Optimal Selfish Mining Strategy for StrongChain

I. Modeling StrongChain

We model StrongChain according to the algorithm proposed by [1] for γ from $0, 0.5, 1$, with various values of α, utilizing an MDP solver on MATLAB (a toolbox provides the resolution of discrete-time Markov Decision Processe, developed by Chadés et al. [2]) and compare the result with Nakamoto Consensus as well as Fruitchain. The ratio T_w/T_s could denotes the distribution of units in StrongChain. For example, when $T_w/T_s = 2$, half of the units are weak headers and the other half are blocks in expectation.

A straightforward encoding of a StrongChain state includes both chain's block/weak header mining sequences. However, the number of states will grow exponentially with the block race length if we encode each combinination that has one block embedded with several weak headers. To compress the state space, we observe that in all results of block race, the honest chain will always be either adopted or abandoned as a whole. Thus we encode of number of blocks in both chains, honest weak headers, attacker's weak headers embedded in blocks and attacker's weak headers not contained in blocks. This simplification limits the attaccker's ability: when the most recent block was built by the honest miners and the PoW of attacker's chain is bigger than the PoW of honest chain, the attacker cannot publish a certain number of blocks and weak headers of the same PoW with the honest chain (this action is called *match*). Hence our StrongChain MDP model favors the honest miners and the complete MDP design is in Appendix A.

As can be seen from Fig. 1, the results demonstrate a remarkable gap between the attacker' revenue in StrongChain and the revenue in other consensus algorithms. The maximum numbers of blocks in StrongChain is set to 24 and T_w/T_s is set to 2; the maximum numbers of blcoks in Fruitchain is set to 13 and $Ratio_{f2b} = 2$ which represents the number of fruits built per blcok; and the maximum fork length of Nakamoto Consensus is set to 80. Thus the attacker is forced to end the block race once the border numbers are reached. It demonstrates that StrongChain performs much better than Nakamoto Consensus and Fruitchain especially when attacker's mining power α is larger than 35% and the influence of γ is diminished significantly in comparision to NC.

Moreover, in StrongChain, we study the influence of T_w/T_s on the attacker's relative revenue. Three different T_w/T_s values 2, 4 and 8 are selected to verify and the maximum block numbers of blocks are set to 15. However, the relative revenue increases as T_w/T_s increases, as it shown in Fig. 2. Namely, a larger T_w/T_s does not help improve the performance of StrongChain, which might be because more weak headers give the attacker more windows to try *wait* action and orphan honest blocks and headers.

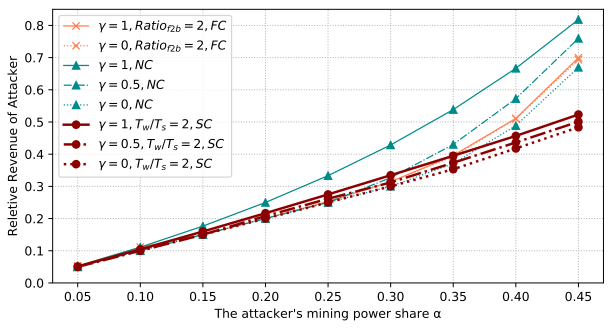

Fig. 1. Attacker's relative revenue of Nakamoto Consensus (NC), Fruitchain (FC), and StrongChain (SC). Ideally, relative revenue of attacker is α

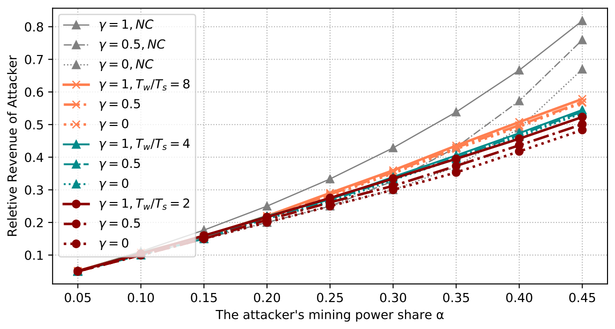

Fig. 2. Attacker's relative revenue of StrongChain with various T_w/T_s

II. Optimal Selfish Mining Policy

Below we illustrate the general behaviour of the optimal policy in StrongChain returned by MDP Model. The core idea of the optimal policy is decribed by Fig. 3 and the concret attack strategy could be found in Algorithm 1 and Algorithm 2.

A. Overview of the optimal policy

Looking into lots of optimal policies with different values of variables (α, maximum number of blocks, γ, and T_w/T_s), we see the main idea of the optimal attack strategy which is depicted in Fig. 3, where sb denotes the number of blocks generated by selfish nodes, hb the number of blocks generated by honest nodes, $SPoW$ the PoW of the secret branch mined by selfish miners, and $HPoW$ the PoW of the branch mined by honest nodes. Due to the fact that in StrongChain the chain with the largest aggregated PoW is determined as the current one, the difference between $SPoW$ and $HPoW$ could provide decision basis for the malicious selfish miner and the value of $SPoW - HPoW$ will be occasionally denoted by δ. Alternatively, since only weak headers contained in blocks could get a reward, it will make a huge difference whether the number of blocks is zero.

Thus the attacker would choose different tactics when she meets different situations: $i)$ when $sb = 0$ and $hb = 0$, namely, the first several units in a block race are only selfish weak headers and honest weak headers, the attacker will not publish them regardless of how large $SPoW$ is and keep working on its branch until a block is built, as the weak headers not included in a block receive no reward; $ii)$ when $sb = 0$ and $hb > 0$, viz, there are some honest blocks containing honest weak headers while no selfish blocks have been created, there will be two cases: if the gap between $SPoW$ and $HPoW$ is not too large (no less than $P1$), it allows the attacker to "catch up from behind", otherwise, the attacker accepts the honest network's chain and starts to mine on top of the latest honest block; $iii)$ when $sb > 0$ and $hb = 0$, i.e., the attacker has built several blocks privately while honest miners have no blocks, three things arise: the attacker publishes all the units when $0 < \delta \leq P2$; the attacker keeps working on its branch when $\delta > P2$ or $\delta < 0$; and the attacker publishes some confilicting blocks and weak headers of the same PoW with public chain when δ; and $iv)$ when $sb > 0$ and $hb > 0$, four cases occur: the attacker publishes all the blocks and weak headers when $0 < \delta \leq P2$; the attacker keeps mining on its private chain when $P3 \leq \delta \leq P2$; the attacker gives its branch up and adopts the honest chain when $\delta < P3$; and the attacker chooses *match* action when $\delta = 0$.

The intuition behind this policy is as follows: First, if there are no honest blocks (i.e., $hb = 0$), the attacker will refuse to adopt the public chain since the latest public block does not change. Conversly, if $hb > 0$, the attacker would *adopt* the public chain to refrain from wasting time and hash power once it is far behind its competitor. Secondly, it might seem odd that the attacker would do *wait* rather than *override* when $SPoW$ is much larger than $HPoW$ (i.e., $\delta > P2$), while the states in the range $\delta > P3$ are unreachable since δ will be reset before reaching the state $\delta > P3$ via *override* action. Thus it is trivial that which action would be performed in such state. Thirdly, the attacker attempts to catch up with the more powerful public chain from a disadvantage (e.g., $\delta < P1$ and $\delta < P3$). Formally the values of $P2$ and $P3$ depend on the PoW of selfish chain as well as the attacker's hash power α, and the detail is discussed in Algorithm 2. Our results show that, as $SPoW$ and α increase, the attacker utilizes *wait* more extensively, improving the chance of winning in the block race.

B. Evaluation of the optimal policy

We evaluate the optimal policy for an attacker with $T_w/T_s = 2$, α in range $[0.1, 0.45]$ and γ from $\{0, 0.5, 1\}$ and we can make a series of observations. As it shown in Fig 4, among four actions *adopt* accounts for the largest percentage and *match* is the performed used action as it is a rare event that two rival chains possess the same PoW. On the one hand, as α goes up from 0.1 to 0.45, the attacker tends to use less *adopt* while employing more *wait*. This confirms the basic idea which we have discussed in Section II-A. Besides, the ratios of *match* and *override* keep almost unchanged as α increases. On the other hand, the use of *adopt* and *override* shows a decline with the γ rising whereas the use of *wait* increases. Therefore, we claim that more utilization of *wait* and less *adopt* enhance the profit of the attacker.

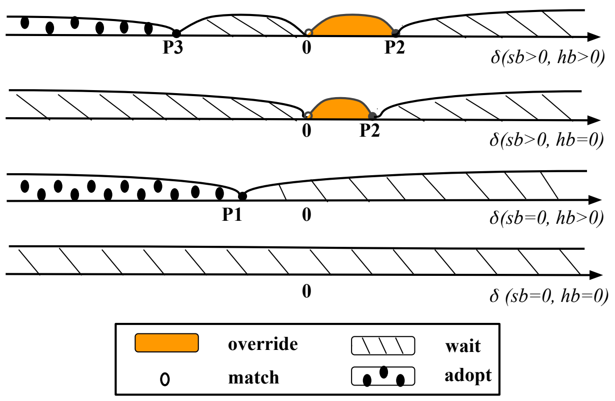

Fig. 3. Main idea of the optimal selfish mining policy in StrongChain, where sb denotes the number of blocks generated by attacker; hb denotes the number of blocks generated by honest nodes; δ denotes the difference between PoW of the selfish secret chain ($SPoW$) and PoW of the honest chain ($HPoW$).

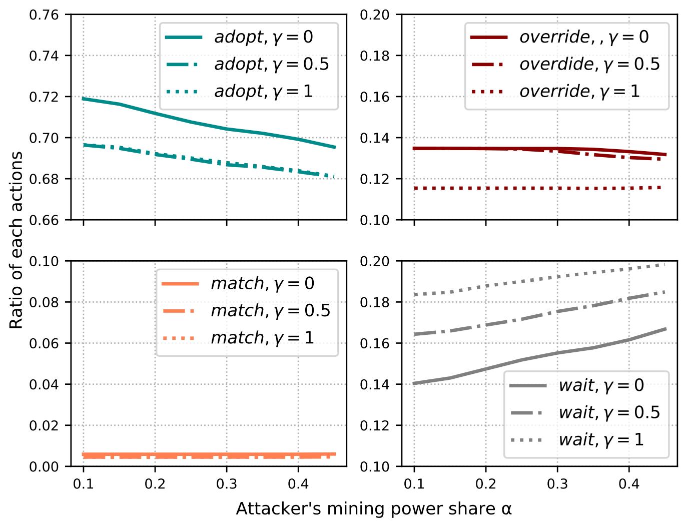

Fig. 4. The proportion of actions (*adopt, override, match, wait*) in the optimal selfish mining attack policy for StrongChain.

References

[1] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 515–532.

[2] I. Chadès, G. Chapron, M.-J. Cros, F. Garcia, and R. Sabbadin, "Mdptoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems," *Ecography*, vol. 37, no. 9, pp. 916–920, 2014.

Algorithm 1: Optimal Selfish Mining Policy in StrongchainChain

```
function OnSb=0():
    if $Hb = 0$ then
        wait
    else
        /*$\delta = SPoW - HPoW$*/
        if $\delta \geq P1$ then
            wait
        else
            adopt
function OnSb>0():
    if $\delta = 0$ then
        if $fork = ishb$ then
            match
        else
            wait
    else
        if $Hb > 0$ then
            if $\delta > P2$ then
                wait
            else if $0 < \delta \leq P2$ then
                override
            else if $0 > \delta \geq P3$ then
                wait
            else
                adopt
        else
            if $\delta > P2$ then
                wait
            else if $0 < \delta \leq P2$ then
                override
            else
                wait
```

Algorithm 2: Set Parameters

```
function DefineP($SPOW$):
    while mining do
        $P1 \leftarrow T_s/T_w - 1$
        if $Hb = 0$ then
            $P2 \leftarrow T_s/T_w$
        else if $SPOW > T_w/T_s + 1$ then
            $P2 \leftarrow 1 + T_s/T_w$
            $P3 \leftarrow -(1 + 3 * T_s/T_w)$
        else if $SPOW > T_w/T_s$ then
            $P2 \leftarrow 1$
            $P3 \leftarrow -(1 + 2 * T_s/T_w)$
        else
            $P2 \leftarrow 1 - T_s/Tw$
            $P3 \leftarrow -(1 + T_s/T_w)$

function AdjustP($\alpha$):
    if $min(Hb, Ab) > 0$ then
        if $\alpha \geq 0.2$ then
            $min(P3) \leftarrow -1$
        else
            $max(P3) \leftarrow -(1 - 2 * T_s/T_w)$
```

Appendix A
StrongChain MPD Design

Unlike the MDP of Nakamoto Consensus algorithm where a block is generated at the end of each step, in StrongChain MDP, each step ends with the creation of a unit—either a block or a weak header.

A. State Space

Based on our observations in Section I, the state space is defined by 6-tuples of the form *sb, hb,* sw_c*, hw,* sw_s*, fork*. The first two entries represent the number of blocks of the selfish miner's chain and honest miners' chain, built after the latest fork. The feild sw_c represents the number of selfish weak headers which are contained by the blocks. Accordingly, sw_s denotes the number of selfish weak headers which haven't been included by blocks and we call them "scattered selfish weak headers". Additionally, *hw* represents the number of honest weak headers and *fork* obtains three values, dubbed *ishb, nothb* and *active*. *ishb* denotes that the last block is built by honest nodes, and $nothb$ denotes the complete opposite case. The third label *active* represents the case where the honest network is already split owing to a *match* action.

B. Actions

The attacker can choose from four actions: *Adopt, Override, Match and Wait.*

- *Adopt.* The attacker abondon its secret chain and mine on top of the public chain. This action is always feasible.
- *Override.* The attacker publish all the units to orphan the honest network's chain, and is feasible whenever $SPoW > HPoW$.
- *Match.* The attacker publish until the published attacker's chain is of the same PoW with the public chain to cause a tie, and then keep mining on the secret chain. Feasible when the $fork = ishb$ and $SPoW = HPoW$ or $SPoW - sw_s = HPoW$.
- *Wait.* The attacker does not publish any unit and keeps mining on the secret chain, which is always feasible.

This limited set of actions allows constrained *match* function, due to that the attacker cannot do *match* in all atates satisfying $SPoW - HPoW > 0$. This is mainly because the detail combination of each block with several weak headers is omitted to compress the state space. Consequently, the attacker could do *match* in two cases: 1 the PoW of both chains are the same; 2 the PoW of selfish chain except the scattered weak headers is equal to the PoW of public chain.

C. Reward Allocation and State Transition

A valid weak header receives T_s/T_w so that on average one unit of reward is issued per block. The honest miners get $R_c = hb + hw * T_s/T_w$ after *Adopt*. After *Override*, the attacker gets rewards for all released units which is $R_s = sb + (sw_c + sw_s) * T_s/T_w$. After these two actions, information regarding the blocks or weak headers which are permanently accepted or abandoned by both miners will be cleared in the new state. No reward is generated after *Wait* with $fork \neq active$.

Besides, there are four outcome states after *Adopt, Override* and *Wait* when $fork \neq active$, where the new mining product might be an attacker block, an attacker weak header, an honest block or an honest weak header, with probability $\alpha * T_s/T_w$, $\alpha * (1 - T_s/T_w)$, $(1-\alpha) * (T_s/T_w)$, and $(1-\alpha) * (1 - T_s/T_w)$, respectively. Meanwhile, after *Match* or *Wait* when $fork = active$, the new honest block or weak header could be mined on either chains, leading to six outcome states. The probability will be influenced by γ ratio, for instance, the probability of an honest block mined on top of the attacker's chain is $\gamma * (1-\alpha) * T_s/T_w$.

D. Objective Function

At first, it should be noted that the hash power α of the attacker $\mathcal{A}$ must be less than 50%. Here, as the objective of attacker is to maximize its relative profit using selfish mining, the objective function is defined as:

$$R(\pi, \alpha) = \min_{\pi} \lim_{t \to \infty} \frac{\sum R_s}{\sum R_h + \sum R_s}, \qquad (1)$$

where $\sum R_s$ is the accumulated reward of the attacker and $\sum R_h$ is the honest miners' accumulated revenue. Let π be an attack policy of the attacker. Ideally, $R(\pi, \alpha) = \alpha$.