Flux: Revisiting Near Blocks for Proof-of-Work Blockchains

Alexei Zamyatin*,†, Nicholas Stifter†,‡, Philipp Schindler†, Edgar Weippl†, and William J. Knottenbelt*

* Centre for Cryptocurrency Research and Engineering (IC3RE), Imperial College London, United Kingdom
{a.zamyatin, wjk}@imperial.ac.uk
† SBA Research, Austria
{nstifter, pschindler, eweippl}@sba-research.org
‡ Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), TU Wien, Austria

***Abstract*—The term *near* or *weak* blocks describes Bitcoin blocks whose PoW does not meet the required target difficulty to be considered valid under the regular consensus rules of the protocol. Near blocks are generally associated with protocol improvement proposals striving towards shorter transaction confirmation times. Existing proposals assume miners will act rationally based solely on intrinsic incentives arising from the adoption of these changes, such as earlier detection of blockchain forks.**

In this paper we present *Flux*, a protocol extension for proof-of-work blockchains that leverages on near blocks, a new block reward distribution mechanism, and an improved branch selection policy to incentivize honest participation of miners. Our protocol reduces mining variance, improves the responsiveness of the underlying blockchain in terms of transaction processing, and can be deployed without conflicting modifications to the underlying base protocol as a velvet fork. We perform an initial analysis of selfish mining which suggests *Flux* not only provides security guarantees similar to pure Nakamoto consensus, but potentially renders selfish mining strategies less profitable.

I. Introduction

Attempting to deploy consensus rule changes in cryptocurrencies is often a controversial topic as it can introduce the possibility of permanent forks in the underlying blockchains. Hence, many of the parametrizations that characterize a cryptocurrency, such as the targeted block interval, total number of currency units, or maximum permissible block size, are considered to be virtually unchangeable in practice. However, a recently proposed protocol upgrade mechanism in Kiayias et al. [25], referred to as a *velvet fork*, is aimed at avoiding the possibility of permanent blockchain forks by rendering protocol changes only *conditionally* applying. That is, the changed rules of the protocol upgrade are not enforced by consensus participants through discarding legacy blocks, but are instead only selectively applied to blocks that are fully valid under the new protocol rules, while still also accepting all other legacy blocks as valid. Such an approach requires backward compatibility to be maintained and hence not all protocol changes can be deployed as a velvet fork [53]. Nevertheless, changes to some of the characteristic parametrizations of a cryptocurrency, such as the *block interval*, can still be rendered compatible with this new scheme.

In this paper we introduce *Flux*, a protocol extension for Bitcoin-like proof-of-work blockchains that builds upon the concept of velvet forks in order to reduce the targeted block interval. Conceptually, Flux is based on the structure of near (or weak) blocks and subchains [38], [39], the remunaration approach of the decentralized mining pool P2Pool [2] and technical aspects of merged mining [46], [9], [23]. The protocol extension is capable of reducing mining variance and allows miners to earlier detect conflicting blocks and forks. Furthermore, upgraded nodes can more quickly gain confidence that a transaction will be confirmed and included through the protocol's extended transaction processing mechanism and branch selection policy.

An initial simulation-based analysis shows Flux not only provides similar security guarantees as Bitcoin, but is potentially capable of improving the robustness of the system to selfish mining attacks. Additionally, the concept of the Flux protocol extension can serve as basis for a framework for conflict-free deployment of protocol extensions, i.e., without necessitating potentially problematic soft or hard forks. Thereby, compatible protocol updates can be introduced in a conditional manner, i.e., as velvet forks, such that legacy and upgraded miners accept the same set of blocks [53], [25].

The rest of this paper is structured as follows. In Section II we provide the necessary background information on Bitcoin and proof-of-work blockchains, as well as on the principles of proof-of-work reusal, near blocks, P2Pool and merged mining. Section III provides a detailed description of the Flux protocol, while a security analysis is given in Section IV. We discuss applications and give an outlook on future work in Section V, and conclude our paper in Section VI.

II. Background

In this section we provide the necessary background information on Bitcoin and comparable proof-of-work blockchains, as well as the general notion of near blocks and subchains. To gain an overview of the principles surrounding decentralized ledgers we point the interested reader towards literature such as [31], [51], [12], [33].

A. Bitcoin and Proof-of-Work Blockchains

Bitcoin was introduced as the first decentralized digital currency in 2008 by the pseudonymous Satoshi Nakamoto [31]. At its core lies a distributed peer-to-peer network where each participating node maintains the current state of the system, defined by the historic transaction record, and communication is facilitated by a gossip protocol. Transactions are consolidated and grouped together in blocks which in turn are chained

together via the hashes of their predecessors, forming a data structure referred to as the *blockchain*.

The set of consensus participants is dynamically changing and unknown[1], and agreement on the current state of the system is achieved through a mechanism referred to as *Nakamoto consensus* [12]. Hereby, participants are required to solve computationally intensive cryptographic puzzles, i.e., perform a so called Proof-of-Work (PoW), which establishes priced identities that both combats Sybil attacks [15] and also serves as a leader election mechanism for proposing state updates to the blockchain. The first consensus participant or *miner* to find a puzzle solution, such that it fulfills a pre-defined difficulty target, becomes leader and updates the state by appending a new block to the blockchain. The puzzle's input has to contain a reference to a previous block which it wants to append to and can contain new (valid) transactions, such that modifications to this input would invalidate the puzzle solution. As remuneration for the invested computational effort, miners are awarded new units of the underlying cryptocurrency, as well as fees collected from processed and included transactions. Consensus is formed by introducing a protocol rule that leaders only append their solutions to the head of the, to them known, chain with the most cumulative PoW, where all blocks are considered valid. Under the assumption that a majority of consensus participants is acting economically rational by following these protocol rules, agreement on a common prefix of the blockchain is eventually reached [18].

B. Sustainability and Future Outlook of Proof-of-Work

Proof-of-work is a well established concept in terms of providing security and immutability in blockchains, however its utilization is not without controversy. Although the resource heavy computations required by PoW increase the cost of attacks, they result in significant power consumption and raise the question of long-term sustainability [6], [35]. Furthermore, while initially any participant could successfully participate in the mining process, rising competition resulted in the creation of pooled mining which in turn has led to centralization where computational resources are bundled under the control of only a few actors [20], [23], [19]. The introduction of ASICs, i.e., hardware specifically constructed for performing proof-of-work computations for (Bitcoin) mining [50], further amplified this development. As a result, the majority of computational power in Bitcoin and other cryptocurrencies is concentrated in a small set of mining pools [10], [20], [23]

While efforts towards replacing the resource-intensive mining process have so far yielded various promising approaches [7], [26], [30], [54], their viability in practice is yet to be tested at a larger scale. Furthermore, due to the high degree of adoption of proof-of-work in various cryptocurrencies and the difficulties related to changing this consensus critical component, it can be assumed that PoW will remain an integral part of the overall cryptocurrency landscape in the foreseeable future.

[1]Pseudonymous identities of consensus participants are established once they find and successfully broadcast a valid puzzle solution.

C. Weak Blocks and Subchains

The general idea of reusing proof-of-work such that the computational effort invested may also serve to verify a separate computation was first introduced by Jakobsson and Juels under the term *bread pudding protocols* in 1999 [22]. The selected terminology points towards the main idea of the scheme: reuse computation by-products to minimize wasted resources. In the context of proof-of-work, this means to recycle unused or *stale* computations and utilize them as proof-of-work for other tasks.

The idea of *weak blocks* was initially proposed by TierNolan (pseudonym) in 2013 [38] and later extended in Rizun's *subchain* concept [39]. Weak blocks represent otherwise valid blocks, which do not meet the target difficulty d of the underlying cryptocurrency but satisfy some lower difficulty d_{weak}, i.e. $d_{weak} < d$. Instead of being discarded, these blocks can be reused and exchanged between miners to potentially reduce transaction confirmation times.

Weak blocks form so called subchains between consecutive full blocks by referencing the header of a preceding weak block in an additional pointer. As the difficulty target of weak blocks can, in principle, be chosen arbitrarily (only requirement is that $d_{weak} < d$), the interval between such blocks can be significantly lower than that of full blocks. This potentially allows faster (weak) transaction confirmations and can be of advantage for miners: by participating in building and validating subchains, miners can determine diverging blockchain branches, i.e., so called *forks*, earlier. As a result, they face a lower risk of investing computational effort on a blockchain branch, which will be later discarded.

Despite having seen active discussion [3], [4], [41], [40], [29], [43], as of today there exists no implementation of the weak blocks concept. One possible explanation for the absence of development in this area is the lack of incentives for miners to participate in building subchains. The intrinsic rewards in form of earlier fork detection may be insufficient when put in contrast to the possible overhead for participating miners in terms of computation[2], bandwidth and maintenance.

D. P2Pool

P2Pool [2] is a decentralized Bitcoin mining pool and was announced and launched in 2011. In contrast to conventional mining pools, P2Pool requires no operator to verify each miner's contribution to the mining operation of the pool. Instead, a network of peer-to-peer miner nodes is created parallel to Bitcoin and the proof-of-work of mining pool shares is reused for verification of each miner's contribution.

The key concept behind P2Pool is the so called *sharechain*. The sharechain is also a blockchain, which runs in parallel to Bitcoin but maintains a significantly lower difficulty $d_{share} < d_{BTC}$, targeting a block interval of 30 seconds. Sharechain blocks are structurally equivalent to Bitcoin blocks, however

[2]Not proof-of-work, but verification of transactions and additional consensus rules.

they only meet a difficulty target of d_{share} but not d_{BTC}, i.e., equate to shares submitted by miners to a mining pool.

Miners participating in P2Pool initially follow the normal mining process, as if solo mining. When building the block, a miner inserts her own payout address(es) in the outputs of the coinbase transaction and starts to search for solution candidates for the resulting PoW puzzle. However, in contrast to solo mining, P2Pool miners do not keep the complete block reward to themselves. Each time the miner finds a valid PoW solution where $d_{share} < d_{PoW} < d_{BTC}$, i.e., a valid share but not a full Bitcoin block, she publishes this block to the network of P2Pool miners. After the majority of the peers has verified that the block is valid, it is appended to the sharechain and all miners resume their search for the next Bitcoin block.

However, when building the preliminary block structure, miners now must include the payout address(es) of the previous sharechain block in the outputs of the coinbase transaction. Otherwise, P2Pool peers will discard the block as invalid and the respective miner will not be rewarded for the submitted share when a full block is found. Whenever any miner participating in the scheme finds a full Bitcoin block, she publishes it to the Bitcoin network. As a result, all miners who have submitted sharechain blocks during this round will receive a portion of the reward, directly distributed through the coinbase transaction of the block.

E. Bitcoin-NG

Bitcoin-NG, introduced by Eyal et al. [16], is an extension of Bitcoin aiming at improving latency and bandwidth consumption, while maintaining similar security properties. The protocol distinguishes between normally mined key blocks and microblocks, which require no PoW. One of the main differences to Bitcoin, however, is that key blocks are solely used for the leader election process, while the transaction processing is shifted to micro blocks. Each time a node is able to provide a valid PoW solution matching the require difficulty target and hence generate a key block, she is identified as the new *leader*. Consequently, she is allowed to create and sign microblocks at will, until the next leader is elected. The rest of the network can check that the signature provided with micro blocks belongs to the current leader and hence accept the corresponding state updates.

To incentivise honest behavior, the block rewards and transaction fees are split in a 40:60 ratio between each two consecutive leaders. However, since leaders can decide on the transactions included in the chain during their ruling period, they can attempt censor transactions or to perform double spending attacks. To this end, Bitcoin-NG makes use of fraud proofs or so called *poison transactions*, which can be used to prove malicious behavior of a leader to the rest of the network (i.e., the following leaders) and as a result invalidate potential revenue gains of an attacker.

III. The Flux Protocol Extension

In this section we provide a comprehensive overview of the Flux protocol and it's differences to Bitcoin's current implementation. Note that while we use Bitcoin as reference in the rest of this paper for simplification, Flux can in principle be implemented on top of any Bitcoin-like proof-of-work cryptocurrency, such as Bitcoin Cash [1], Litecoin [28], Namecoin [32] and Zcash [45]. Flux imposes no conflicting alteration to Bitcoin's consensus mechanism, but instead acts as an optional protocol update, i.e., all modifications and additional rules described in the following sections can be deployed as a *velvet fork* [25], [53].

Following TierNolan's concept of weak blocks [38], extended upon by Rizun [39], the core idea of Flux is to encourage miners to publish near miss PoW solutions as *sub-blocks*, forming so called *sub-chains* between each two consecutive Bitcoin blocks (c.f. Figure 1). For the sake of clarity, Bitcoin blocks in their current form are referred to as *legacy blocks*, while blocks matching the difficulty target $d_k = d$ in Flux are denoted as *key blocks* in the rest of this paper. Just like in a mining pool, the block rewards and transaction fees are distributed among miners according to their contribution to solving the PoW of each key block, measured by submitted sub-blocks. To allow quicker block intervals in sub-chains and reduce incentives for deliberate forks, we introduce the *heaviest chain* branch selection policy, by weighing key and sub-block differently, based on the difficulty relation d_k/d_s.

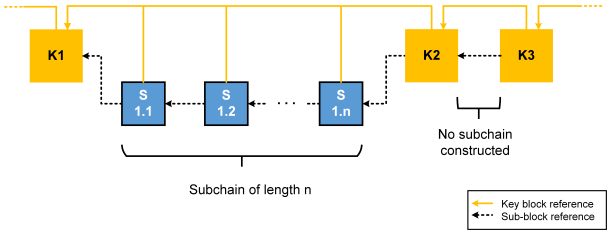


Fig. 1. Simplified visualization of subchains (optionally) constructed from sub-blocks S between consecutive key-blocks K.

A. Key Blocks

Key blocks are backward compatible to Bitcoin's legacy blocks in terms of size, structure and functionality. The requirement for a hash based proof-of-work over the block header also remains the same. The required difficulty target d_k is chosen such that a block is found approximately every 10 minutes and is adjusted according to the mining power present in the network every 2016 blocks. As legacy blocks, key blocks are responsible for consolidating the transactions defining the current state of the blockchain and distributing the block reward and transactions fees.

However, unlike in Bitcoin, each key block contains an additional reference to the previous sub-block in the blockchain [3]. Furthermore, the distribution of the block reward and transactions fees are now part of the block validation process. In contrast to the current Bitcoin implementation, they no longer are paid solely to the miner of the key block. Instead, the

[3]Can be stored using the `OP_RETURN` opcode in Bitcoin [8].

remuneration is split between the key block miner and the miners of sub-blocks included in the subchain between the current and the previous key block. The coinbase transaction of each key block hence must contain outputs transferring a part of the newly minted coins and earned fees to the respective sub-block miners. We describe the new reward distribution policy in more detail in Section III-E. Note that neither the additional block reference, nor the adjusted reward distribution mechanism affect the consensus rules of the underlying Bitcoin blockchain, i.e., legacy miners remain oblivious to these changes.

B. Sub-blocks

Sub-blocks represent otherwise valid key blocks, which do not meet difficulty target d_k but some predefined lower target $d_s < d_k$ for the PoW and hence maintain a lower block interval. When trying to find a key block, miners iterate over billions of possible solutions until one is found that matches the difficulty requirements d_k of a key block. Instead of discarding weak solutions, miners broadcast these "byproducts" if a predefined minimal difficulty target d_s is met in form of sub-blocks, thereby publicly declaring on which key block they are currently mining on.

Sub-blocks are integrated into the Bitcoin blockchain by creating subchains between consecutive key blocks. Just like key blocks, sub-blocks contain an additional reference to the previous sub-block, the 256-bit hash of the previous sub-block header. If no sub-block is present, i.e., at the start of a subchain, the reference points to the previous key block.

The minimal difficulty target d_s required for sub-blocks defines the interval at which the latter are generated by the network. A higher generation frequency of sub-blocks leads to a lower mining variance, quicker fork detection and better responsiveness to transactions. However, this comes at the cost of a higher accidental fork rate, lowering the security properties of sub-block confirmations. We note that the identification of optimal parameters requires detailed game theoretic analyses and will be subject of future work. Furthermore, alternative and more complex difficulty adjustment mechanisms, such as adjustment after every block or based on simulated annealing [52], may provide improved robustness against adversarial behavior.

C. Transaction Processing

Since the structure of key- and sub-blocks is the same as of legacy Bitcoin blocks, the current predefined maximum block size in Bitcoin applies to both. As such, each sub-block and specifically all sub-blocks in a subchain in total cannot contain more transactions than the succeeding key block.

Taking into account the high amount of pending Bitcoin transactions at peak times (ranging from a few thousand to 100.000+ at the time of writing [11]), the first sub-block in a subchain will very likely already contain all transactions of the subchain and the following key block respectively. As a result, in a naive implementation, succeeding sub-blocks would only serve as additional *sub-confirmations* to already included transactions.

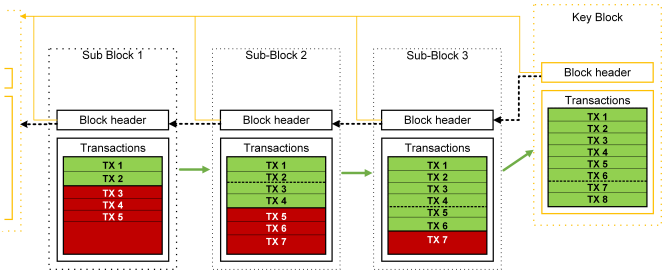


Fig. 2. Simplified visualization the transaction processing mechanism in Flux. Sub-blocks split transactions into *confirmed* (green) and *unconfirmed* (red) sets. Subsequent sub-blocks copy previously confirmed transactions.

However, some use cases may require not only fast transaction confirmations but also the ability to process and confirm multiple dependent transactions within the 10 minute block interval. This is already possible in Bitcoin by spending from an unconfirmed UTXO (e.g. used in "Child-pays-for-parent" (CPFP [49] transactions), but confirmations from sub-blocks may be able to provide stronger security guarantees.

Hence, we propose to allow miners to split transactions of sub-blocks into two sets: *confirmed* and *unconfirmed*. Confirmed transactions represent a small subset of all transactions initially included in the block and, once included in a sub-block, receive a sub-confirmation. The set of all sub-confirmed transactions in a subchain will receive a full confirmation through the next key block. In total, the number of such sub-confirmed transactions cannot exceed the bound set by the maximum size of a key block. A simplified visualization is provided in Figure 2.

As a result, clients can reference outputs of such confirmed transactions as input for transactions in subsequent sub-blocks. Ignored transactions on the other hand receive no confirmations, neither through sub- nor through key blocks, until they appear in the confirmed transaction set. Clearly, the risk of double spending still persists and high value transactions should be performed under recommended confirmation times. However, the higher interval of sub-blocks allows clients to select more fine grained confirmation thresholds before accepting a payment.

D. Mining

The mining process in Flux can be described as follows: If a miner finds a solution to Bitcoin's PoW puzzle satisfying d_k, she creates a key block and broadcasts it to the network. However, upon finding a solution missing d_k but satisfying the weaker target d_s the miner now creates a sub-block. In order to claim remuneration, the miner must add a new output to the coinbase transaction, which moves a predefined portion of the revenue to his address. Thereby, a miner must differentiate between four cases as described in the following. For simplification, we do not consider forks in this section and provide a separate discussion in Section III-F.

a) Start of a new subchain: If the mined block is the first sub-block in a new subchain, the reference to the previous sub-block must point to the key block the subchain is building on instead. To claim remuneration, the miner follows the same pattern as with Bitcoin legacy blocks: she adds an output to the coinbase transaction transferring fund to his address. However, she can only claim a part of the block reward and transaction fees, predefined by the reward distribution policy. The later is validated by all other Flux nodes, which will discard the sub-block in case of violations.

b) Sub-block in existing subchain: If the mined block is a sub-block and there already exists a subchain, the miner includes a reference to the previous sub-block. To claim rewards, the miner copies all existing outputs from the coinbase transaction of the previous block, thereby re-calculating the reward distribution. Only then can she append a new output transferring her share of the remuneration to a chosen address.

c) Key block found: In case the found block fulfills the requirements to become the next key block, the miner finalizes the subchain, if present. This is accomplished by referencing the last sub-block of the subchain and including all outputs from its coinbase transaction in the coinbase transaction of the key block. Similar to the previous case, the miner must re-calculate the reward distribution prior to adding her own address to the reward payouts.

d) Last block was a legacy block: In a scenario where not all miners have adopted Flux, it is possible that the previous N blocks do not adhere to the described new protocol rules. Hence, the miner will include a reference to the last key or sub-block adhering to Flux rules, creating a skiplist of new and legacy blocks. If the last block adhering to Flux rules was a key block, the miner follows the standard transaction processing mechanism as described in Section III-C. Otherwise, any remaining unconfirmed transaction in the pending subchain *must be included* in the next key or sub-block. Only then may the miner include new transactions from the mempool to fill up the remaining space in the block. This guarantees that once a transaction was included in a sub-block, it will be at some point included in a key block, except if the sub-block is pruned or a conflicting transaction has already been accepted into the chain earlier.

We note, however, that most transactions contained in the previous sub-block will have already been included in the following legacy blocks. A simplified visualization is provided in Figure 3.

Note that each miner, both of sub- and key blocks, can specify multiple payout addresses to receive the reward shares, i.e., split up the reward among multiple receiving addresses, as long as the revenue distribution remains compliant with the rules described above.

E. Remuneration Model

Flux makes use of an alternative reward distribution model that can reduce the variance of mining, in particular for smaller miners or mining poolsd. Instead of the "*the winner takes it all*" approach implemented in most PoW blockchains, the

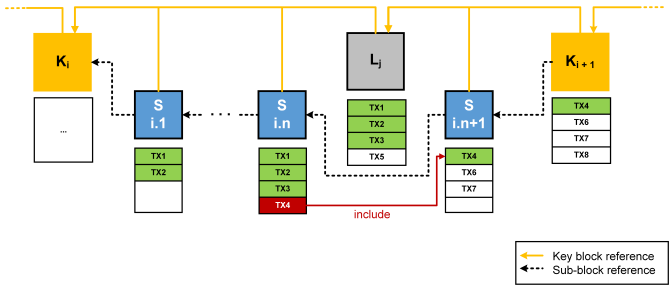


Fig. 3. Skiplist of key blocks K, sub-blocks S and legacy blocks L. Key and sub-blocks always have a reference to the last block adhering to the Flux protocol rules. Transactions remaining unconfirmed by intermediate legacy blocks must be included in subsequent key/sub-blocks (enforced by protocol rules).

block reward and transaction fees are distributed among all miners who have contributed to the latest key block, similar to the approach used by mining pools. Thereby, we make use of an approach initially introduced in P2Pool [2]: the hash rate of each miner is estimated using the number of sub-blocks submitted to the subchain[4].

Hence, the expected revenue $\mathbb{E}[R_i]$ of a miner i can be calculated as follows:

$$\mathbb{E}[R_i] = \mathbb{E}[R_k^i] + \mathbb{E}[R_s^i] \tag{1}$$

Thereby, $\mathbb{E}[R_k^i]$ is the expected key block revenue of miner i, defined as

$$\mathbb{E}[R_k^i] = \lambda_k \cdot \frac{R_b}{\mathbb{E}[s]+1} = \frac{h_i}{d_k} \cdot \frac{R_b}{\mathbb{E}[s]+1} \tag{2}$$

where $\mathbb{E}[s] = \frac{d_k}{d_s}$ is the expected number of sub-blocks per subchain, defined by the difficulty relation between key and sub-blocks, and R_b is the revenue generated by a Bitcoin block, i.e., block reward and transaction fees. The revenue a miner i expects to earn from submitting sub-blocks to a subchain can in turn be calculated as follows:

$$\mathbb{E}[R_s^i] = \lambda_s \cdot \frac{\mathbb{E}[s_i]R_b}{\mathbb{E}[s]+1} = \frac{h_i}{d_s} \cdot \frac{\mathbb{E}[s_i]R_b}{\mathbb{E}[s]+1} \tag{3}$$

where $\mathbb{E}[s_i]$ represents the number of sub-blocks a miner i with hash rate h_i is expected to find in the time period t_k between two consecutive key blocks. This number can be calculated as

$$\mathbb{E}[s_i] = \lambda_s t_k = \frac{h_i d_k}{d_s H} \tag{4}$$

where H denotes the combined hash rate of all miners in the network adhering to Flux protocol rules. By applying substitution rules to Equation 1 using Equations 2 and 3 we receive:

$$\mathbb{E}[R_i] = h_i R_b \cdot \left(\frac{1}{\frac{1}{d_s} + d_k} + \frac{h_i d_k}{H(d_s + d_k)} \right) \tag{5}$$

[4]In the terminology used to describe mining pools, each sub-block would represent a *share* submitted to the pool's operator.

We note that transaction fees are only rewarded for transactions included in a key block. This is specifically relevant, in case a sub-chain is interrupted by one or more legacy blocks, since a subset of the transactions included in a subchain may be ceded to legacy miners (c.f. Figure 3).

The new reward distribution mechanism not only provides an incentive to publish sub-blocks, but also reduces the variance of participating miners, as if they were to join a mining pool. The higher the adoption of Flux, the lower the variance of income for upgraded miners over the same time frame.

We note, however, that each sub-block appended to a subchain reduces the reward attributed to miners of previous sub-blocks. As a result, miners having submitted sub-blocks to a subchain may have an incentive to prevent new sub-blocks from being included in the chain, e.g. by ceasing to forward or forking away sub-blocks. Alternatively, at some point miners may no longer see extending a subchain as profitable and decide to turn off their equipment/switch to an alternative PoW blockchain, until the next key block is found. While the new branch branch selection policy described in Section III-F helps mitigate deliberate forking of sub-blocks, a thorough game theoretic analysis is necessary to predict the behavior of economically rational[5] participants. This, however, goes beyond the scope of this paper and will be subject for future work.

F. Mitigating Forks: Heaviest Chain Rule

If two or more different blocks are created at the same height, i.e., referencing the same preceding block, the blockchain is split into so called *forks*. Typically, this can happen if multiple miners find a block around the same time [6] or a malicious miner attempts to cut off an existing block, i.e., purposely chooses not to reference the preceding block. As a result, miners will add blocks to any of the existing branches, causing the network to diverge into multiple states, until the conflict is resolved. Branches and blocks which were not accepted into the blockchain as result of such a fork are referred to as *pruned*[7].

In Bitcoin, accidental forks were measured to happen every 60 blocks on average [14]. The underlying protocol prescribes that in the presence of multiple valid conflicting branches, miners must extend the chain which contains the most accumulated proof-of-work. Thereby, the required PoW difficulty per block is taken into account, rather than the actually performed work per block. Originally, ties were resolved by selecting the first branch the miner was aware of, however, to counteract the effectiveness of selfish mining branch selection for blocks at the same height is now performed randomly [17]. Given the PoW difficulty in Bitcoin is constant every 2016 blocks, the above rule is reduced to the so called *longest chain rule*.

[5]i.e., ready to deviate from the protocol to increase revenue.

[6]Neither are aware of the existence of the other miner's block.

[7]While the term *orpahn blocks* is often used in this context, it is misleading since these blocks actually have a parent in the blockchain, as pointed out in [16]

In Flux, sub-blocks are created at a higher frequency than key blocks, hence yielding the probability of accidental forks in subchains significantly higher than in the Bitcoin main chain. In addition, the different PoW thresholds of key and sub-blocks require an approach to handle forks consisting of different typed blocks. To this end, we propose to implement the *heaviest chain rule* in Flux, i.e., explicitly evaluate the accumulated PoW "weight" of a branch including both its key and sub-blocks. Thereby, the weight of each block is determined by its difficulty target, i.e., d_k for key blocks and d_s for sub-blocks. The resulting weight w of a branch of k key blocks is hence calculated as

$$w = \sum_{k}^{i=0}(d_k + \sum_{s_i}^{j=0} d_s) \tag{6}$$

where s_i denotes the length of the subchain prepending a key block i. As a result, if a branch consisting of a single key block conflicts with a branch of sub-blocks accumulating a greater weight for the same height, i.e. $\sum d_s > d_k$, the key block will be discarded. The described modification to the branch selection policy potentially also provides improved robustness against selfish mining attacks, as discussed in Section IV.pre

Note, however, that to maintain full compatibility to the base protocol rules, Flux miners must differentiate between legacy and upgraded key and sub-blocks. As such, if a legacy block conflicts with a key block prepended by a subchain, the heaviest chain rule is *not applied*, i.e., the weight of the sub-blocks is not considered. Otherwise, Flux miners would constantly fork legacy blocks and at some point, namely at around 25% Flux adoption[8], a permanent split of the blockchain would occur.

IV. Security Analysis

In this section we evaluate the security properties of the Flux protocol and contrast these to the security guarantees provided by the current Bitcoin implementation.

A. Selfish Mining

Selfish mining attacks were shown to allow an adversary to increase their relative revenue by intentionally withholding blocks and secretly attempting to mine a chain that outperforms the one of the rest of the network [17], [44], [21], [34]. Thereby, the attacker will only occasionally reveal a selected number of blocks from her secret chain, depending on the concrete strategy, forcing honest miners to discard their progress and reorganize their view of the public blockchain. The success probability of such adversarial strategies depends on the fraction of computational power controlled by the attacker α and her network connectivity γ. The latter parameter defines the probability that the majority of honest miners will accept the attacker's fork in case of a tie in terms of performed PoW.

[8]The exact number depends on the weight relation between key and sub-blocks.

1) Flux Specific Attacks: Since miners following the Flux protocol differentiate between key and sub-blocks and take into account the different PoW requirements, the notion of selfish mining attacks is slightly different, when compared to Bitcoin. As such, we identify the following three base attack strategies, defined by when the adversary starts mining on a secret chain:

- *Always*: As described for Bitcoin in previous research work, a straightforward approach is to perform attacks always, i.e., without requiring the occurrences of specific conditions. When following this strategy, an adversary will constantly attempt to override the public chain.
- *Key block triggered*: Taking into account the heaviest chain rule in Flux, an adversary may attempt to exploit the different weighing of key and sub-blocks. Since key blocks are attributed a significantly higher PoW weight, a viable strategy is to launch attacks only after finding a key block, while remaining honest otherwise.
- *Minimal revenue*: Attack only after having found the first N sub-blocks in a (public) subchain, i.e., knowing that a minimal revenue is guaranteed, even if the attack should fail.

B. Double Spending

Double-spending spending attacks were one of the first studied security problems in proof-of-work blockchains [5], [42], [24], [37]. While the original Bitcoin client implementation identified transactions as confirmed only after they have received at least 6 confirmations, the time until acceptance of a transaction varies from merchant to merchant and is often dependent on the transferred value. As such, even zero-confirmation transactions are accepted for payments considered infeasible to double-spend.

Since sub-blocks require a lower PoW difficulty target than key blocks, the block interval in sub-chains is expected to be significantly lower than that of the main chain. As a consequence, instead of trusting transactions with zero confirmations participants can wait for the respective transaction to be included in a sub-block, as pointed out by Rizun [39]. While sub-block confirmations clearly provide lower security guarantees than normal block confirmations, performing a double-spending attack on such a transaction would require to explicitly fork the subchain. Mainly, however, subchains allow the definition of finer grained thresholds for security waiting periods in terms of PoW weight. As such, merchants can require N sub-block confirmations for payments, instead of relying on zero-confirmation transactions for small payments. The transaction processing mechanism described in Section III-C hereby guarantees transactions confirmed in sub-blocks will eventually be accepted into the blockchain[9] even if ignored/not seen by legacy miners.

C. Simulation Model

Sapirstein et al. [44], Nayak et al. [34] and Gervais et al. [21] performed detailed analyses of selfish mining strategies using Markov Decision Processes (MDP). Due to the complexity of our model, arising from the variable weighting of blocks and the new reward distribution policy, i.e., sub-blocks are not paid out instantly but must be tracked until the next key block is found on the main chain, we opted for a discrete event based simulation model for our initial analysis of selfish mining under the Flux protocol extension. Furthermore, since Markov Chains, which lie at the core of MDPs, are memory-less, each state is defined by a set of variables and depends only on the previous state and the transition between them. However, since key and sub-blocks are attributed different PoW weights in Flux, it is necessary to keep track of the entire blockchain when modeling the adversaries' decision process. Specifically, if a MDP were to be used, it would not be possible to construct more complex strategies known to outperform standard selfish mining, such as *stubborn mining* [34], where an attacker inter alia can decide to reveal only a limited number of blocks instead of always publishing the entire secret chain. However, for the interested reader we provide an initial MDP model of Flux, simplified to cover standard selfish mining strategies only, i.e., the attacker will always publish her entire secret chain when overriding the public chain, in Appendix A.

Following the approach of previous research, we differentiate between two actors in our simulation model, namely an adversary with hash rate α and the rest of the network assumed to be honest with hash rate $\beta = 1-\alpha$. However, honest miners are in turn split into two categories: *legacy* miners with hash rate ϵ and *upgraded* miners following Flux protocol rules with hash rate $\beta-\epsilon$. While honest miners always adhere to protocol rules, the adversary may choose to withhold blocks, attempting to create a secret chain, and to override the public chain when in lead.

Blocks are sampled randomly from exponential distributions with rate parameters defined by the hash rate portions of each actor and the difficulty relation between key and sub-blocks, assuming a constant PoW difficulty for the duration of the simulation. When defining the transition probabilities we must distinguish between key and sub-blocks, as these have different requirements in terms of PoW. We denote $\alpha_s = \alpha \cdot \frac{d_s}{d_k}$ as the probability of the attacker finding a sub-block and $\alpha_k = \alpha-\alpha_s$ as the probability of the attacker finding a key block. The probability that honest miners find a sub-block shall be defined as $\beta_s = (\beta-\epsilon)\cdot\frac{d_s}{d_k}$ and $\beta_k = \beta-\beta_s$ shall denote the probability of honest miners finding a key block. A summary is provided in Table IV-C. For simplicity, we set $d_s = 1$ and normalize d_k based on the relation d_s/d_k: E.g., for $d_s/d_k = 0.1$ we receive $d_k = 10$.

D. Simulating Selfish Mining

We now continue to present the results sampled from our simulations. In a first step, for comparison purposes and to validate our simulator with results generated by MDPs in previous work, we simulate the original selfish mining strategy in Bitcoin without the Flux protocol extension. As depicted in Figure 4, the received results correspond with the observations made by Sapirshtein et al. in [44].

[9] As long as no chain reorganization occurs.

Table I. Summary of state transition probabilities.

α	Percentage of the overall hash rate controlled by the attacker /probability of the attacker finding a block
$\alpha_s = \alpha \cdot \frac{d_s}{d_k}$	Attacker finds a sub-block
$\alpha_k = \alpha - \alpha_s$	Attacker finds a key-block
ϵ	Probability of a legacy miner finding a (key) block
$\beta = 1 - \alpha$	Probability of an honest miner finding a block
$\beta_s = (\beta - \epsilon) \cdot \frac{d_s}{d_k}$	Honest miners find a sub-block
$\beta_k = \beta - \beta_s$	Honest miners find a key-block
γ	Probability that honest miners extend attacker's fork during conflict resolution

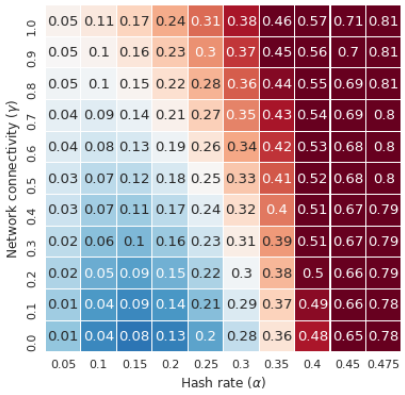


Fig. 4. Simulation results for selfish mining in Bitcoin as described by Eyal and Sirer [17], without the Flux protocol extension (provided for comparison).

Next, we simulate Eyal and Sirer's original selfish mining strategy applied to sub-blocks, i.e., *override* and *match* actions take place when the attacker find both a key or sub-block. To evaluate the impact of different adoption rates of our protocol extension, we selectively assumed 0%, 25% and 50% of the hash rate being controlled by legacy miners, ignoring Flux protocol rules. As visualized in Figure 5, our analysis suggests selfish mining attacks as described by Eyal and Sirer [17] are less effective under the deployment of the Flux protocol extension, than under pure Bitcoin consensus rules. We observe that lower block intervals in subchains slightly reduce the effectiveness of selfish mining for adversaries controlling up to 40% of the overall hash rate. Note, however, that these evaluations do not yet take into account possible network layer effects. Applying these results to the current mining power distribution in Bitcoin, an adversary with currently realistic hash rate portion of 30%[10] and $\gamma = 0.5$ will generate between 25% less revenue for subchains of expected length 10 and and 47% for length 50 respectively.

We further simulate *key block triggered* attacks, specific to the Flux protocol extension and present the results in Figure 6. Thereby, we observe a significant improvement in terms of effectiveness for small and medium attackers. Interestingly, the impact of the network connectivity is noticeably lower than with standard selfish mining. This is explained by the fact that the adversary performs much less attack attempts, namely only when she has the advantages of being a "heavy" key block ahead of the honest miners.

An exemplary comparison of the relative revenue generated by selfish mining in Bitcoin with ($1 = \beta + \alpha, \epsilon = 0$) and without ($1 = \epsilon + \alpha$) adoption of Flux protocol rules for $\gamma = 0$ is provided in Table II.

E. Simulating Stubborn Mining Strategies

Nayak et al. have previously shown selfish mining is outperformed by so called *stubborn mining* strategies, where the parameters of the *adopt* and *override* conditions, as well as the amount of blocks revealed from the secret chain are altered. Hence, to extend our evaluation of selfish mining under the protocol updates introduced by Flux, we simulate approximately 234.000 stubborn mining strategies with varying parametrization of the adversary's decision process. As explained before, using a finite state MDP for stubborn mining under variable block weights is not feasible. Hence, due to the large problem space of this optimization problem, for an initial analysis in this paper we only simulate the effectiveness of stubborn mining strategies for attackers with 25%, 30% and 47.5% of the overall hash rate, assuming $\mathbb{E}(s) = 10$, $\epsilon = 0$ and $\gamma = 0$.

In our simulation model, each *stubborn mining* strategy is defined by the *attack trigger* (at) (c.f. Section IV-A1), the *adopt* (ad) and *override* (o) conditions, as well as the *published weight* (p) from the secret chain when overriding the public chain. To correctly model the parameters of the attacker's decision process, we must first introduce two new variables:

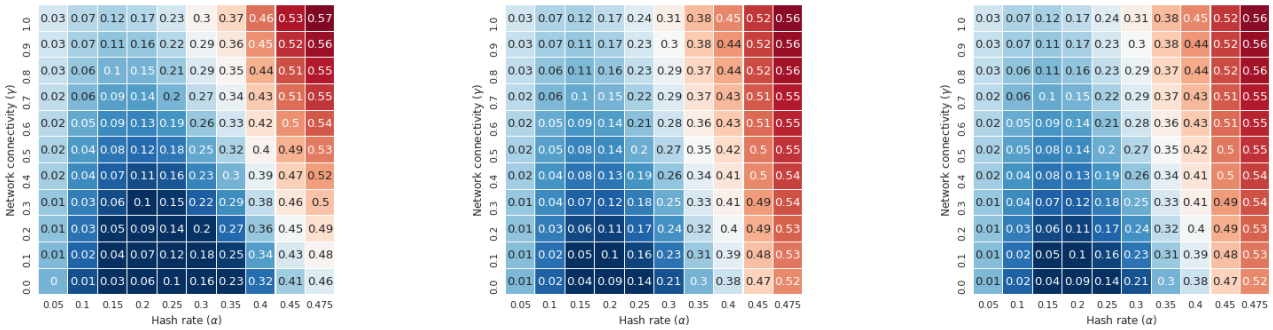
- $\mathbf{w_a}$ - Proof-of-work "weight" of the attacker's secret chain, i.e., the sum of products of sub- and key-block PoW difficulties: $w_a = \sum_{i=0}^{k_a} d_k + \sum_{i=0}^{s_a} d_s$, where k_a is the number of key blocks and s_a is the number of sub-blocks present in the attacker's secret chain.
- $\mathbf{w_h}$ - Proof-of-work "weight" of the honest miner's public chain. Analogous to the attacker's chain, $w_h = \sum_{i=0}^{k_h} d_k + \sum_{i=0}^{s_h} d_s$, where k_h is the number of key blocks and s_h is the number of sub-blocks present in the public chain.

The attack strategy parameters are hence defined as follows:

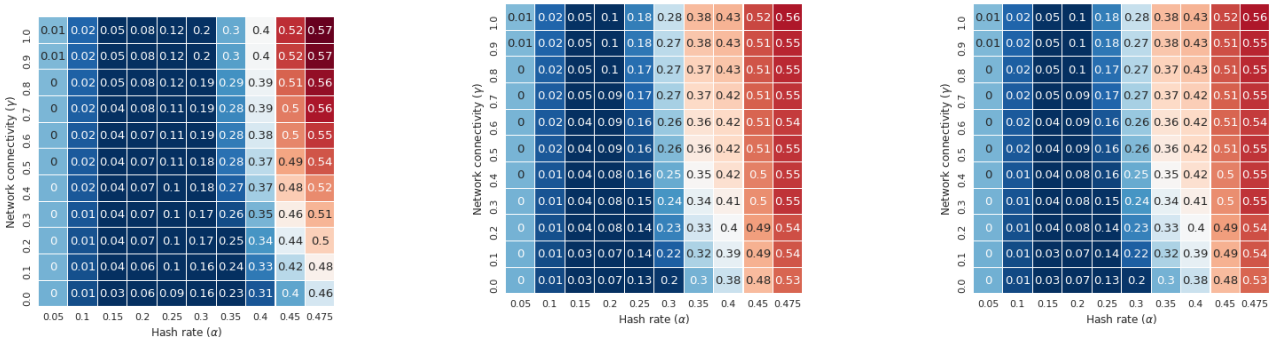
- *Adopt condition* (ad): Determines how far (in terms of PoW weight) an attacker can fall behind until she adopts the public chain, i.e., $w_a \leq w_h - ad$. In our simulation, we test for $ad \in [0, .., 6 \cdot d_k]$.
- *Override condition* (o): Defines how long the attacker waits to reveal the secret chain and override the public chain [11], i.e., how close the gap between w_a and w_h is allowed to become: $w_a > w_h + o$, tested for $o \in [0, ..., 6 \cdot d_k]$.
- *Published weight* (p): Determines how much of the secret chain the attacker is willing to publish, when overriding

[10]Using the block attribution technique introduced in [23], we measured the largest mining pool in Bitcoin constitutes approximately 30% of the overall mining power between February and March 2018

[11]Note that overriding is only feasible if $w_a \geq w_h$.



(a) $\epsilon = 0$, $\mathbb{E}(s) = 10$ (b) $\epsilon = 0.25$, $\mathbb{E}(s) = 10$ (c) $\epsilon = 0.50$, $\mathbb{E}(s) = 10$



(d) $\epsilon = 0$, $\mathbb{E}(s) = 50$ (e) $\epsilon = 0.25$, $\mathbb{E}(s) = 50$ (f) $\epsilon = 0.50$, $\mathbb{E}(s) = 50$

Fig. 5. Visualization of the relative revenue of an attacker following the improved sub-block level version of **standard selfish mining** strategy[17] for different α and γ values, simulated for 100.000 blocks for expected subchain length of 10. Results provided for 0% (a, d), 25% (b, e) and 50% (c, f) legacy miner presence. Red coloring indicates revenue gain, blue coloring indicates loss when compared to the expected revenue, defined by α.

Table II. Effectiveness of selfish mining in Bitcoin before (SM1) and after the Flux protocol extension, exemplary for $\gamma = 0$ and $\epsilon = 0$.

α	SM1 in Bitcoin [17], [44]	Sub-block level selfish mining			Key block triggered selfish mining		
		$\mathbb{E}(s) = 10$	$\mathbb{E}(s) = 50$	Avg. difference in %	$\mathbb{E}(s) = 10$	$\mathbb{E}(s) = 50$	Avg. difference in %
0.30	0.30	0.16	0.16	-46.7	0.29	0.29	-3.3
0.35	0.37	0.23	0.23	-37.8	0.35	0.35	-5.4
0.40	0.42	0.32	0.31	-25.0	0.41	0.4	-3.6
0.45	0.65	0.41	0.4	-37.7	0.48	0.48	-26.2
0.475	0.78	0.46	0.46	-41.0	0.52	0.5	-34.6

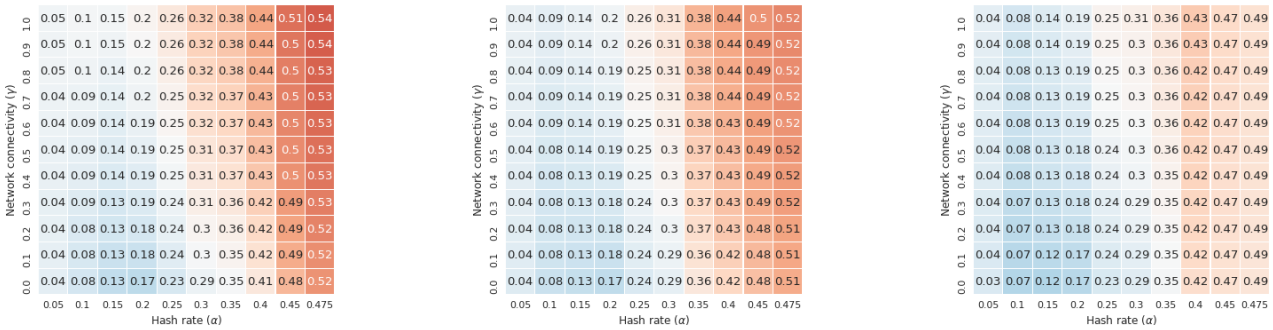
the public chain, tested for $p \in [1, ..., 2 \cdot d_k, all]$. Note that this value must be viewed as a "target", as in some constellations the order of key and sub-blocks in the secret chain may require the attacker to publish more weight than defined by p.

Thereby, the values used as upper and lower bounds for the parametrization of the tested strategies are thereby selected based the results and discussion of existing research work [17], [44], [34]. Each performed simulation run again consists of 100.000 blocks.

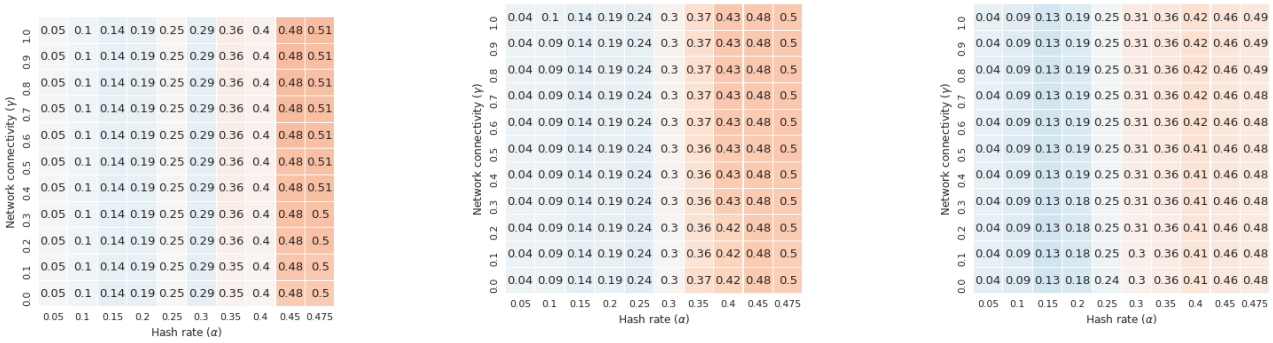
A first observation is the varying profitability of attack triggers, depending on the attacker's hash rate. As such, an attacker with 47.5% of the overall computational power achieves the highest revenue gains by constantly attempting to perform attacks. However, a miner with 30% can achieve near-optimal[12] following a more careful and less detectable strategy, such as key block triggered selfish mining. As such, the best observed stubborn mining strategy for an attacker with 25% hash rate is based on the *minimal revenue* attack trigger.

A visualization of the effects of the attack parameters ad, o, and p on the relative revenue of the attacker are presented in Figure 7. As can be seen, a miner controlling a significant portion of the overall hash rate is incentivised to perform stubborn mining attacks, i.e., continue attacking even when falling behind (*trail stubborn mining*) and wait longer for the honest miners to catch up before overriding the public chain (*lead stubborn mining*), as already pointed out in [34]. The weight published from the secret, however, does not seem to significantly impact the effectiveness of attacks.

[12]Optimal when compared to the simulated problem space. The existence of even more effective strategies cannot be excluded.



(a) $\epsilon = 0$, $\mathbb{E}(s) = 10$ (b) $\epsilon = 0.25$, $\mathbb{E}(s) = 10$ (c) $\epsilon = 0.50$, $\mathbb{E}(s) = 10$



(d) $\epsilon = 0$, $\mathbb{E}(s) = 50$ (e) $\epsilon = 0.25$, $\mathbb{E}(s) = 50$ (f) $\epsilon = 0.50$, $\mathbb{E}(s) = 50$

Fig. 6. Visualization of the relative revenue of an attacker following the **key-block triggered** selfish mining strategy (c.f. Section IV-A1) for different α and γ values, simulated for 100.000 blocks for expected subchain length of 50. Results provided for 0% (a, d), 25% (b, e) and 50% (c, f) legacy miner presence. Red coloring indicates revenue gain, blue coloring indicates loss when compared to the expected revenue, defined by α.

In Table III we summarize the most effective selfish/stubborn mining strategies measured during our simulations and compare them to Eyal and Sirer's selfish mining strategy (SM1) [17], as well as Sapirshtein et al.'s optimtimal strategy (ϵ-OPT) [44]. A first observation is the notable effect of the Flux adoption rate on the feasibility of attacks. For 100% adoption, selfish/stubborn mining is less effective than currently in Bitcoin, especially for attackers controlling significant portions of the computational power. Interestingly, under the presence of legacy miners controlling a significant portion of the overall hash rate, an adversary following Flux protocol rules appears to outperform selfish mining strategies in Bitcoin. We note, however, that we have so far not acquired exact numerical values for Nayak et al.'s stubborn mining strategies in Bitcoin. The later, however, have been shown to outperform standard selfish mining by up to 25%. Hence, we expect the comparison of stubborn mining in Bitcoin with and without Flux protocol rules to be similar as the observations made for standard selfish mining.

Summarizing, we have performed a first analysis of the effectiveness of selfish and stubborn mining strategies under the Flux protocol extension, which suggest that the introduction of subchains with an alternative reward distribution scheme and the heaviest chain branch selection policy could contribute towards improving the robustness of Bitcoin against adversarial strategies.

V. Outlook and Future Work

Up until now we have primarily outlined Flux as a protocol upgrade mechanism to reduce the block interval of Bitcoin-like cryptocurrencies. However, the velvet fork concept and general framework for protocol updates outlined in Flux is not necessarily limited to this particular goal. In [48] Sompolinsky et al. propose a new *Greedy Heaviest-Observed Sub-Tree* (GHOST) algorithm for selecting what is considered the best chain by (honest) miners upon which they will mine new blocks. Such a mechanism is in particular beneficial to high block creation rates, however Sompolinsky et al. outline that increasing this rate in Bitcoin would require a hard fork in the protocol and therefore the acceptance of a majority of the mining power. We have shown that the Flux protocol allows decreasing the block interval without necessitating hard or soft forks, and hence the question of whether GHOST or other protocol improvement proposals could also be deployed in this manner is naturally raised. A modified variant of GHOST has been successfully implemented in Ethereum [13], thereby introducing rewards for pruned blocks. Hence, instead of simply relying on Bitcoin's longest-chain rule, a variant of GHOST similar to that of Ethereum could be implemented in

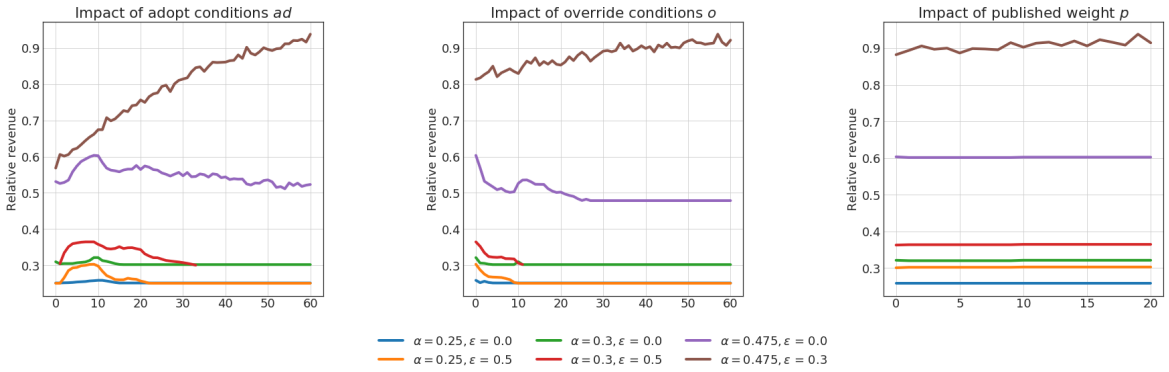


Fig. 7. Visualization of the correlation between different adopt conditions ad (a), override conditions o (b) and, published weights p (c) and attacker's revenue for 100% and 50% Flux adoption ($\epsilon = 0$ and $\epsilon = 0.5$ respectively). Results sampled from simulations of 100.000 blocks each.

Table III. Summary of best performing selfish/stubborn mining strategies sampled from simulations of 100.000 blocks each with $\gamma = 0$ for different Flux adoption rates, compared to existing selfish mining strategies in Bitcoin.

Attacker's Hash Rate	Legacy Hash Rate	Revenue	Flux Strategy Parametrization				Bitcoin Selfish Mining	
			ad	at	o	p	SM1 [17]	ϵ-OPT [44]
0.25	0	0.25188	10.0	minimal revenue	0.0	1-20, ALL	⋆	⋆
	0.5	0.30268	9.0	always	0.0	10-20	-	-
0.3	0	0.32114	9.0	always	0.0	10-20	0.3	0.33705
	0.5	0.36448	9.0	always	0.0	10-20	-	-
0.475	0	0.60337	9.0	always	0.0	ALL	0.78254	0.80172
	0.3†	0.93796	60.0	always	57.0	19	-	

⋆These selfish mining strategies are only considered feasible with hash rate portions beyond 30% in Bitcoin.
†We simulate attacks performed by a 47.5% Flux selfish miner only under 30% legacy miner hash rate, as using 50% would assume the very unlikely case where nearly all Flux miners are a single malicious entity.

the subchain, requiring miners to take into account pruned sub-blocks when resolving conflicts between auxiliary branches.

Other proof-of-work-based blockchain proposals, such as DAG-based approaches [27], [47] or modifications intended to increase the *chain quality* and discourage selfish mining such as Fruitchains [36], may also be adaptable such that it could be deployed in a manner similar to Flux. In this respect we outline that there remain many open questions regarding possible reward distribution schemes and ensuring incentive-compatibility of both velvet forks and Flux-like protocol upgrades.

VI. Conclusion

In this paper we presented Flux, a velvet fork extension for proof-of-work blockchains, which is based on the concepts of near blocks and sub-chains and an improved new branch selection policy. Flux can facilitate faster and better estimates if particular transactions will be eventually included in the main chain through its transaction processing mechanism. Furthermore, Flux introduces an alternative reward distribution mechanism, thereby reducing the payout variance of participating miners compared to the legacy protocol. We have performed an initial simulation analysis of selfish mining, the results of which suggest Flux is capable of yielding selfish and stubborn mining strategies less feasible.

VII. Acknowledgments

We would like to thank Iain Stewart, Georg Merzdovnik, Sam Werner, Arthur Gervais, Sjors Provoost, Dominik Harz and especially Aljosha Judmayer for helpful comments and insightful discussions.

References

[1] Bitcoin Cash. https://www.bitcoincash.org/. Accessed: 2017-01-24.
[2] P2pool. http://p2pool.org/. Accessed: 2017-05-10.
[3] G. Andersen. Comment in "faster blocks vs bigger blocks". https://bitcointalk.org/index.php?topic=673415.msg7658481#msg7658481, 2014. Accessed: 2017-05-10.
[4] G. Andersen. [bitcoin-dev] weak block thoughts... https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011157.html, 2015. Accessed: 2017-05-10.
[5] E. Androulaki, S. Capkun, and G. O. Karame. Two bitcoins at the price of one? double-spending attacks on fast payments in bitcoin. In *CCS*, 2012.
[6] J. Becker, D. Breuker, T. Heide, J. Holler, H. P. Rauer, and R. Böhme. Can we afford integrity by proof-of-work? scenarios inspired by the bitcoin currency. In *WEIS*. Springer, 2012.
[7] I. Bentov, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. https://eprint.iacr.org/2016/919.pdf, 2016. Accessed: 2016-11-08.
[8] Bitcoin community. OP_RETURN. https://en.bitcoin.it/wiki/OP_RETURN. Accessed: 2017-05-10.
[9] Bitcoin Wiki. Merged mining specification. https://en.bitcoin.it/wiki/Merged_mining_specification. Accessed: 2017-05-10.
[10] Blockchain.info. Hashrate Distribution in Bitcoin. https://blockchain.info/de/pools. Accessed: 2017-05-10.
[11] Blockchain.info. Unconfirmed bitcoin transactions. https://blockchain.info/unconfirmed-transactions. Accessed: 2017-05-10.

[12] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *IEEE Symposium on Security and Privacy*, 2015.

[13] V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper, 2014. Accessed: 2016-08-22.

[14] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[15] J. R. Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[16] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Security Symposium on Networked Systems Design and Implementation (NSDI'16)*. USENIX Association, Mar 2016.

[17] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[18] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*, pages 281–310. Springer, 2015.

[19] A. E. Gencer, S. Basu, I. Eyal, R. Renesse, and E. G. Sirer. Decentralization in bitcoin and ethereum networks. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC). Springer*, 2018.

[20] A. Gervais, G. Karame, S. Capkun, and V. Capkun. Is bitcoin a decentralized currency? volume 12, pages 54–60, 2014.

[21] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun. On the security and performance of proof of work blockchains. https://eprint.iacr.org/2016/555.pdf, 2016. Accessed: 2016-08-10.

[22] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Secure Information Networks*, pages 258–272. Springer, 1999.

[23] A. Judmayer, A. Zamyatin, N. Stifter, A. G. Voyiatzis, and E. Weippl. Merged mining: Curse or cure? In *CBT'17: Proceedings of the International Workshop on Cryptocurrencies and Blockchain Technology*, Sep 2017.

[24] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun. Misbehavior in bitcoin: A study of double-spending and accountability. volume 18, page 2. ACM, 2015.

[25] A. Kiayias, A. Miller, and D. Zindros. Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963, 2017. Accessed:2017-10-03.

[26] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[27] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. Inclusive block chain protocols. In *Financial Cryptography and Data Security*, pages 528–547. Springer, 2015.

[28] Litecoin community. Litecoin reference implementation. https://github.com/litecoin-project/litecoin. Accessed: 2018-05-03.

[29] G. Maxwell. Comment in "[bitcoin-dev] weak block thoughts...". https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011198.html, 2016. Accessed: 2017-05-10.

[30] S. Micali. Algorand: The efficient and democratic ledger. http://arxiv.org/abs/1607.01341, 2016. Accessed: 2017-02-09.

[31] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, Dec 2008. Accessed: 2015-07-01.

[32] Namecoin community. Namecoin reference implementation. https://github.com/namecoin/namecoin. Accessed: 2017-05-10.

[33] Narayanan, Arvind and Bonneau, Joseph and Felten, Edward and Miller, Andrew and Goldfeder, Steven. Bitcoin and cryptocurrency technologies. https://d28rh4a8wq0iu5.cloudfront.net/bitcointech/readings/princeton_bitcoin_book.pdf?a=1, 2016. Accessed: 2016-03-29.

[34] K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *1st IEEE European Symposium on Security and Privacy, 2016*. IEEE, 2016.

[35] K. J. O'Dwyer and D. Malone. Bitcoin mining and its energy footprint. 2014.

[36] R. Pass and E. Shi. Fruitchains: A fair blockchain. http://eprint.iacr.org/2016/916.pdf, 2016. Accessed: 2016-11-08.

[37] C. Pérez-Solà, S. Delgado-Segura, G. Navarro-Arribas, and J. Herrera-Joancomartí. Double-spending prevention for bitcoin zero-confirmation transactions. http://eprint.iacr.org/2017/394, 2017. Accessed: 2017-06-29.

[38] Pseudonymous("TierNolan"). Decoupling transactions and pow. https://bitcointalk.org/index.php?topic=179598.0, 2013. Accessed: 2017-05-10.

[39] P. R. Rizun. Subchains: A technique to scale bitcoin and improve the user experience. *Ledger*, 1:38–52, 2016.

[40] K. Rosenbaum. Weak blocks - the good and the bad. http://popeller.io/index.php/2016/01/19/weak-blocks-the-good-and-the-bad/, 2016. Accessed: 2017-05-10.

[41] K. Rosenbaum and R. Russell. Iblt and weak block propagation performance. *Scaling Bitcoin Hong Kong (6 December 2015)*, 2015.

[42] M. Rosenfeld. Analysis of hashrate-based double spending. http://arxiv.org/abs/1402.2009, 2014. Accessed: 2016-03-09.

[43] R. Russel. Weak block simulator for bitcoin. https://github.com/rustyrussell/weak-blocks, 2014. Accessed: 2017-05-10.

[44] A. Sapirshtein, Y. Sompolinsky, and A. Zohar. Optimal selfish mining strategies in bitcoin. http://arxiv.org/pdf/1507.06183.pdf, 2015. Accessed: 2016-08-22.

[45] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[46] Satoshi Nakamoto. Comment in "bitdns and generalizing bitcoin" bitcointalk thread. https://bitcointalk.org/index.php?topic=1790.msg28696#msg28696. Accessed: 2017-06-05.

[47] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. Spectre: A fast and scalable cryptocurrency protocol. Cryptology ePrint Archive, Report 2016/1159, 2016. Accessed: 2017-02-20.

[48] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security*, pages 507–527. Springer, 2015.

[49] Suhas Daftuar. Bitcoin merge commit: "mining: Select transactions using feerate-with-ancestors". https://github.com/bitcoin/bitcoin/pull/7600. Accessed: 2017-05-10.

[50] M. B. Taylor. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, page 16. IEEE Press, 2013.

[51] F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. In *IEEE Communications Surveys Tutorials*, volume PP, pages 1–1, 2016.

[52] P. J. Van Laarhoven and E. H. Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[53] A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottebelt. (Short Paper) A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 18 (FC). Springer*, 2018.

[54] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Renesse. Rem: Resource-efficient mining for blockchains. http://eprint.iacr.org/2017/179, 2017. Accessed: 2017-03-24.

APPENDIX A
MDP-BASED SECURITY MODELING

In this section we present some initial results of the work-in-progress modeling effort to represent selfish mining attacks using a Markov Decision Process (MDP), similar to [44], [34], [21]. State transitions occur each time a key or a sub-block is found, according to the probabilities defined in Table IV-C.

A. Notation

Each state is defined by the tuple $(w_a, w_h, u_a, k_a, w_k, fork, last)$, consisting of the following variables:

- $\mathbf{w_a}$ - Proof-of-work "weight" of the attacker's secret chain, i.e., the sum of products of sub- and key-block PoW difficulties: $w_a = \sum_{i=0}^{k_a} d_k + \sum_{i=0}^{s_a} d_s$, where k_a is the number of key blocks and s_a is the number of sub-blocks present in the attacker's secret chain.

Table IV. The transition and reward matrices for selfish mining attacks under Flux protocol rules. We use · instead of *fork* and *last* to express that in the current state the value of the respective is irrelevant or remains unchanged. For simplicity, we only track the relative reward earned by the attacker.

State x Action	Resulting State	Probability	Reward (in block reward)
$(w_a, w_h, u_a, k_a, w_k, \cdot, \cdot), adopt$	$(d_s, 0, 0, 0, 0, irrelevant, sub)$	α_s	$\left(\frac{u_a}{(u_a+w_h)}, \frac{w_h}{(u_a+w_h)}\right)$
	$(d_k, 0, 0, 1, d_k, irrelevant, key)$	α_k	$\left(\frac{u_a}{(u_a+w_h)}, \frac{w_h}{(u_a+w_h)}\right)$
	$(0, d_s, 0, 0, 0, irrelevant, last)$	β_s	$\left(\frac{u_a}{(u_a+w_h)}, \frac{w_h}{(u_a+w_h)}\right)$
	$(0, d_k, 0, 0, 0, irrelevant, last)$	β_k	$\left(\frac{u_a}{(u_a+w_h)}, \frac{w_h}{(u_a+w_h)}\right)$
$(w_a, w_h, u_a, k_a, w_k, \cdot, sub), override^{\dagger}$	$(d_s, 0, u_a + max(w_a - w_k, w_a), 0, 0, irrelevant, sub)$	α_s	$(k_a, 0)$
	$(d_k, 0, 0, 1, w_a + d_k, irrelevant, key)$	α_k	$\left(k_a + \frac{u_a}{(u_a+w_h)}, 0\right)$
	$(0, d_s, max(w_a - w_k, w_a), 0, 0, relevant, \cdot)$	β_s	$(k_a, 0)$
	$(0, d_k, 0, 0, 0, relevant, \cdot)$	β_k	$\left(k_a + \frac{u_a}{(u_a+w_h)}, 0\right)$
$(w_a, w_h, u_a, k_a, w_k, \cdot, key), override^{\dagger}$	$(d_s, 0, 0, 0, 0, irrelevant, sub)$	α_s	$(k_a, 0)$
	$(d_k, 0, 0, 1, d_k, irrelevant, key)$	α_k	$(k_a, 0)$
	$(0, d_s, 0, 0, 0, relevant, \cdot)$	β_s	$(k_a, 0)$
	$(0, d_k, 0, 0, 0, relevant, \cdot)$	β_k	$(k_a, 0)$
$(w_a, w_h, u_a, k_a, w_k, irrelevant, \cdot), wait$	$(w_a + d_s, w_h, u_a, k_a, w_k, irrelevant, sub)$	α_s	$(0, 0)$
	$(w_a + d_k, w_h, u_a, k_a + 1, w_a + d_k, irrelevant, key)$	α_k	$(0, 0)$
$(w_a, w_h, u_a, k_a, w_k, relevant, \cdot), wait$	$(w_a, w_h + d_s, u_a, k_a, w_k, relevant, \cdot)$	β_s	$(0, 0)$
	$(w_a, w_h + d_k, u_a, k_a, w_k, relevant, \cdot)$	β_k	$(0, 0)$
$(w_a, w_h, u_a, k_a, w_k, active, \cdot), wait$	$(w_a + d_s, w_h, u_a, k_a, w_k, active, sub)$	α_s	$(0, 0)$
	$(w_a + d_k, w_h, u_a, k_a + 1, w_a + d_k, active, key)$	α_k	$(0, 0)$
$(w_a, w_h, u_a, k_a, w_k, relevant, \cdot), match^{\ddagger}$	$(w_a - w_h, d_s, max(w_a - w_k, w_a), k_a, w_k, relevant, \cdot)$	$\beta_s \cdot \gamma$	$(k_a, 0)$
	$(w_a, w_h + d_s, u_a, k_a, w_k, relevant, \cdot)$	$\beta_s \cdot (1-\gamma)$	$(0, 0)$
	$(w_a - w_h, d_k, 0, k_a, w_k, relevant, \cdot)$	$\beta_k \cdot \gamma$	$\left(k_a + \frac{u_a}{(u_a+w_h)}, 0\right)$
	$(w_a, w_h + d_k, u_a, k_a, w_k, relevant, \cdot)$	$\beta_k \cdot (1-\gamma)$	$(0, 0)$

†Only feasible if $w_a > w_h$
‡Only feasible if $w_a \geq w_h$

- $\mathbf{w_h}$ - Proof-of-work "weight" of the honest miner's public chain. Analogous to the attacker's chain, $w_h = \sum_{i=0}^{k_h} d_k + \sum_{i=0}^{s_h} d_s$, where k_h is the number of key blocks and s_h is the number of sub-blocks present in the public chain.
- **fork** - Provides information on the previously mined block and hence, relevant for the attacker's decision process. Can be one of:
  - *irrelevant* - The last block was found by the attacker, hence *match* is not feasible
  - *relevant* - The last block was found by the honest network, *match* is feasible if $w_a \geq w_h$.
  - *active* - The attacker has performed a *match* action and a publicly visible fork is ongoing. Depending on γ, the honest network will either extend the public chain or the attacker's now public fork.
- **last** - Defines whether the last block found by the attacker was a key or a sub-block. This is relevant for the attacker's decision process, as owning a key-block on the private chain provides a significant advantage above the honest miners (*key block triggered* attack).
  - *sub* - denotes that the last block found by the attacker was a sub-block
  - *key* - denotes that the last block found by the attacker was a key-block
- $\mathbf{u_a}$ - Unpaid sub-blocks in terms of PoW (invertible) of the attacker. Since rewards are only distributed once a key block is found and are dependent on the number of sub-blocks in a subchain (i.e., we cannot predict the exact reward until a key block closes a subchain), it is necessary to track how much PoW/ how many sub-blocks of the attacker's secret chain have already been accepted by the honest miners and hence will be rewarded.
- $\mathbf{k_a}$ - Number of key block in the attacker's secret chain.
- $\mathbf{w_k}$ - Proof-of-work weight of the part of the attacker's chain, which will be rewarded instantly in case of a successful *override*, i.e., payouts already confirmed in key blocks. Both k_a and w_k are necessary to determine how much reward the attacker will receive instantly and how much will be paid later, as only knowing the number of key blocks gives no information on the actual PoW weight of the chain, and vice versa.

1) Actions: Adversary Alice may perform the following actions when finding a block:

- **Adopt** - Always feasible. The attacker discards her secret chain and adopts the public chain of the honest miners, receiving no rewards.
- **Override** - Only feasible if $w_a > w_h$. The attacker publishes her secret chain and triggers a chain reorganization, after which the honest miners will accept the secret chain. Depending if the attacker performed this action after finding a key or sub-block, rewards are either paid instantly, or delayed to the next key block on the now public chain.
- **Wait** - Always feasible. The attacker ignores the public chain and continues to mine blocks on her secret chain.
- **Match** - Only feasible if $w_a \geq w_h$. After the honest

miners have found a key or sub-block, the attacker publishes a block of the same PoW weight for the same height. Note: the attacker must have had the block ready *before* the triggering event on the public chain. Otherwise, the the honest miners will have accepted the block on the public chain before the attacker can react.

2) Restrictions: In order to model the selfish mining attack problem under variable block weights (which implicitly affects the paid rewards) using a MDP, we must introduce limitations to the state space, given that Markov Chains cannot easily capture the order of occurred events, i.e., the ordering of key and sub-blocks:

- When performing an *override* action, the attacker always published her entire secret chain.

A possible workaround would be to store the type of the last N blocks in additional variables. However, as the state space increases exponentially with each additional tracked variable, we do not consider this a feasible approach. As such, it is unfortunately not feasible to model more complex adversarial strategies, such as unfortunately *stubborn mining* [34] using MDPs under variable block weights.