attacks.

9 RELATED WORK

Game theory has been used for analyzing attacks and protocols in Bitcoin. Kroll et al. [16] analyzed the economics of Bitcoin mining under the assumption that all miners are rational. They showed that there is a Nash equilibrium in which all miners comply with the Bitcoin protocol when considering a 51% attack. Several studies [14, 20] modeled a game in which two pools decide whether to trigger a DDoS attack against an opponent. Lewenberg et al. [21] considered miners' interactions among mining pools as a cooperative game. They found that some miners would always switch among pools for their profit if the communication delay in the network is large. Luu et al. [22] modeled a power splitting game to analyze how an attacker can optimally execute the BWH attack against multiple pools. Moreover, Eyal [9] showed that a BWH game between two pools results in the prisoner's dilemma. Carlsten et al. [5] and Tsabary et al. [30] analyzed a game among miners when miners earn only transaction fees as block rewards in the future. Kwon et al. [17] proposed the FAW attack and analyzed the FAW game between two pools in which can break the prisoner's dilemma. Yoo et al. [32] studied an incentive design in proof-of-work blockchains, considering a cooperative and non-cooperative strategy of miners. Kwon et al. [18] analyzed a rational behavior of miners when two coins with a compatible proof-of-work mechanism exist. To the best of our knowledge, this paper is the first attempt to consider and analyze both FAW and BWH attacks together in a repeated game. Moreover, we propose infinitely many strategies inducing cooperation among rational pools under the presence of both FAW and BWH attacks.

10 CONCLUSION

In this paper, by modeling a repeated game, we studied how pools can cooperate to avoid the health of systems being weakened from the attacks. Because the stage game, FAW-BWH game, highly differs from the prisoner's dilemma, it may be challenging to find a strategy inducing cooperation among pools. To solve this challenging problem, we proposed novel infinitely many strategies, called ARS, which are likely to be adopted by rational pools. In ARS, a pool first cooperates and then retaliates against the attacker in the case when attacks occur. ARS provably strikes a good balance between retaliation and selfishness. Moreover, there are several parameters required to use ARS in practice. Thus, we discuss the methods to determine the parameters, investigating the real-world data collected from mining pools. As a result, ARS makes cooperation among pools stable, sustainable, and recoverable.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2016-0-00160, Versatile Network System Architecture for Multi-dimensional Diversity); This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (No. 2016R1A2A2A05921755).

REFERENCES

[1] Robert Axelrod et al. 1987. The evolution of strategies in the iterated prisonerâĂŹs dilemma. *The dynamics of norms* (1987), 1–16.
[2] Blockchain Info 2018. Hashrate Distribution. https://blockchain.info/pools. (2018). [Online; accessed 02-May-2018].
[3] Robert Boyd. 1989. Mistakes allow evolutionary stability in the repeated prisoner's dilemma game. *Journal of theoretical Biology* 136, 1 (1989), 47–56.
[4] BTC.com 2018. BTC.com. https://pool.btc.com/pool-stats. (2018). [Online; accessed 30-April-2018].

[5] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. 2016. On the Instability of Bitcoin without the Block Reward. In *Conference on Computer and Communications Security*. ACM.
[6] Cloud Mining 2018. bestcloudmining. http://www.bestcloudmining.net/. (2018). [Online; accessed 2-May-2018].
[7] Nicolas T Courtois and Lear Bahack. 2014. On Subversive Miner Strategies and Block Withholding Attack in Bitcoin Digital Currency. *arXiv preprint arXiv:1402.1718* (2014).
[8] Philip Daian, Ittay Eyal, Ari Juels, and Emin Gün Sirer. 2017. (Short Paper) PieceWork: Generalized Outsourcing Control for Proofs of Work. In *International Conference on Financial Cryptography and Data Security*. Springer, 182–190.
[9] Ittay Eyal. 2015. The Miner's Dilemma. In *Symposium on Security and Privacy*. IEEE.
[10] Ittay Eyal and Emin Gün Sirer. 2014. How to Disincentivize Large Bitcoin Mining Pools. (2014). [Online; accessed 1-May-2017].
[11] Ittay Eyal and Emin Gün Sirer. 2014. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *International Conference on Financial Cryptography and Data Security*. Springer.
[12] Drew Fudenberg and Eric Maskin. 2009. The folk theorem in repeated games with discounting or with incomplete information. In *A Long-Run Collaboration On Long-Run Games*. World Scientific, 209–230.
[13] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *Conference on Computer and Communications Security*. ACM.
[14] Benjamin Johnson, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. 2014. Game-theoretic analysis of DDoS attacks against Bitcoin mining pools. In *International Conference on Financial Cryptography and Data Security*. Springer, 72–86.
[15] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending Fast Payments in Bitcoin. In *Conference on Computer and Communications Security*. ACM.
[16] Joshua A Kroll, Ian C Davey, and Edward W Felten. 2013. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Proceedings of WEIS*, Vol. 2013.
[17] Yujin Kwon, Dohyun Kim, Yunmok Son, Eugene Vasserman, and Yongdae Kim. 2017. Be Selfish and Avoid Dilemmas: Fork After Withholding (FAW) Attacks on Bitcoin. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 195–209.
[18] Yujin Kwon, Hyoungshick Kim, Jinwoo Shin, and Yongdae Kim. 2019. Bitcoin vs. Bitcoin Cash: Coexistence or Downfall of Bitcoin Cash? *arXiv preprint arXiv:1902.11064* (2019).
[19] Yujin Kwon, Hyoungshick Kim, Yung Yi, and Yongdae Kim. 2019. An Eye for an Eye: Economics of Retaliation in Mining Pools. *arXiv preprint arXiv:1908.10781* (2019).
[20] Aron Laszka, Benjamin Johnson, and Jens Grossklags. 2015. When bitcoin mining pools run dry. In *International Conference on Financial Cryptography and Data Security*. Springer, 63–77.
[21] Yoad Lewenberg, Yoram Bachrach, Yonatan Sompolinsky, Aviv Zohar, and Jeffrey S Rosenschein. 2015. Bitcoin mining pools: A cooperative game theoretic analysis. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 919–927.
[22] Loi Luu, Ratul Saha, Inian Parameshwaran, Prateek Saxena, and Aquinas Hobor. 2015. On Power Splitting Games in Distributed Computation: The Case of Bitcoin Pooled Mining. In *Computer Security Foundations Symposium (CSF)*. IEEE.
[23] Ralph C Merkle. 1980. Protocols for Public Key Cryptosystems.. In *Symposium on Security and privacy*. IEEE.
[24] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. 2016. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In *European Symposium on Security and Privacy*. IEEE.
[25] Pete Rizzo 2018. Double Spending Risk Remains After July 4th Bitcoin Fork. https://www.coindesk.com/double-spending-risk-bitcoin-network-fork/. (2018). [Online; accessed 30-April-2018].
[26] Pool Distribution 2018. Pool Distribution. https://btc.com/stats/pool. (2018). [Online; accessed 2-May-2018].
[27] Proof 2018. Proof of Work. https://en.bitcoin.it/wiki/Proof_of_work. (2018). [Online; accessed 30-April-2018].
[28] Meni Rosenfeld. 2011. Analysis of Bitcoin Pooled Mining Reward Systems. *arXiv preprint arXiv:1112.4980* (2011).
[29] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. 2015. Optimal Selfish Mining Strategies in Bitcoin. *arXiv preprint arXiv:1507.06183* (2015).
[30] Itay Tsabary and Ittay Eyal. 2018. The Gap Game. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 713–728.
[31] ViaBTC 2018. ViaBTC. https://pool.viabtc.com/. (2018). [Online; accessed 3-May-2018].
[32] Seunghyun Yoo, Seungbae Kim, Joshua Joy, and Mario Gerla. 2018. Promoting Cooperative Strategies on Proof-of-Work Blockchain. In *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.