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.

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 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, 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’s 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 identify 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 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 “Eligitus” 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 the attacker systematically 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 generates intentional forks through pools instead of generating them by herself. Of course, in the victim pool, the manager can exploit 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 through 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 $I$ be the set of all pools and solo miners¹, and denote by $\alpha_i$ the computational power of $i \in I$. For analytical convenience, we normalize the total computation power with 1, and thus $\sum_{i \in I} \alpha_i = 1$. 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

¹A 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 \( \frac{R_i}{a_i} \), where \( R_i \) 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 \( (N, (A_i)_{i \in N}, (U_i)_{i \in N}) \), where \( N \) 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 \( a \in (A_i)_{i \in N} \). 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 \( a^t := (a^t_i)_{i \in N} \) denote the action profile at stage \( t \), i.e., the actions by all players at stage \( t \). Also, we denote by \( a_i := (a^t_i)_{0 \leq t < T} \) the actions of player \( i \) for \( T \) stages. For \( t \geq 1 \), 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 \( h^t \) and chooses the next action \( a^t_i \) according to i’s strategy, thus \( a^t = (a^t_i)_{i \in N} \) is determined. Here, a strategy \( s_i \) for player \( i \) in the repeated game is a sequence of maps \( s^t_i \)—one for each stage \( t \)—that map a \((t-1)\)-history \( h^{t-1} \) to an action \( a^t \) 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 \( a = (a_1, a_2, \ldots) \), the payoff \( U_i(a) \) of player \( i \) in the corresponding repeated game is the discounted average payoff, i.e.,

\[
U_i(a) = \sum_{t=0}^{\infty} \delta^{t-1} U_i(a^t),
\]

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 \( U_i(s_i, s_{-i} | h^t) \). Then the strategy profile \( s^* = (s^*_i)_{i \in N} \) is a subgame perfect Nash equilibrium if

\[
U_i(s^*_i, s^*_{-i} | h^t) = \max_{s^t_i \in S_i} U_i(s_i, s^*_{-i} | h^t)
\]

for all \( i \in N \), \( h^t \in H^t \), \( \forall t > 0 \),

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\( \text{p} \) and Pool\( \text{q} \)). 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 \( U_i(a^t) \). First, each pool \( i \)'s action is defined in terms of which attack is performed and the
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].

$$U_i^{FF}(f_i, f_{-i}) = \frac{a_i - b_i}{(1 - f_i - f_{-i})(a_i + f_i)} + \frac{f_i(1 - a_i - a_{-i})}{(1 - f_i)(a_i + f_i)}$$

In (2), the first term is obtained from the honest mining of each pool, achieved with the computational power remaining after deducting the inflation power. Note that Pool$_i$ gets a reward of $\frac{a_i - a_{-i}}{1 - f_i}$ 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 inflation power of Pool$_i$ generate a block, and the probabilities of these events are $\frac{1 - a_i - a_{-i}}{1 - f_i}$ 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 inflation powers of Pool$_1$ and Pool$_2$ find blocks, and then a fork with three branches occurs. If the inflation power of Pool$_1$ finds a block faster than that for Pool$_{-i}$, its probability would be $f_i - \frac{f_i}{1 - f_i} - \frac{1 - a_i - a_{-i}}{1 - f_i - f_{-i}}$. On the other hand, if the inflation power of Pool$_{-i}$ generates a block faster than that for Pool$_i$, its probability would be $\frac{f_{-i}}{1 - f_i} - \frac{1 - a_i - a_{-i}}{1 - f_i - f_{-i}}$. Considering these facts, the third term is derived. Lastly, the fourth term is from its inflation 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).

$$U_i^{BB}(b_i, b_{-i}) = \frac{a_i - b_i}{(1 - b_i - b_{-i})(a_i + b_i)} + \frac{(U^{BF}(f_i, f_{-i}) + 1) \cdot b_i}{a_i + b_i} - 1.$$

**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$_1$ and Pool$_{-i}$ execute FAW and BWH attacks, respectively. Then, the payoff $U_i^{FB}(f_i, b_{-i})$, which quantifies the extra reward density, turns out to be given by:

$$U_i^{FB}(f_i, b_{-i}) = \frac{a_i - f_i}{(1 - f_i - b_{-i})(a_i + b_i)} + \frac{(U^{BF}(f_i, f_{-i}) + 1) \cdot b_i}{a_i + b_i} - 1.$$

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 inflation 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\textsubscript{i} because Pool\textsubscript{−i} does not generate forks in Pool\textsubscript{i}.

Now, when Pool\textsubscript{i} and Pool\textsubscript{−i} execute BWH and FAW attacks with infiltration power \(b_i\) and \(f_{−i}\), respectively, we have:

\[
    U^{BF}(b_i, f_{−i}) = \frac{a_i - b_i}{(1 - b_i - f_{−i})(a_i + f_{−i}) + 1 - b_i} + f_{−i} \left(1 - \frac{a_i - a_{−i}}{(1 - b_i - f_{−i})(a_i + f_{−i}) + 1}ight) b_i. \tag{5}
\]

Because only Pool\textsubscript{−i} executes the FAW attack, forks are generated by Pool\textsubscript{i} only in Pool\textsubscript{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\textsubscript{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) \(a^* = (a_i^*, a_{−i}^*)\) in the stage game; it is characterized as:

\[
    (a_i^*, a_{−i}^*) = \left( (f_i^*, 0), (f_{−i}^*, 0) \right), \quad \text{where} \quad f_i^* > 0 \text{ and } f_{−i}^* > 0. \tag{6}
\]

Further, the following payoff values are obtained for different cases of two pools’ computational powers \(a_i\) and \(a_{−i}\):

\[
    U_i(a_i^*, a_{−i}^*) > 0, \quad U_{−i}(a_i^*, a_{−i}^*) < 0 \quad \text{if} \quad a_i > a_{−i}, \tag{7}
\]

\[
    U_i(a_i^*, a_{−i}^*) = U_{−i}(a_i^*, a_{−i}^*) = 0 \quad \text{if} \quad a_i = a_{−i}. \tag{8}
\]

(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 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 \((a_i^t, a_{−i}^t)\). 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 \(stnd\) representing the standing of Pool\textsubscript{i} is maintained by each pool; \(stnd\) represents whether Pool\textsubscript{i} has followed ARS\(K\) well or not at the previous stage. We use the notation \(a_i^t, ars\) to refer to the action when ARS\(K\) is played, in order to differentiate an action \(a_i^t\) from a different strategy. Thus, \(a_i^t = a_i^t, ars\) when Pool\textsubscript{i} plays ARS\(K\).

**ARS\(K\):** In ARS\(K\), Pool starts to cooperate with no-attack when \(t = 0\), and initializes its standing variable \(stnd\) to GOOD. At each stage \(t\), Pool\textsubscript{i} first sets its standing \(stnd_i\), depending on whether its stage \(t-1\) action \(a_i^{t-1}\) matches \(a_i^{t-1}, ars\) from ARS\(K\) (SI). Thus, if Pool\textsubscript{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: \((stnd, stnd, -i) = (G, G), (G, B), (B, G), (B, B)\), 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\textsubscript{i}’s action \(a_i^t, ars\) at stage \(t\) according to ARS\(K\). In the figure, Rt is the output of Retaliate with \(K\) (denoted by Retaliate\(K\) in Algorithm 1), and −(+) 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

Figure 2: ARS\(K\) has four states, depending on the two pools’ standings. In each state box, \((G \text{ or } B, G \text{ or } B)\) represents \((stnd_i, stnd_{−i})\), and Pool\textsubscript{i}’s action \(a_i^t, ars\) 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 state change.
not present the action tuples that do not change a state (e.g., in (G, G), the action (0, 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 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 \(a_{t-1}^i\) and \(a_{t-1}^{\text{ars}}\), which are two inputs of Retaliate, should be 0 (see the actions at edges toward (G, B)). This is because (i) if \(a_{t-1}^i \neq 0\), it indicates that the opponent’s standing at stage \(t = 1\) was B4D, and thus Pool_i should not attack according to ARS_K, i.e., \(a_{t-1}^{\text{ars}} = 0\), or (ii) if \(a_{t-1}^{\text{ars}} \neq 0\), \(a_{t-1}^i\) should be 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 \(a_{t-1}^i\), \(a_{t-1}^{\text{ars}}\), \(\text{stdn}_i\), and \(\alpha\) 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 \(a_{t-1}^i\), \(a_{t-1}^{\text{ars}}\), and \(a_{i,t-1}^{\text{ars}}\), 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

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

1. **Start when** \(t = 0\):
   - Start the stage game with no-attack (i.e., \(a_{0}^i = 0\)), and set a variable \(\text{stdn}_i = 0\).

2. **At each stage** \(t \geq 1\):
   - **S1. Set the standing of this stage.**
     - If \((a_{t-1}^i = a_{t-1}^{\text{ars}})\) and \(\text{stdn}_i = 0\), \(\text{stdn}_i = 0\) and \(a_{t}^i = 0\).
   - **S2. Estimate the infiltration power.**
     - If \((a_{t-1}^i = 0\) and \(\text{stdn}_i = 0\))
       - \(a_{t}^i = \text{Retaliate}_K(a_{t-1}^i, a_{t-1}^{\text{ars}}, a_{t-1}^{\text{ars}}, a_{i,t-1}^{\text{ars}})\)
     - else \(a_{t}^i = 0\).
   - **S3. Output** \(a_{t}^i\).

### Algorithm 2: Retaliate Subroutine where \(M^f_i(\alpha, a_{i-1})\) and \(M^B_i(\alpha, a_{i-1})\) are given in (13) and (14), respectively.

- **Input:** Local. computation power \(a_i\), previous action \(a_{t-1}^i\)
- **Input:** Opponent. computation power \(\alpha_{i-1}\), previous action \(a_{t-1}^{\text{ars}}\), previous ARS action \(a_{i,t-1}^{\text{ars}}\)
- **Output:** Pool_i’s action \(a_{t}^i\)

#### 5.1 Retaliation with FAW

1. **Construct the infiltration set for FAW**
   \[ I_{faw} = I_{\text{ars}}(a_{t-1}^i, a_{t-1}^{\text{ars}})\]

2. **Find the following two sets \(F_1\) and \(F_2\) as follows:**
   \[ F_1 = \min \left\{ f_{i} \in I_{faw} \mid U(a_{t-1}^i, a_{t-1}^{\text{ars}}) - U(a_{t-1}^i, a_{i,t-1}^{\text{ars}}) \geq U_{\text{ars}}(f_i, 0, 0) \right\}, \]
   \[ F_2 = \min \left\{ f_{i} \in I_{faw} \mid f_{i} \in I_{faw} \right\}, \]

3. **Compute the infiltration power**
   \[ f_{i} = \min \{F_1 \cup F_2\}, \text{ and set } a_{t}^i = (f_i, 0). \]

#### 5.2 Retaliation with BWH

1. **Construct the infiltration set for BWH**
   \[ I_{bwh} = I_{\text{ars}}(a_{t-1}^i, a_{t-1}^{\text{ars}})\]

2. **Find the following two sets \(F_1\) and \(F_2\) as follows:**
   \[ B_1 = \min \left\{ b_{i} \in I_{bwh} \mid U(a_{t-1}^i, a_{t-1}^{\text{ars}}) - U(a_{t-1}^i, a_{i,t-1}^{\text{ars}}) \geq U_{\text{ars}}(f_i, 0, 0) \right\}, \]
   \[ B_2 = \min \left\{ b_{i} \in I_{bwh} \mid b_{i} \in I_{bwh} \right\}, \]

3. **Compute the infiltration power**
   \[ f_{i} = \min \{B_1 \cup B_2\}, \text{ and set } a_{t}^i = (b_i, 0). \]

#### 5.3 Terminate. Output \(a_{t}^i\).
have more strength in damaging the opponent more severely [17]. In the full version [19], we prove the non-emptiness of $H_{bwh}$.

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 $f_2$ is chosen to be closest to the infiltration power $M^F_i(\alpha_1, \alpha_2)$ maximizing Pool$_i$'s payoff $U_i$, expressed as:

$$M^F_i(\alpha_1, \alpha_2) = \frac{-1 + \sqrt{1 - \alpha - (1 + \alpha - \alpha_1)(1 - \alpha + \alpha_2)}}{\alpha_i(1 - \alpha_i - \alpha_i)}$$

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 $M^B_i(\alpha_1, \alpha_2)$ is derived from [9], given as:

$$M^B_i(\alpha_1, \alpha_2) = \frac{-\alpha_{-i}(1-\alpha_i) + \sqrt{\alpha_{-i}^2 + 4\alpha_{-i}(1-\alpha_i)(1+\alpha_i-\alpha_{-i})}}{1-\alpha_i-\alpha_{-i}}$$

Note that Retaliate outputs $\emptyset$ 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 $\emptyset$ in this case, and Pool$_i$ would not attack for retaliation because $IP_{law}$ 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 $\delta$.

**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 $\alpha_1$. Moreover, $(ARS_K, ARS_K)$ 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 $\delta$ 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 $\delta$ 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$_i$ and Pool$_j$’s sizes. We consider a case in which Pool$_i$ deviates from $ARS_K$ to attack Pool$_j$ while maximizing its payoff $U_i$ during one stage. As a result, Pool$_j$ would retaliate against Pool$_i$ according to $ARS_K$. In this section, we consider a strategy $ARS_{1-}$, where $K$ is close to 1.

Fig. 3 represents when Pool$_i$ 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 $r_i = (r^F_i, r^B_i)$, where $r^F_i$ and $r^B_i$ are proportions of infiltration power $f_i$ and $b_i$ for Pool$_i$’s computational power, respectively (i.e., $r^F_i = \frac{\bar{f}_i}{\bar{f}_1}$, $r^B_i = \frac{\bar{b}_i}{\bar{b}_1}$). Fig. 3a represents Pool$_2$’s infiltration ratio $r^F_2$ 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 $r^B_2$ 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 $U_1(\alpha^F_1, \alpha^F_1) \frac{U_1(\alpha^F_1, \alpha^F_1)}{2}$ 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 1’s infiltration ratio for retaliation using the FAW attack.
(b) Pool 1’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 1 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 1 for two stages.

Figure 4: Pool 1 optimally executes the BWH attack.

(a) Pool 1’s infiltration ratio for retaliation with the FAW attack against Pool 1’s FAW attack.
(b) Pool 1’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 1 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 1 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\textsubscript{1} makes FAW attacks unprofitable. Moreover, Fig. 3d shows that Pool\textsubscript{2} can completely recover a loss from Pool\textsubscript{1}’s attack in the case where Pool\textsubscript{1} retaliates with the FAW attack.

Fig. 4 represents when Pool\textsubscript{i} optimally executes the BWH attack to maximize its short-term payoff. Similar to Fig. 2, Figs. 4a and 4b represent Pool\textsubscript{i}’s infiltration ratio \(r_{i}^{F}\) and \(r_{i}^{B}\), respectively. Figs. 4c and 4d respectively represent the average payoffs of Pool\textsubscript{1} and Pool\textsubscript{2} for two stages. Fig. 4c shows that Pool\textsubscript{1} always suffers a loss from the retaliation of Pool\textsubscript{2} when Pool\textsubscript{2} 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\textsubscript{i}’s infiltration ratio used for attacks, assuming that Pool\textsubscript{1}’s size is 0.2 (20%). Figs. 5 and 6 represent Pool\textsubscript{1}’s execution of FAW and BWH attacks, respectively. The \(x\) and \(y\)-axes are Pool\textsubscript{1}’s infiltration ratio used for attack and Pool\textsubscript{2}’s sizes, respectively. Figs. 5a and 5b show the infiltration ratio \(r_{i}^{F}\) and \(r_{i}^{B}\) for retaliation against Pool\textsubscript{1}, respectively. Because the extent of retaliation by ARS depends on the loss caused by the opponent’s attack, Pool\textsubscript{1}’s infiltration ratio for retaliation depends on Pool\textsubscript{1}’s attack infiltration ratio. Fig. 5c and 5d represent the average payoffs of Pool\textsubscript{1} and Pool\textsubscript{2}, respectively, for two stages in which Pool\textsubscript{1} executes the FAW attack and then Pool\textsubscript{2} retaliates. Pool\textsubscript{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\textsubscript{2} can earn extra profit in the process of retaliation, as shown in Fig. 5d. Similar to Fig. 5, Fig. 6 shows Pool\textsubscript{2}’s infiltration ratio \(r_{i}^{B}\) for retaliation, and the attacker’s and victim’s average payoffs for two stages when Pool\textsubscript{1} executes the BWH attack. In most cases, Pool\textsubscript{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\textsubscript{1}, we omit Pool\textsubscript{1}’s infiltration ratio \(r_{i}^{B}\) for retaliation with the BWH attack because the region of such cases is very small (see small areas bordered 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\textsubscript{1}.

<table>
<thead>
<tr>
<th>Name</th>
<th>(r_{i}^{F}, r_{i}^{B}) (%) against FAW</th>
<th>Total Payoff</th>
<th>(r_{i}^{F}, r_{i}^{B}) (%) against BWH</th>
<th>Total Payoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>AntPool</td>
<td>(0, 14.33%)</td>
<td>-1.89%</td>
<td>(46.2%, 0)</td>
<td>-0.78%</td>
</tr>
<tr>
<td>ViaBTC</td>
<td>(0, 13.7%)</td>
<td>-0.54%</td>
<td>(47.2%, 0)</td>
<td>-0.15%</td>
</tr>
<tr>
<td>DPool</td>
<td>(0, 17.71%)</td>
<td>-0.004%</td>
<td>(0, 13.14%)</td>
<td>-1.1%</td>
</tr>
<tr>
<td>Bixin</td>
<td>(0, 21%)</td>
<td>-0.025%</td>
<td>(0, 13%)</td>
<td>-0.63%</td>
</tr>
</tbody>
</table>

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\textsubscript{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\textsubscript{1}, the four pools make the attacks of BTC.com unprofitable.

### 6 IDENTIFYING THE OPPONENT’S ATTACK

To follow ARS, Pool\textsubscript{i} needs to know seven parameters in Table 2: \(\alpha_{i}, \text{std}_{i}, a_{i}^{t-1}, \text{std}_{i-1}, a_{i-1}^{t}, \text{std}_{i-1, i, \text{ars}}^{t-1}, a_{i, i, \text{ars}}^{t-1}\). 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\textsubscript{i} already knows \(\alpha_{i}, \text{std}_{i}, a_{i}^{t-1}\), and \(a_{i-1}^{t-1}\), which are referred to as *internal variables* in this paper. Also, Pool\textsubscript{i} can easily obtain \(\alpha_{i-1}\) because the computational power of pools can be approximately calculated from the mined block information [2]. Among the remaining parameters, \(\text{std}_{i-1}, a_{i-1}^{t-1}\), and \(a_{i-1, i, \text{ars}}^{t-1}\), Pool\textsubscript{i} needs to know \(a_{i-1, i, \text{ars}}^{t-1}\) because \(\text{std}_{i-1}\) is determined by \(a_{i-1}^{t-1}\) and \(a_{i-1, i, \text{ars}}^{t-1}\). Moreover, the value \(a_{i-1, i, \text{ars}}^{t-1}\) is 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\textsubscript{i} can determine \(a_{i-1, i, \text{ars}}^{t-1}\) by obtaining the opponent’s action \(a_{i-1}^{t-1}\) at stage \(t - 2\). As a result, Pool\textsubscript{i} only needs to know the opponent’s previous action in order to determine \(\text{std}_{i-1}, a_{i-1, i, \text{ars}}^{t-1}\), and \(a_{i-1, i, \text{ars}}^{t-1}\).

To guess the opponents’ actions, Pool\textsubscript{i} can plant moles in other pools. Through the moles, Pool\textsubscript{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 \(\delta\) 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.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\alpha_{i})</td>
<td>Computational power of Pool\textsubscript{i}</td>
</tr>
<tr>
<td>(\text{std}_{i})</td>
<td>Standing of Pool\textsubscript{i}</td>
</tr>
<tr>
<td>(a_{i}^{t-1})</td>
<td>The action of Pool\textsubscript{i} at time (t - 1)</td>
</tr>
<tr>
<td>(\alpha_{i-1})</td>
<td>Computational power of the opponent</td>
</tr>
<tr>
<td>(\text{std}_{i-1})</td>
<td>Standing of the opponent</td>
</tr>
<tr>
<td>(a_{i-1}^{t-1})</td>
<td>The action of the opponent at time (t - 1)</td>
</tr>
<tr>
<td>(a_{i-1, i, \text{ars}}^{t-1})</td>
<td>Output of ARS of the opponent at time (t - 1)</td>
</tr>
</tbody>
</table>

### Detection of Attacks:

First, Pool\textsubscript{i} must determine whether it is being attacked. Indeed, Pool\textsubscript{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*\textsuperscript{3} and *short-term miners*\textsuperscript{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% attacks directly.

\textsuperscript{3}Miners who submitted a relatively small number of FPoWs

\textsuperscript{4}Miners who stayed in the pool for relatively short period of time
Identification of Attackers: The FAW or BHW 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 $\alpha$ and $\beta$ to represent the attacker’s size and the victim’s size, respectively, instead of $\alpha_{-i}$ and $\alpha_I$. We also suppose the attacker infiltrates a fraction $\gamma$ of the victim’s computational power into the victim pool. Moreover, we denote by $R_d$ 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 $N$, $R_d$ would be $\frac{1}{\alpha} + \frac{NY}{\beta + \gamma}$. We can easily check that $N$ has a geometric distribution with a parameter \((1-\gamma)a\) for FAW attack, and \((1-\gamma)a\) for BHW attack (see the full version [19] for details).

If $\alpha$ does not change and the pool does not attack, the reward density $R_{dp}$ for the period $P$ is fixed at $\frac{1}{\beta}$. Meanwhile, if the pool attacks, $R_{dp}$ would continuously change depending on $N$. Therefore, the victim can identify the attacking pool immediately by observing its variances in $R_{dp}$ after planting moles in pools. However, even if attacks do not occur, the reward density as well as $\alpha$ and $\beta$ changes in practice. If $\alpha$ 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 $R_{dp}$ 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 $R_{dp}$ would usually

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. $R_{dp}$ 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 $R_{dp}$ 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 BHW 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 BHW attacker, the pool’s reward density $R_{dp}$ 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]).
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 $R_{dp_l}$ 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$_i$; $i = 1 \text{ to } n$). 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 $\text{stnd}_{ij}, a_{ij}^{t-1}$, and $a_{ij,ars}^{t-1}$. Then, Pool$_i$ maintains $n-1$ dimensional standing vectors $(\text{stnd}_{ij})_{i \neq j}$ and action vectors $(a_{ij}^{t-1})_{i \neq j}, (a_{ij,ars}^{t-1})_{i \neq j}$. Note that $a_{ij}^{t-1} = (f_{ij}^{t-1}, b_{ij}^{t-1})$, where $f_{ij}^{t-1}$ and $b_{ij}^{t-1}$ 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 $\text{stnd}_{ij}$ and $\text{stnd}_{ji}$ 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 $f_{ij}$ and $b_{ij}$ 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$_3$), DPoP (Pool$_4$), and Bixin (Pool$_5$)) at the same time while maximizing its short-term payoff, and four other pools follow ARS$_K$ (see Table 3). In Table 3, we denote the infiltration ratios for FAW and BWH attacks of Pool$_i$ against Pool$_j$, $\text{ars}_{ij}^{F}$ and $\text{ars}_{ij}^{B}$, 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, DPoP, 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$_j$ [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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Computational Power</th>
<th>( r_{ij}^F ) against FAW</th>
<th>( (r_{ij}^F, r_{ij}^B) ) against FAW</th>
<th>( U_i )</th>
<th>( r_{ij}^B ) against BWH</th>
<th>( (r_{ij}^F, r_{ij}^B) ) against BWH</th>
<th>( U_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AntPool</td>
<td>15%</td>
<td>22.7%</td>
<td>(0, 13.67%)</td>
<td>-5.4%</td>
<td>9.5%</td>
<td>(46.2%, 0)</td>
<td>-1.55%</td>
</tr>
<tr>
<td>ViaBTC</td>
<td>10%</td>
<td>15.1%</td>
<td>(0, 13.7%)</td>
<td>6.4%</td>
<td>2.2%</td>
<td>(0, 13.14%)</td>
<td></td>
</tr>
<tr>
<td>DPool</td>
<td>3.5%</td>
<td>5.3%</td>
<td>(0, 13.14%)</td>
<td>1.3%</td>
<td>0</td>
<td>(0, 13%)</td>
<td></td>
</tr>
<tr>
<td>Bixin</td>
<td>2%</td>
<td>3%</td>
<td>(0, 13%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 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


[22] Lei Liu, Rui Li, Xuan Hong Vu, and Hubert Ritzdorf. 2015. On the Performance of Pooled Mining. In Computer Security Foundations Symposium (CSF). IEEE.