On the deployment of FlyClient as a velvet fork: *chain-sewing* attacks and countermeasures

Tristan Nemoz[1,2,3] and Alexei Zamyatin[1]

[1] Imperial College London, London, United Kingdom
{tristan.nemoz19,alexei.zamyatin17}@imperial.ac.uk
[2] Télécom Paris, Palaiseau, France
[3] EURECOM, Biot, France

Abstract. Because of the everlasting need of space to store even the headers of a blockchain, Ethereum requiring for example more than 4 GiB for such a task, *superlight* clients stood out as a necessity, for instance to enable deployment on wearable devices or smart contracts. Among them is FlyClient [2], whose main benefit was to be non-interactive. However, it is still to be shown how a such protocol can be deployed on an already existing chain, without contentious soft or hard forks. FlyClient suggests the use of velvet forks [5,15], a recently introduced mechanism for conflict-free deployment of blockchain consensus upgrades – yet the impact on the security of the light client protocol remains unclear.
In this work, we provide a comprehensive analysis of the security of FlyClient under a velvet fork deployment. We discover that a naive velvet fork implementation exposes FlyClient to *chain-sewing attacks*, a novel type of attack, concurrently observed in similar superlight clients [6]. Specifically, we show how an adversary subverting only a small fraction of the hash rate or consensus participants can not only execute double-spending attacks against velvet FlyClient nodes, but also print fake coins – with high probability of success. We then present three potential mitigations to this attack and prove their security both under velvet and, more traditional soft and hard fork deployment. In particular, our mitigations do not necessarily require a majority of honest, up-to-date miners.

Keywords: Blockchain · Superlight clients · FlyClient · Velvet forks · Cross-chain communication · Chain-sewing.

1 Introduction

FlyClient [2] is a *superlight* client designed in 2019 whose goal is to overcome the flaws in the Superblock NIPoPoW protocol [5]. As such, the FlyClient protocol aims to prove to a client which hasn't access to a given blockchain that a certain transaction lies within this chain, just like an SPV client [10]. Unlike an SPV client however, a *superlight* client only requires a logarithmic number of blocks with respect to the length of the chain to perform this.

FlyClient has been a major breakthrough within the blockchain community, and there are now plethora of protocols that builds upon, or try to improve

it [7,9,13,14,8]. FlyClient has shown especially useful in the case of cross-chain communication where the client could be for instance a Smart Contract, whose storage and number of performed operations must be as low as possible to avoid excessive fees when verifying a transaction. This enthusiasm for FlyClient's usefulness on this topic is most likely motivated by the fact that cross-chain communication now involves several millions of dollars [12], not to mention the recent rise of cryptocurrencies value [11], together with the hype for them, leading to the need for clients like smartphones with low storage to handle multiple chains at once.

In order to prove that a given block is included within a chain, FlyClient requires an additional piece of data to be stored within blocks: the interlink data. Hence, deploying FlyClient on an already existing chain implies that a protocol change strategy has to be chosen. The very fact that FlyClient induces a protocol change via a fork is considered by numerous to be its main drawback. This explains in part for instance the need to build upon it to build the HawkClient [9] or why the current BTC-Relay implementation only considers SPV clients for now [4]. While the original FlyClient paper describes how the protocol would have to be deployed via contentious soft or hard forks, the authors also suggest that FlyClient can also be deployed via a backward compatible velvet fork. A velvet fork [15] is a blockchain consensus upgrade strategy which does not require a majority of miners to adopt the new protocol, unlike soft or hard forks: blocks mined by both upgraded and non-upgraded miners are ensured to be accepted by consensus. However, FlyClient's authors provide neither a detailed velvet fork implementation, nor do they prove the security of their velvet fork implementation proposal.

In this paper, we emphasize the fact that the secure deployment of FlyClient under a velvet fork requires to consider the dangers arising from this method. In particular, we show that FlyClient, were it to be naively deployed on a velvet fork as indicated by its authors, is subject to so-called *chain-sewing* attacks [6]. A *chain-sewing* attack may arise whenever a protocol using an interlink data, such as FlyClient or NIPoPoW, is deployed on a velvet fork, which gives an adversary the possibility to include malicious data to fool the client. We describe how such attack strategies can be executed by adversaries controlling less than half of the total computational power in Proof-of-Work blockchains such as Bitcoin, and succeed with very high probability. Not only do chain-sewing attacks in this setting allow to double-spend coins, but also allow the adversary to convince a velvet FlyClient client to accept blocks and transactions invalid under the chain's consensus rules – ultimately breaking the assumption of light clients that the longest ("main") chain will always contain valid blocks. We describe different strategies that an adversary can perform to improve their probability of success while minimizing their cost. Furthermore, we propose modifications to the FlyClient protocol to make it resilient against these attacks and discuss the incurred cost overhead compared to the original FlyClient implementation. In particular, we discuss countermeasures allowing to release the assumption of a majority of honest, up-to-date miners. Finally, we discuss the relevance of

implementing FlyClient as a velvet fork, considering the drawbacks this method suffers from.

We present in section 2 a quick introduction to FlyClient and velvet forks, which is the minimum knowledge to understand the attack. In section 3, we explain how FlyClient can be subject to a *chain-sewing* attack when deployed as a velvet fork. We provide in section 4 an analysis of the attack, highlighting how much does it cost to an adversary to set this attack up in average and how long do they have to wait to fool the client. Countermeasures to this attack are highlighted in section 5. Finally, section 6 concludes this paper.

2 FlyClient and velvet forks background

2.1 FlyClient

Presentation of FlyClient FlyClient [2] is a superlight client [5,2] proposed in 2019. As such, its goal is to prove to a client which hasn't access to a blockchain C that a transaction TX lies within this chain, just like an SPV client [10]. Unlike an SPV client however, FlyClient is a superlight client, which means that it only requires a logarithmic number of blocks with respect to the length of the chain to prove this inclusion.

However, since the client does not ask for every block in the chain, it cannot recursively check that every Proof of Work (PoW) that forms the chain is valid. Hence, an adversary with less than half of the total hashrate can try to have the client accepting a transaction which lies in a block that isn't on the main chain. Such a situation is depicted on Figure 1. Note that since the client verifies the node claiming to have the heaviest chain first, that is the longest one in the case of a constant difficulty protocol, and then completely trusts this node, the adversary has to include fake blocks in their fork in order to have a chain which is as heavy as the main chain for their proof to be considered.
FlyClient's goal is to sample at least one invalid block to prove that the attacker is dishonest. In order to do this, FlyClient's authors determined the optimal sampling strategy that must be used by the client. We won't present in this section what this optimal strategy is. Still, a very important property is to be denoted: the optimal sampling strategy samples more recent blocks with higher probability.

Merkle Mountain Ranges In order for the client to know that a sampled block is indeed included within the chain, the FlyClient protocol extensively uses Merkle Mountain Ranges (MMR). An example of a MMR with 11 leaves is given in Figure 2.

The construction of an MMR won't be described in this paper. What is important to notice however is the following:

- **Given the MMR root, the client can efficiently verify the inclusion of a block within the chain** with a similar method to the one used with

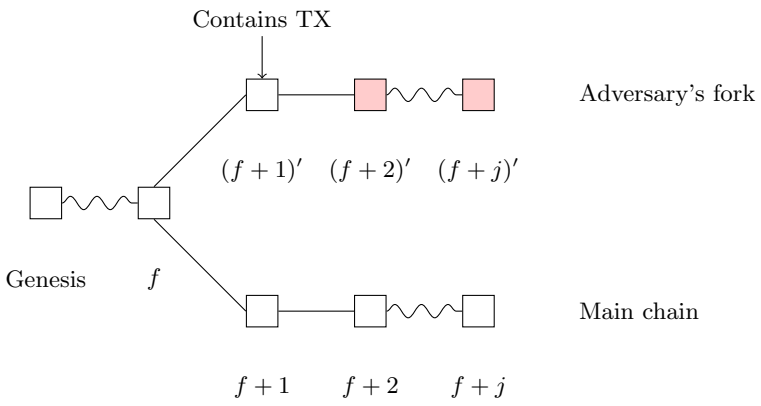

Fig. 1. FlyClient use case. Curved lines represent one or more blocks. Red blocks represent invalid blocks. The attacker forks the chain starting from block f and tries to have the client accept a transaction which lies within the block at height $f+1$. However, since the attacker owns less than half of the total hashrate, they are forced to include fake blocks in their fork to have a chain as long as the main chain. If they don't, their chain would be smaller than the main chain, which will lead in the other chain being verified and accepted before they can submit their proofs.

Merkle Trees: the prover provides the client with the path along the MMR to reach the MMR root. Lying on a block eventually requires the prover to find another pre-image to the MMR root, which is believed to be computationally infeasible, H being a cryptographic hash function.

- **The root of the MMR which only contains the first k blocks of the chain can be verified to be consistent with the one of the MMR containing every block in the chain.** This ensures that the adversary cannot simply put in their chain a precomputed block with a valid PoW, since its interlink data would be found to be inconsistent with the provided MMR root. Furthermore, it is easy for a node, given the MMR root and the inclusion proof of the k-th block, to compute the root of the MMR containing the first $k-1$ blocks.
- **It is easy for a node to maintain an MMR that represents their local chain**, since adding a leaf to an MMR is computationally easy.

To put it in a nutshell, the FlyClient protocol operates this way:

1. The client asks the prover for the last block of their chain, which contains the root of the MMR containing every block preceding the last one, along with the length of their chain. If both provers don't claim having the same chain length, the longer chain is verified first, as described in the Superblock NiPoPow paper [5].
2. The client asks the prover for a number of blocks logarithmic with respect to the length of the chain according to the optimal sampling strategy described in the original FlyClient paper [2]. Every block is provided to the client along

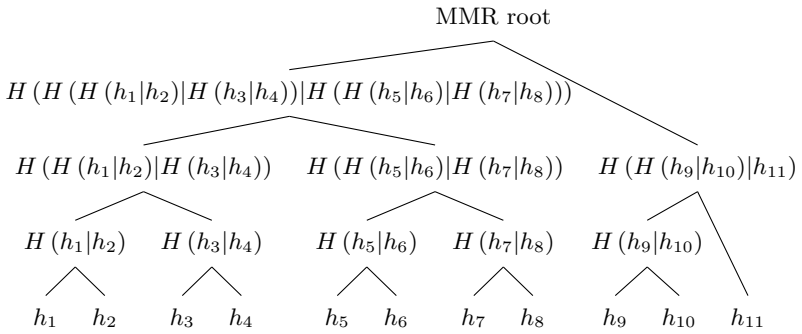

Fig. 2. A MMR with 11 leaves. h_i are block hashes and H is a cryptographic hash function like SHA-256.

with an MMR proof of inclusion, that is the path along the MMR to reach the root.

3. If every sampled block is valid, the client trusts this prover and does not consider other proofs. Otherwise, it checks the following prover.

Note that this assumes a model where at least one prover is honest, which is the one used in the original FlyClient paper [2]. Furthermore, the protocol described in the FlyClient original paper [2] is non-interactive. This does not change our analysis however.

2.2 Velvet forks

Velvet forks are an uncontentious way to deploy a new protocol on top of an old one [15]. Let us consider a situation where an arbitrary (potentially small) portion of miners are updated. Besides, let us assume that the new protocol reduces the set of valid blocks, just like a soft fork. Hence, a block mined by an updated miner is considered valid by a non-updated miner. The principle of a velvet fork is the following: every non-updated miner continues to mine on top on the main chain, just like they used to. Every updated miner mines blocks accordingly to the new protocol on top on the main chain, regardless of whether previous blocks are updated or not. This is shown on Figure 3.

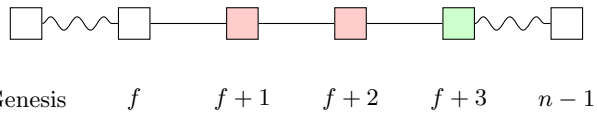

Fig. 3. Example of how a velvet fork works in practice. Red blocks represent invalid blocks with respect to the new protocol, green ones represent valid blocks with respect to the new protocol. The velvet fork is launched at height f.

Hence, whatever the percentage of updated miners is, a velvet fork allows the protocol to be deployed, assuming that the fact that invalid blocks are present in the chain. From now on, we call invalid blocks with respect to the new protocol legacy blocks. However, it may be necessary for the new protocol to be adapted, so it considers the fact that invalid blocks with respect to this new protocol are now part of the main chain.

2.3 FlyClient under a velvet fork deployment

According to the FlyClient original paper: "For FlyClient the velvet fork would lead to a constant fraction α of blocks containing an MMR root. Blocks created by outdated nodes would not contain the root. The FlyClient protocol would simply treat multiple blocks as one. Concretely, blocks that do not contain an MMR root are viewed as part of the next upgraded block. The miner will always download and check these joined blocks together." [2]

Hence, for FlyClient, deploying the protocol as a velvet fork is tackled by considering legacy blocks as part of valid blocks. What this means is that if a legacy block is sampled, then the prover will provide the client with every following block until a block mined by an updated miner is found. This is shown on Figure 4.

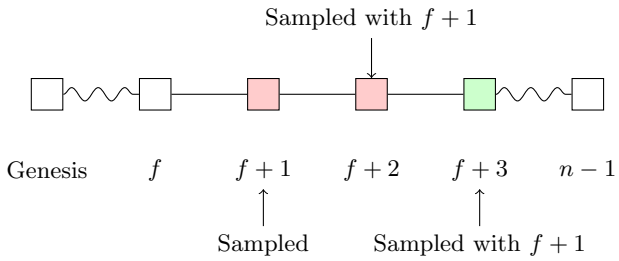

Fig. 4. Example of what happens when a legacy block is sampled. Block at height $f+1$ contains no interlink data, while block at height $f+2$ contains invalid interlink data. The prover will provide the client with blocks from $f+1$ to $f+3$ along with an MMR proof of inclusion for block at height $f+3$.

A crucial point is that a prover can declare any block as legacy. Indeed, from a non-updated miner's point of view, updated blocks containing a valid interlink piece of data and a block containing an arbitrary piece of data in the interlink field is indistinguishable. As such, both these kinds of blocks are accepted within the main chain.

Now, in order to show why this capability is essential, let us consider a flawed protocol not allowing the provers to declare blocks containing data as legacy. A very simple attack for an adversary would be to mine a block containing random data in the interlink field. If this block is sampled during an honest prover's proof,

they would be unable to provide an MMR proof of inclusion for this block, since its interlink data is not consistent with the honest prover's MMR root. Since it contains data, they cannot declare it as legacy either and thus are unable to provide the client with a valid proof. Hence, mining a single block on the main chain allows the adversary to completely break the protocol, since even honest miners can't provide the client with valid proofs.

To put it in a nutshell, it is mandatory for the protocol's sake to allow any prover to declare any block as legacy. When they do so, they have to provide the client with any subsequent blocks until one for which they are able to provide a valid MMR proof of inclusion.

Finally, it is also important to remark that detecting that a prover is dishonest is not enough for the protocol to be sane. One may think that preventing any transaction to be considered when a prover is dishonest is enough, but in the situation where the honest prover wants a transaction to be verified while the adversary doesn't want to, not considering this transaction actually benefits the adversary. Hence, the client is to determine which prover is dishonest before taking any decision concerning the transaction to be verified.

3 *Chain-sewing* attack on FlyClient

The principle of a *chain-sewing* attack has been firstly described in [6]. The principle is to take advantage of the fact that every miner can put arbitrary data in the interlink field to fool the miner. The goal of the adversary running this attack is to have a potentially invalid transaction accepted by the client, assuming the client uses the FlyClient protocol.

3.1 Models

Provers model We work with the two-provers case, as described in the original FlyClient paper: a client asks two provers for a proof of inclusion of a transaction within the chain. While the original model considered by FlyClient's authors considered that at least one prover was honest, we will assume that exactly one prover is honest, the other one being dishonest. Indeed, if both provers are honest, they will agree on the fact that the transaction is present within the main chain, and the random sampling won't have to be performed. Hence, we assume that both provers don't agree on whether the transaction is included within the chain, which means that at least one of them is dishonest. Since we still assume that at least one prover is honest, it leads to one honest prover and one dishonest prover.

Distributed Ledgers model Just like the original FlyClient paper [2], we put our analysis under the Bitcoin Backbone model [3]. Specifically, the mining process is represented as a memoryless process, where each player queries a random oracle H until they get a value such N such that $H(N) < T$, where T is the current difficulty of the mining process. Throughout our analysis, we will

assume T to be constant, that is, we work in the constant difficulty setup. This assumption is released in subsection 4.5, where we justify that our attacks still work in the variable difficulty setup.

Since we can model the mining process by a memoryless process, that we work in the two-provers model and that a given player having a computational power of μ mines by definition a ratio μ of the blocks, we model the mining process in rounds, where at each round, a player owning a computational power of μ has a probability μ of mining the block. At the end of the round, exactly one player is chosen to be the miner of the block according to their hashrate and another round begins. Note that this is true even if a player performs selfish mining: another round begins as soon as the adversary mines a block, even though this block is not broadcasted. Indeed, the mining process being memoryless, the probability that an honest player mines a block does not depend on the time they already spent trying to mine it.

Finally, we consider a "fresh" client that connects to an already running blockchain. It does not know anything about the main chain but its genesis block, and the chain already has a considerable number of blocks in it. As a consequence, the client did not store any block from the main chain that it trusts to aid future synchronisation.

Adversary model Once again, we adopt the same adversary model as in the original FlyClient paper. An adversary in our model owns a portion $\mu < \frac{1}{2}$ of the total hashrate. They are able to mine on the main chain, to perform selfish mining on a fork or to split their hashrate between a fork of their own and the main chain. The original FlyClient paper also assumes that the adversary is able to reorder messages within a round of the Bitcoin Backbone protocol. Since the adversary does not use it for their attack, we omit this assumption in our model.

The goal of the adversary is to have the client accept a transaction lying in a block the main chain does not know about. Since the client knows nothing about the main chain but its genesis block, the adversary is free to fork the main chain as far as they want, as long as their genesis block is the same as the one the client knows about, given that the client trusts the chain with the most accumulated difficulty on it, which is, in the constant difficulty setup, the longest chain.

Since they own less than half of the total hashrate, they are able to put fake blocks within their fork to make their fork as long as the main chain. A fake block can be:

- A block having an invalid PoW, which means that their hash is not lower than the target difficulty T;
- A block having invalid transactions in it, like invalid signed transactions or transactions spending from an invalid UTXO, as explained in section 3.3;
- A block whose previous block reference in its header does not match the block that precedes it, be it a block lying in the main chain or in the adversary's fork.

Creating such a block is computationally easy for the adversary, since they can simply sample a block from the previous part of the chain to satisfy the PoW

requirement. Of course, these blocks would have an invalid previous block reference, and the adversary won't be able to provide a correct MMR proof of inclusion for these. They can present any block to the main chain, which will be accepted if and only if its PoW is valid and its previous block reference matches that of the latest block of the chain. The adversary is also able to create a block with a valid PoW and previous block reference but whose transactions are invalid. This is the case of the forking block, as explained in section 3.3. Note that since we're working in the velvet fork setup, the interlink data (i.e. the MMR root) does **not** have to be valid, nor present, for a block to be accepted, as explained in subsection 2.2.

We make the reasonable assumption that the adversary owns less than half of the total hashrate. However, we do **not** assume that more than half of the updated miners are honest. For instance, it is possible that 40% of the hashrate is updated, while the adversary owns 25% of the hashrate and is dishonest. This may happen if several adversaries collude or if a powerful adversary is amongst the first miners to be updated.

Since the adversary can choose whenever they want to try to fool the client, we also make the assumption that the client is requested to verify a transaction the adversary sent it. In a more realistic setting, the client would ask the provers to provide it with the transactions that concern it. The only thing that changes between the two situation is whether the adversary has to prepare the attack and launch it whenever they want, or if they have to wait for a client to connect, once the attack is set up.

3.2 Notations

We use a Python-like indexing for the blockchain's blocks. Hence, the genesis block is denoted $\mathsf{C}[0]$, the last one $\mathsf{C}[-1]$ and the set of blocks from height i inclusive to height j exclusive $\mathsf{C}[i:j]$. The length of the chain is denoted by n. Hence, the last block can also be denoted $\mathsf{C}[n-1]$. Blocks that are in the adversary's fork are denoted with a $'$, like $\mathsf{C}[f+1]'$ for instance.

We denote the adversary's portion of the total hashrate by μ. According to our adversary's model, we have $\mu < \frac{1}{2}$.

3.3 Principle of the attack

Setup As stated in our model, we consider a "fresh" client that does not know anything about the blockchain except its genesis block. As stated in our model section, it will connect to two provers, one of which is the adversary, whenever the adversary decides to have the client verify a fake transaction.

Before contacting the client, the adversary was honestly mining on the main chain. As such, they have an up-to-date chain and are connected to several honest miners that they contact whenever they mine a block. Reciprocally, they learn about any new block on the chain mined by a honest miner.

Attack description One can read in the original FlyClient paper: "once a malicious prover forks off from the honest chain, it cannot include any of the later honest blocks in its chain because the MMR root in those blocks would not match the chain" [2]. The *chain-sewing* attack on FlyClient is exactly about this: merging the adversary's fork with the main chain. Such a situation is represented on Figure 5.

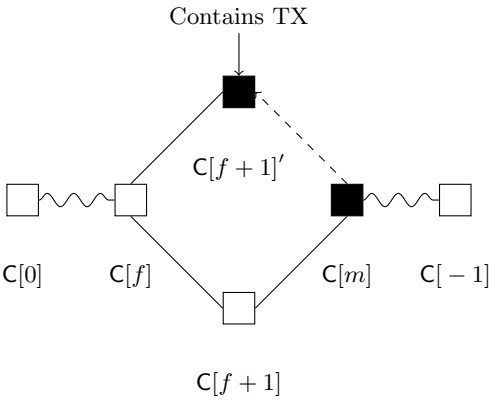

Fig. 5. A *chain-sewing* attack on FlyClient. Curved lines represent one or more blocks. Black blocks are to be mined by the adversary. Dashed arrows represent MMR commitment when different from the previous block reference. Since the root that the adversary includes in their block is root of the MMR containing $\mathsf{C}[f+1]'$, it is easy for them to provide a proof of inclusion of $\mathsf{C}[f+1]'$.

Definition 1. *The block on top of which the adversary begins their fork is called* ***the forking block*** *and is denoted by* $\mathsf{C}[f]$.

Definition 2. *The block that merges the adversary's fork with the main chain is called* ***the merging block*** *and is denoted* $\mathsf{C}[m]$.

The adversary does the following.

1. Starting from the forking block $\mathsf{C}[f]$, the adversary mines a valid block $\mathsf{C}[f+1]'$ but does not broadcast it to the rest of the network.
2. The adversary then mines the merging block, whose previous block reference is the hash of $\mathsf{C}[f+1]$, but the root inside this block is the root of the MMR containing $\mathsf{C}[0:f+1]$ and $\mathsf{C}[f+1]'$.
3. For every block mined by the adversary on the main chain after having mined the merging block, they include in it the root of the MMR where $\mathsf{C}[f+1]'$ replaced $\mathsf{C}[f+1]$.

By doing so, it is now easy for the adversary to provide an MMR proof of inclusion of $\mathsf{C}[f+1]'$ within the chain. Actually, if it isn't for the previous block

reference in $\mathsf{C}[m]$, nothing distinguishes $\mathsf{C}[f+1]$ from $\mathsf{C}[f+1]'$ from the client's point of view. Even worse, for every block sampled by the client, both provers are able to provide an MMR proof of inclusion, by considering the other one's valid blocks as legacy. Hence, it is possible for the adversary to be trusted by the client if their proof is checked before the honest prover's. More generally, the client has no way to distinguish the honest prover from the dishonest one. Hence, the probability of success of this attack is 1, assuming that the client does not check for consistency between previous block reference and sampled blocks. This assumption may be justified by the fact that such a check is not necessary to prove the security of FlyClient. As such, this is not part of the protocol as described in the original article [2].

Adversary's strategies It is high-likely that the adversary won't manage to mine both $\mathsf{C}[f+1]'$ and $\mathsf{C}[m]$ before the honest miners do. Hence, they have basically two strategies:

1. **Add fake blocks to their fork** until they manage to mine $\mathsf{C}[m]$;
2. **Retry the attack** if they don't manage to mine $\mathsf{C}[m]$ before the honest miners do.

We capture both these strategies by defining a new parameter $\overline{F}$.

Definition 3. ***The maximum number of fake blocks that an adversary accepts in their fork*** *is denoted $\overline{F}$. If the adversary does not manage to include less than $\overline{F}$ fake blocks in their fork, they redo the attack.*

For instance, setting $\overline{F} = 0$ means retrying the attack until managing to have the setup shown on Figure 5. Setting $\overline{F} = +\infty$ however means adding fake blocks until managing to mine the merging block. Such a situation is shown on Figure 6.

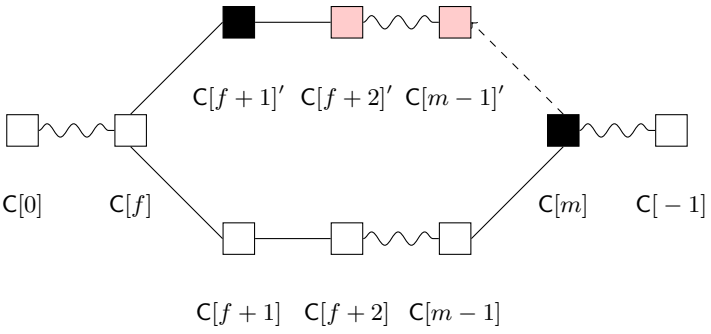

Fig. 6. An example of a situation where the adversary had to include fake blocks in their fork. Curved lines represent one or more blocks. Red blocks represent invalid blocks. Dashed arrows represent an MMR commitment if different from the previous block reference.

If the adversary includes fake blocks in their fork, then the probability of success of the attack decreases, since the client will declare the adversary as dishonest if it samples a fake block. Hence, setting $\overline{F} = 0$ leads to a higher probability of success. However, it may be hard for the adversary to manage to put the attack in place with a small $\overline{F}$, especially if they have a small μ. On the other hand, setting $\overline{F} = +\infty$ leads to the minimal probability the adversary can get, but is the easiest scenario to put in place. This will be discussed further in section 4. This situation is what we call thereafter the **direct setup**.

Definition 4 (Direct setup). *The adversary uses the so-called **direct setup** with parameter $\overline{F}$ if they try to mine the merging block $\mathsf{C}[m]$ as soon as possible while accepting a maximum of $\overline{F}$ fake blocks in their fork, as shown on Figure 6.*

The adversary adds a fake block to their fork whenever they don't manage to mine $\mathsf{C}[m]$ before the honest miners. If the number of fake blocks exceeds $\overline{F}$, then the adversary redo the attack from the beginning.

Finally, even though it wasn't described in the FlyClient original paper, it is possible that the client checks the previous block reference between two consecutive sampled blocks. In this case, the setup depicted on Figure 5 does not succeed with probability 1 anymore, since $\mathsf{C}[m]$ being sampled reveals an inconsistency between its previous block reference and $\mathsf{C}[f+1]'$, which the adversary has to provide since it contains the transaction to be accepted. In order to overcome this problem, the adversary can increase its probability of success by mining another valid block on top of $\mathsf{C}[f+1]'$ before mining the merging block, as shown on Figure 7.

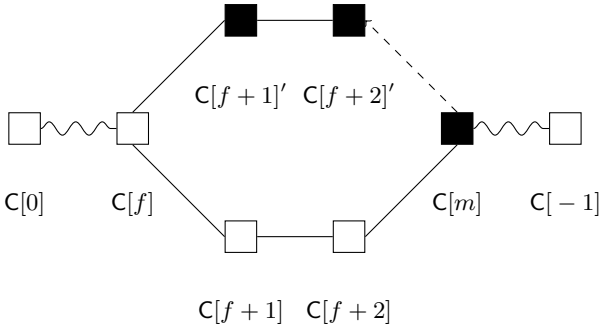

Fig. 7. Trying to mitigate the merging block sampling problem. Curved lines represent one or more block. Dashed arrows represent MMR commitment if different from the previous block reference.

By doing so, one might believe that the adversary slightly increases the probability of success of the attack. Indeed, the attack now fails if both $\mathsf{C}[f+2]'$ and $\mathsf{C}[m]$ are sampled, or if the prover has to provide the client with the hash of $\mathsf{C}[f+2]'$ and $\mathsf{C}[m]$ is sampled. For instance, if m is odd, then in order to build

the MMR proof of inclusion of $\mathsf{C}[m]$, the prover has to provide the client with the hash of the block just before. Indeed, in order to build the very first hash in the proof of inclusion, the hash of $\mathsf{C}[m-1]$, that is $\mathsf{C}[f+2]'$ would be required, since $\mathsf{C}[m]$ is at the rightmost position in this subtree. However, if m is even, then so is $f+1$. Hence, since $\mathsf{C}[f+1]'$ is sampled, the hash of $\mathsf{C}[f+2]'$ would be present in the MMR proof of inclusion of $\mathsf{C}[f+1]'$. Indeed, since $\mathsf{C}[f+1]'$ is at the leftmost position in this subtree, the hash required to build the very first step of the inclusion proof is the next block's. Hence, in every case, there is a possibility for the client to detect the inconsistency between the previous block reference in $\mathsf{C}[m]$ and the hash of $\mathsf{C}[f+2]'$. In order to use this strategy, yet another valid block has to be included within the adversary's fork, as shown in Figure 8.

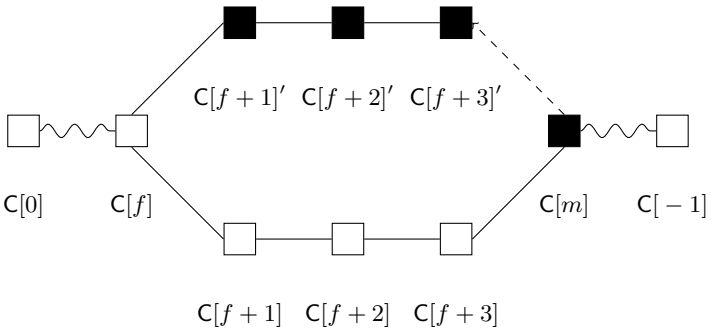

Fig. 8. The valid-between setup, mitigating the merging block sampling problem. Curved lines represent one block or more. Dashed arrows represent MMR commitment if different from the previous block reference.

Now the attack only fails if $\mathsf{C}[m]$ is sampled and if either m is odd or if $\mathsf{C}[f+3]'$ is sampled or if m is even and $\mathsf{C}[f+2]'$ is sampled. Even though this increases the probability of success, this is also much harder to put this in place for the adversary. This will be further discussed in section 4. This strategy will be called the **valid-between setup** from now on.

Definition 5 (Valid-between setup). *The adversary uses the* ***valid-between setup*** *if they try to mine three honest blocks on top on the forking block, the last being* $\mathsf{C}[m]$ *which is broadcasted to the network.*

If the adversary does not manage to mine these three blocks before the honest miners, then they redo the attack from the beginning.

Finally, it is interesting to highlight the fact that even in the case of a deployment under a velvet fork, the adversary still cannot sample from previous parts of the chain to avoid putting fake blocks in their fork. Indeed, if an adversary were to do this, they would have to declare these blocks as legacy since they wouldn't be able to provide a MMR proof of inclusion. Whether the sampled

block has a MMR root in its header does not matter: since it does not commit to the adversary's fork, the proof would fail. Hence, the adversary has no choice but to declare these blocks as legacy. Thus, they would have to provide every subsequent block, eventually having to provide $\mathsf{C}[m]$, whose previous block reference won't match $\mathsf{C}[m-1]$.

Impact of the attack The attack we just described allows the adversary to perform a double-spend transaction: they can have the client accept a transaction that does not exist in the main chain. However, a much more powerful attack is possible using the exact same setup.

Since $\mathsf{C}[f+1]'$ isn't verified by the consensus, it can include invalid transactions and as such, invalid UTXOs from which the adversary can then spend. For instance, let us consider the setup shown on Figure 9. The adversary firstly creates an improperly signed transaction where an arbitrary player C gives them an arbitrary number of coins. They then create transactions recursively spending from this UTXO to create an arbitrary number of properly signed transactions. Finally, they create the transaction they want the client B to accept.

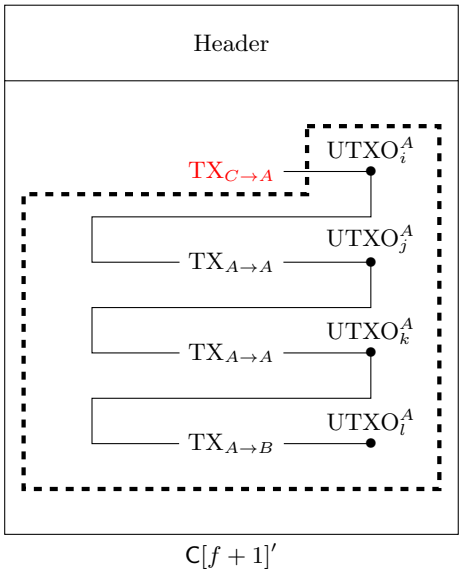

Fig. 9. Printing coins using the *chain-sewing* attack when the client recursively checks up to three UTXOs. $\mathrm{TX}_{X\to Y}$ is a transaction spent by X containing an UTXO that Y can use. Red transactions are improperly signed ones. The dashed zone represents what the client can see. Since the first transaction is created by the adversary, they can include an arbitrary number of coins they can spend later.

The insight behind this is that as a superlight client, B cannot recursively follow every UTXO a transaction is spending from. Indeed, it would otherwise have to sample every block being involved in this UTXO up to the genesis block, which would be a huge loss in efficiency. Hence, there is a maximum depth up to which the client recursively checks that the transactions associated to the UTXO the adversary is spending from are valid.

Of course, this method is defeated by the client asking the adversary the entire block, or recursively checking more UTXOs than the adversary can put within a block. Since, the adversary knows the protocol in advance, they can adapt themselves though. If they know that the client performs such a strategy, they can create one or more blocks within their fork, containing the UTXOs they want to spend from. Such a strategy is shown on Figure 10. Note that this is more difficult for the adversary though, since they have to mine more valid blocks within their fork.

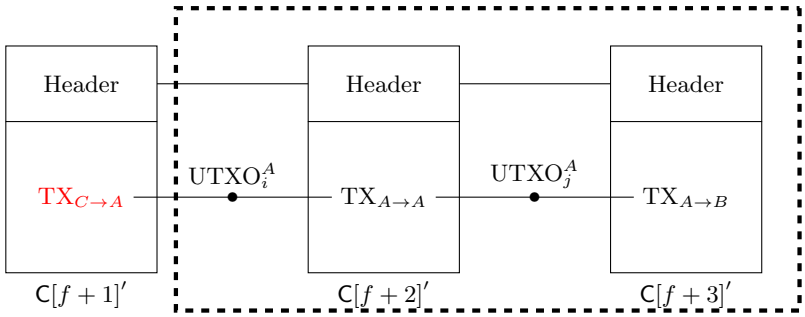

Fig. 10. Printing coins using the *chain-sewing* attack assuming the client samples two whole blocks. $\mathrm{TX}_{X\to Y}$ is a transaction spent by X containing an UTXO that Y can use. Red transactions are improperly signed ones. The dashed zone represents what the client can see. The adversary mines on their fork as much blocks as necessary to prevent the client from checking an improperly signed transaction.

This attack is not possible against an SPV client. Indeed, if an SPV client considers a block in the longest chain with enough confirmations, then it is certain that this block has been verified by the consensus and is thus valid. This is not the case with FlyClient: even a block which is deep in the longest chain can be invalid using the *chain-sewing* attack. In this attack, every block is invalid since they all contain transactions that eventually point to an improperly signed transaction, but the client has no way to know it.

4 Analysis of the *chain-sewing* attack

We now aim to analyse the *chain-sewing* attack. That is, we aim at describing the probability of success of the attack, the time taken by the adversary to put the attack in place and the cost that they must pay to set the attack up.

4.1 Assumptions and notations

Blockchain-related notations We still use a Python-like indexing for the blockchain, and denote its length n. We denote by m the height of the merging block. Hence, $n-m$ represents the depth of the merging block within the chain. We will use the security parameter δ introduced in the original FlyClient [2] but will consider it as a fixed constant with value 2^{-10}, which is the value that was used for demonstrating purposes in the FlyClient original paper. Similarly, the client samples independently q blocks from the blockchain. We will assume q to be equal to 670, accordingly to the values presented in the FlyClient original paper. We define g to be the proportion of upgraded miners. Finally, we denote by R the block reward for mining a block.

Adversary-related notations The adversary's computational power is denoted μ, and is assumed to be less than $\frac{1}{2}$. Since we assume that the adversary is up-to-date, we have $\mu \leqslant g$. Unless specified otherwise, we do not assume that a majority of upgraded miners are honest, which means we do not assume that $\frac{\mu}{g} < \frac{1}{2}$. F represents the number of fake blocks included within the adversary's fork once they managed to put the attack in place, while $\overline{F}$ represent the maximum number of blocks that an adversary accepts in their fork. When the adversary does not manage to have $F \leqslant \overline{F}$, they redo the attack. The number of times that the adversary has to redo the attack is denoted N. The probability of success of the attack is denoted by p_{success}. Finally, the adversary's cost is denoted by C, and the total time in blocks to run the attack is denoted by t. If the adversary wants to run the attack with a time constraint, we also introduce the parameter $\overline{t}$. For instance, if $\overline{t} = +\infty$, then the adversary does not care about the time taken to run the attack. These notations are summarised in Table 1.
It is important to note that we will conduct our analysis under the constant difficulty case for simplicity's sake, and later show in subsection 4.5 that the best strategy an adversary can adopt is even magnified by FlyClient in the variable difficulty case.

Finally, it is important to explain what is the adversary's cost.

Definition 6. ***The adversary's cost*** *C is defined as, in average, the number of block rewards they are losing by mining on their fork rather than on the main chain. It means that the adversary's loses in average $R\,\mu$ coins per block mined on the main chain.*

Note that while the adversary loses coins in average for every block they mine on their fork, they gain in average $R\,(1-\mu)$ coins by mining the merging block, since they have to mine it.

Assumption on the position of the block containing the transaction to be verified There is no particular reason for which the block containing the transaction must be the first one in the adversary's fork. Actually, the adversary increases slightly their probability of success by putting the block containing

Notation	Meaning
$\mathsf{C}[k]$	Block at height k from an honest node's point of view
$\mathsf{C}[k]'$	Block at height k from the adversary's point of view
n	Length of the chain
m	Height of the merging block
$n-m$	Depth of the merging block within the chain
δ	Security parameter of FlyClient
q	Number of blocks sampled by the client
g	Proportion of upgraded miners
R	Block reward
μ	Adversary's computational power
F	Number of fake blocks in the adversary's fork
$\overline{F}$	Maximum number of fake blocks in the adversary's fork
N	Number of times that the adversary has to redo the attack
p_{success}	Probability of success of the attack
C	Adversary's cost for running the attack
t	Time in blocks to run the attack
$\overline{t}$	Average maximum time in blocks to run the attack

Table 1. Notations used throughout the analysis.

the transaction at position $m-1$. This is equivalent to increasing $n-m$ by 1, which leads to a very small increase of their probability of success, as stated in Corollary 1. In order to minimise the adversary's probability of success, we will assume for this analysis that the block containing the transaction to be verified is at position $f+1$ in the adversary's fork. When discussing countermeasures in section 5 though, we release this assumption to catch all possible adversarial strategies.

4.2 Analysis of the direct setup

Adversary's cost We prove Theorem 1 in subsection A.1.

Theorem 1. *When using the direct setup, the cost that an adversary must pay to run the attack is given by:*

$$\mathbb{E}\left[C\middle|\mu,\overline{F}\right] = \frac{R}{1-(1-\mu)^{\overline{F}+1}}. \tag{1}$$

In particular, the adversary's cost is a decreasing function of $\overline{F}$. However, increasing $\overline{F}$ leads to include more fake blocks in average, inducing a smaller probability of success.

Probability of success of the attack while minimising the cost We prove Theorem 2 in subsection A.2.

Theorem 2. *When using the direct setup, the probability of success of the attack $p_{success}$, being given the number of fake blocks F is given by:*

$$\begin{cases} \left[1 - \frac{\ln\left(1-\frac{1}{n-m}\right)}{\ln(\delta)}\right]^q & \text{if } F = 0 \\ \left[1 - \frac{2\,\ln\left(\frac{n-m}{n-m+F}\right)+\ln\left(1-\frac{1}{n-m}\right)}{2\,\ln(\delta)}\right]^q & \text{if } 1 \leqslant F \leqslant \frac{(n-m)\,(1-\delta)-1}{\delta} - 2 \cdot \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Theorem 2 gives us a relation between $\bar{t}$ and $\overline{F}$. Indeed, if we denote t_{setup} the time taken by the adversary to set the attack up, then we have the following inequation, by definition of $\bar{t}$:

$$n - m + \mathbb{E}\left[t_{\text{setup}} \middle| \mu, \overline{F}\right] \leqslant \bar{t}. \tag{3}$$

Hence, choosing $\bar{t}$ also forces the maximum value for $n - m$, which itself upper-bounds $\overline{F}$. However, if the adversary sets $\bar{t} = +\infty$, then they can also set $\overline{F} = +\infty$, hence minimising their cost, according to Theorem 1. We do not solve for every possible $\left(\overline{F}, \bar{t}\right)$ couples, but we will assume in the following that $\overline{F} < \frac{(n-m)\,(1-\delta)-1}{\delta} - 2$. An example of how $\bar{t}$ impacts $\overline{F}$ and thus, the adversary's cost, is shown on Figure 11.

Note that the non-differentiable points on the curve occur whenever the adversary is able to pick a larger value for $n - m$, and as such a larger value for $\overline{F}$, incidentally reducing their cost while still satisfying the constraint derived from $\bar{t}$.

Theorem 2 directly leads to Corollary 1, which is the main result of this part.

Corollary 1. *When using the direct setup, the expected probability of success of the attack is given by:*

$$\mathbb{E}\left[p_{success} \middle| F \leqslant \overline{F}\right] = \frac{\mu}{1-(1-\mu)^{\overline{F}+1}} \left[(1-p_0)^q + \sum_{k=1}^{\overline{F}} (1-p_k)^q\,(1-\mu)^k\right] \tag{4}$$

where p_k is the probability that the attack fails from the first sampled block if there are k fake blocks in the adversary's fork, that is, given that $F = k$:

$$\forall k \in [\![0\,;\,\overline{F}]\!]\,, p_k = \begin{cases} \frac{\ln\left(1-\frac{1}{n-m}\right)}{\ln(\delta)} & \text{if } k = 0 \\ \frac{2\,\ln\left(1-\frac{k}{n-m+k}\right)+\ln\left(1-\frac{1}{n-m}\right)}{2\,\ln(\delta)} & \text{otherwise} \end{cases}. \tag{5}$$

An important remark is that this probability of success converges to 1 as $n - m$ goes to infinity. Since the adversary is the one to choose $n - m$, they can wait as long as they want to have a probability of success arbitrarily close to 1, even when setting $\overline{F} = +\infty$. This is shown on Figure 12.

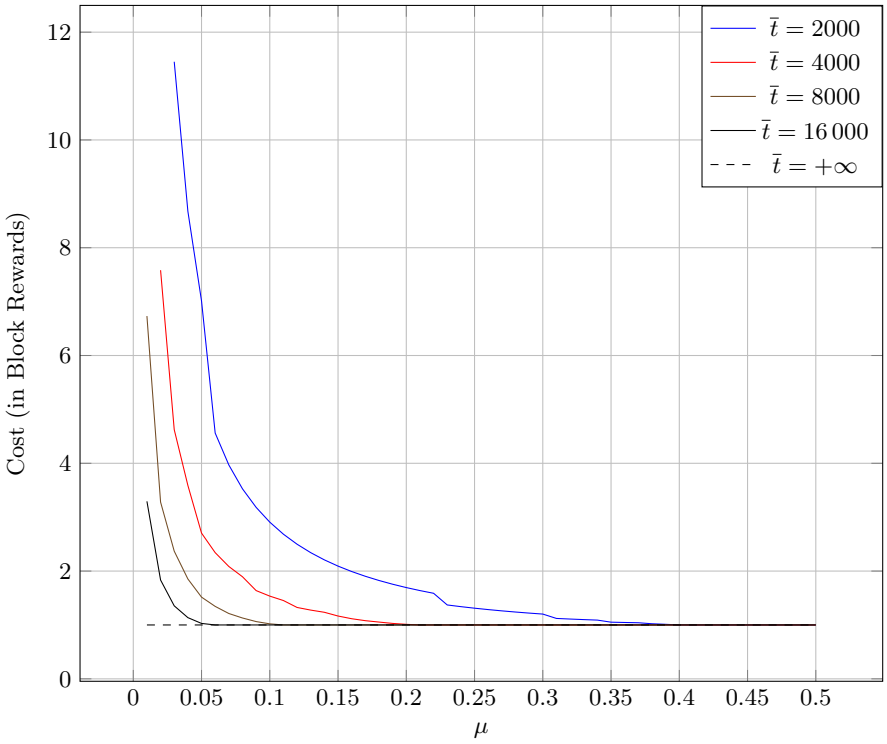

Fig. 11. Evolution of the adversary's cost for different values of $\bar{t}$ in order to get a probability of success larger than 95% knowing that $\overline{F}$ is set at its largest possible value. The smaller $\bar{t}$, the smaller $\overline{F}$, hence the higher the cost. Furthermore, the higher $\bar{t}$, the closer is the cost close to the optimal cost. At the contrary, for too small values of $\bar{t}$, the adversary may be unable to run the attack since no configuration can satisfy the constraint derived from $\bar{t}$.

4.3 Analysis of the valid-between setup

Adversary's cost We prove Theorem 3 in subsection A.3.

Theorem 3. *When using the valid-between setup, the cost that an adversary must pay to run the attack is given by:*

$$\mathbb{E}\left[C|\mu\right] = \left[\frac{3\,\mu + 1}{\mu^3\,\left(10 - 20\,\mu + 15\,\mu^2 - 4\,\mu^3\right)} - 1\right] R\,. \tag{6}$$

This cost is way higher than the direct setup's one, even when setting $\overline{F} = 0$. However, this leads to a higher probability of success.

Probability of success of the attack using the valid-between setup We prove Theorem 4 in subsection A.4.

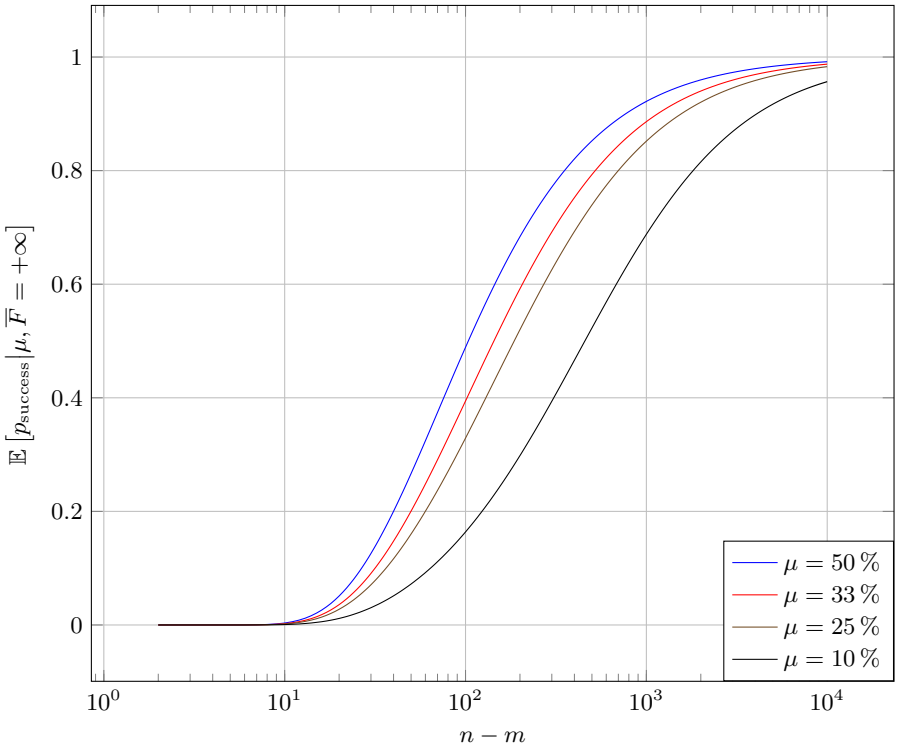

Fig. 12. The convergence of p_{success} to 1 as $n - m$ goes to infinity implies that an adversary can have a probability of success as high as they want, no matter how much fake blocks they include in their forks, as long as they wait for long enough.

Theorem 4. *The probability of success of the attack is given by:*

$$p_{success} = (1 - p_{m-1})^q + (1 - p_m)^q - (1 - p_{m-1} - p_m)^q \tag{7}$$

where p_i is the probability that block at height i is sampled if $q = 1$:

$$p_i = \frac{\ln\left(1 - \frac{1}{n-i}\right)}{\ln(\delta)}. \tag{8}$$

Once again, the adversary can make the probability of success arbitrarily close to 1 by increasing $n - m$, which is shown on Figure 13.

4.4 Comparison of the direct and valid-between setups

It is clear that even if the adversary adopts the direct setup and sets $\overline{F}$ to 0, their cost would still be lower than if they had taken the valid-between setup. However, the valid-between setup can reach a given probability faster than the

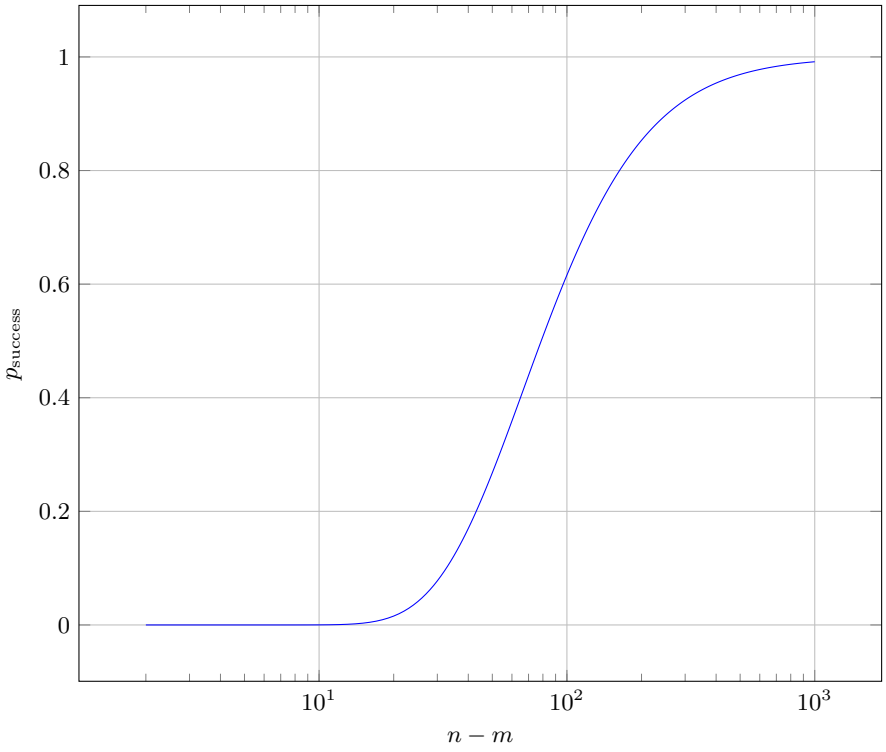

Fig. 13. The convergence of p_{success} to 1 as $n - m$ goes to infinity implies that an adversary can have a probability of success as high as they want as long as they wait for long enough.

direct setup. If the adversary does not care about their cost but only want to run the attack as fast as possible, the valid-between setup is often the best choice. This is shown on Figure 14.

This figure shows that starting from $\mu \approx 10\,\%$, the adversary has to choose the valid-between setup to run the attack as fast as possible while wanting to keep the expected probability of success above $95\,\%$. Note that this implies a way higher cost than the one they would have get by choosing the direct setup.

4.5 Impact of variable difficulty on the attack

FlyClient has originally been designed to work under a variable-difficulty protocol like the Bitcoin one, while we considered a constant difficulty protocol throughout this analysis. In this subsection, we aim to prove that considering FlyClient as a constant difficulty protocol does not change the adversary's strategy of waiting. At the contrary, FlyClient deployed for the Bitcoin protocol incentivises the adversary to wait for the merging block to be deeper in the chain. Indeed, using Lemma 1 and data from [1], we can compute the difference between the

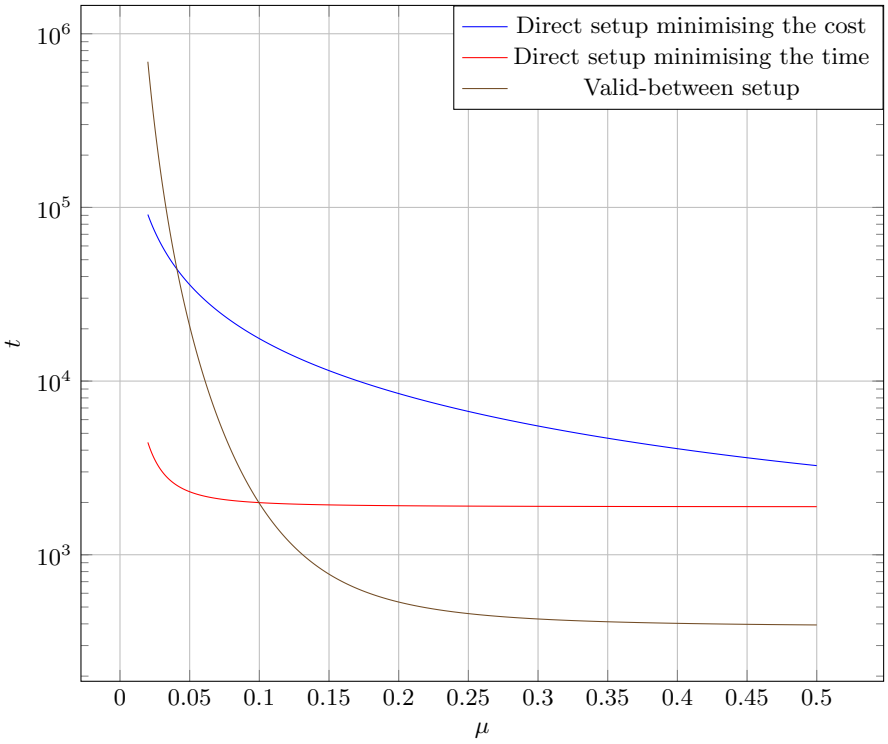

Fig. 14. Comparison of the direct setup and the valid-between setup in terms on how long does it take to set them up in order to have an expected probability of success of 95 %. Starting from $\mu \approx 10\,\%$, at the price of a higher cost, the adversary can put the attack in place faster using the valid-between setup rather than the direct setup.

constant-difficulty probability density function and the variable-difficulty one. This is shown on Figure 15.

What this figure shows is that when FlyClient is deployed for the Bitcoin protocol, it samples even more recent blocks than in the constant-difficulty case. More precisely, if the adversary waits long enough for the merging block to be around a depth of 0.85, then their probability of success would be higher than what we've described in our analysis, since these blocks would be sampled less often than they would have been in the constant difficulty case. Hence, this justifies the fact that we simplify our analysis by working in the constant-difficulty case.

5 Countermeasures to the attack

In this section, we aim at describing how the *chain-sewing* attack against FlyClient can be mitigated by proposing several methods and discussing each one's

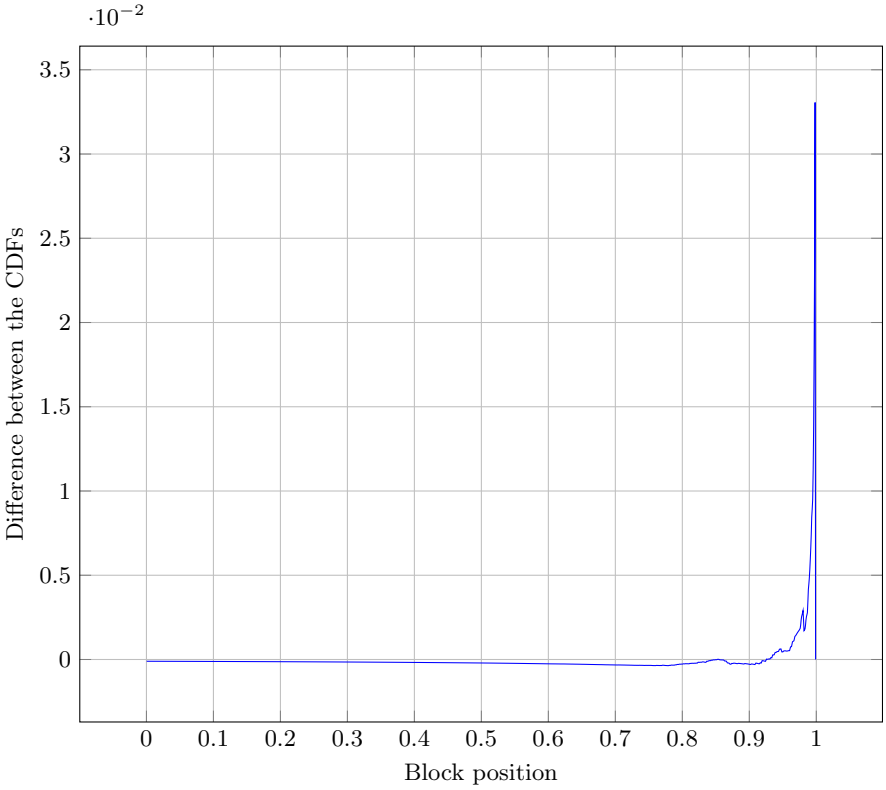

Fig. 15. Difference between the CDFs of a block being sampled at a given height in the Bitcoin case and in the constant-difficulty one.

pros and cons. Note that contrarily to section 4, we do not assume anymore that the block containing the transaction is the first one in the adversary's fork.

5.1 Using a Binary Search

The Binary Search Approach has already been described in the original FlyClient paper. However, it was discarded for being interactive. We describe how can this strategy be used here since, despite being interactive, it is a very efficient approach to prevent *chain-sewing* attacks.

Since by assumption one of the prover is honest, we can compare their answers to determine the position of a merging block. Indeed, we can slice the chain into three distinct parts:

1. From the genesis to the forking block, where both provers agree on their proofs.
2. Between the forking block and the merging block, both non-included, where provers disagree on blocks hashes.

3. After the merging block, where provers disagree on proof of inclusion.

Note that this last part may be potentially empty, if the adversary does not merge their fork into the main chain. Hence, by comparing both provers answers for each block query, it is straightforward for the client to perform a binary search to find the merging block. Two cases are possible: either a merging block is found, or no merging block is found. In the former case, the client knows which prover is the adversary, since they can then ask for $\mathsf{C}[m+1]$ and check the previous block reference in $\mathsf{C}[m]$. If no merging block has been found, it means that the adversary did not close their fork. We can then use another Binary Search to look for the forking block and then sample uniformly in the tip of the chain. Since the adversary did not close their chain, while claiming having a chain as long as the main one, they had to include numerous fake blocks within it. If they have included a portion r of fake blocks within their fork, with r approaching $1-\mu$ with the chain growing, the probability of detecting it after k queries is $1-r^k$.

This approach, in spite of being inherently interactive, can be more efficient than FlyClient, since it only requires $\lceil \log_2(n) \rceil$ queries to the client if the adversary used a merging block and $2\lceil \log_2(n) \rceil + q$ otherwise, q being the number of blocks uniformly sampled. As a comparison, FlyClient samples $\lceil 50\ln(n) \rceil$ blocks to have a probability of success of 2^{-50}. Depending on the length of the fork, q can be adapted to have a probability of success of 2^{-50}.

5.2 Filtering on the number of legacy blocks

When the adversary provides the client with their proofs of inclusion, they include a lot of legacy blocks in it: both blocks from non-updated miners and from updated miners. Similarly, the legacy blocks in an honest prover's proof of inclusion would be the non-updated miners' and the adversary's.

Hence, if we make the assumption that a majority of up-to-date nodes are honest, that is $\frac{\mu}{g} < \frac{1}{2}$, it is possible to determine which prover is the adversary by filtering on the number of legacy blocks included in their proofs. This assumption is justified by Theorem 5, which we prove in subsection A.5.

Theorem 5. *For each query, the probability that the adversary includes more declared legacy blocks in their inclusion proof than an honest node is equal to* $1-\frac{\mu}{g}$. *Furthermore, the number of declared legacy blocks in the adversary's proof follows a geometric distribution of parameter* μ *and support* $\mathbb{N}$, *while the number of declared legacy blocks in the honest prover's proof follows a geometric distribution of parameter* $g-\mu$ *and support* $\mathbb{N}$.

We now have two potential strategies : counting how many times a prover included more legacy blocks than the other, or summing every number of legacy blocks and compare them at the end. In every case, we define N to be the number of queries done after the forking block, so that the provers have different proofs.

Counting how many times did a prover include more legacy blocks than the other. Using Theorem 5, we can consider each query as a Bernoulli experiment with probability of success $1 - \frac{\mu}{g}$. Hence, we can define the random variable C which counts the number of times the honest prover included less declared legacy blocks in their proof than the adversary. As a reminder, we assume that $\frac{\mu}{g} < \frac{1}{2}$. C follows a binomial distribution with parameters N and $1 - \frac{\mu}{g}$. Hence, the adversary succeeds if C is strictly less than $\left\lfloor \frac{N}{2} \right\rfloor$, so that they have a strict majority. In case of an equality, another sampling must be performed.

$$p_{\text{success}} = \mathbb{P}\left[C \leqslant \left\lfloor \frac{N-1}{2} \right\rfloor\right] = \sum_{k=0}^{\left\lfloor \frac{N-1}{2} \right\rfloor} \binom{N}{k} \left(1 - \frac{\mu}{g}\right)^k \left(\frac{\mu}{g}\right)^{n-k}. \tag{9}$$

Summing the numbers of legacy blocks in each prover's proof. A more efficient method consists in summing the numbers of legacy blocks included in each prover's proof and to compare them at the end. Intuitively, this allows the honest prover to take some advance on the adversary: every time the honest prover includes less blocks than the adversary, which happens with probability at least $\frac{1}{2}$, it increases a bit the probability of winning at the end, since they are more likely to include in total less legacy blocks when they lose than the adversary does. This is more formally described by Theorem 6, which we prove in subsection A.6.

Theorem 6. *The probability that the total number of legacy blocks included by the adversary's in their proof is less than the honest prover's can be approximated by:*

$$p_{success} \approx \Phi\left(-\frac{\frac{g-2\mu}{\mu(g-\mu)}N + 1}{\sqrt{N\left(\frac{1-\mu}{\mu^2} + \frac{1-g+\mu}{(g-\mu)^2} + \frac{2}{g}\right)}}\right). \tag{10}$$

where Φ is the cumulative distribution function of the standard normal distribution.

Note that this approximation gets better with N getting larger. Figure 18 shows how close the model distribution is to the actual distribution for N being relatively large.

Comparison of both methods We are now able to numerically compare these methods. Three parameters are to be considered : the number N of queries done after the forking block, the adversary's hashrate μ and the ratio of updated nodes g. Figure 16 shows the maximum hashrate that an adversary can own given g and N.

This figure shows that the summing method performs better overall than the counting method. Furthermore, this methods allows to keep the same level of

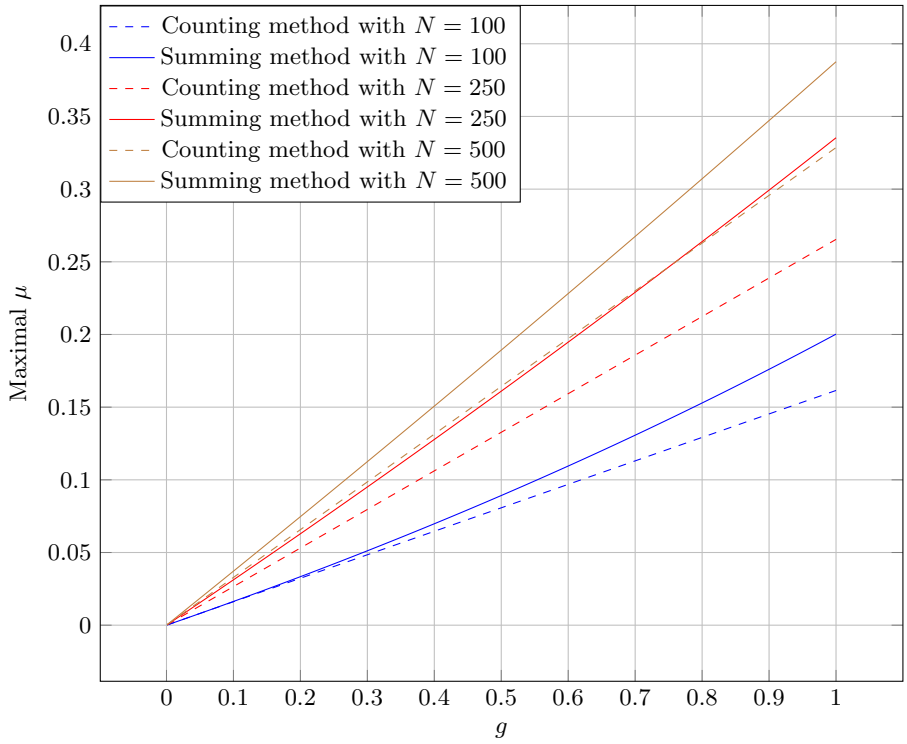

Fig. 16. Comparison of both methods for different N. Counting method results are dashed, while summing method results are solid. The measure taken to compare them is the maximal hashrate an adversary can own such that $p_{\text{success}} < 2^{-50}$. We observe that the summing method is overall better than the counting method.

security, which is a probability in succeeding to fool the client being less than 2^{-50} for an adversary which owns around 33 % of the updated computational power with $N \approx 250$.

This method has several advantages over the other ones. First of all, it is efficient, since it does not require to add any blocks to the proofs. Furthermore, note that the larger N, the more secure the protocol. Since N is the number of blocks sampled after the forking block, waiting for the forking block to be deep in the chain, which the adversary's main strategy, also means increasing N, thus increasing the security of the protocol. Indeed, one can approximate N with the following formula:

$$N \approx q \, \frac{\ln\left(\frac{\delta\, n}{n-m+F}\right)}{\ln(\delta)} \xrightarrow[n\to+\infty]{} q\,. \tag{11}$$

The influence of N on the maximum possible hashrate when using the summing method is shown on Figure 17. It also shows that the approximation only works for relatively large N: it is not possible to find an $N \leqslant 63$ such that $p_{\text{success}} <$

2^{-50}. This directly comes from the fact that the approximation using the Central Limit Theorem gets linearly better with $\sqrt{N}$.

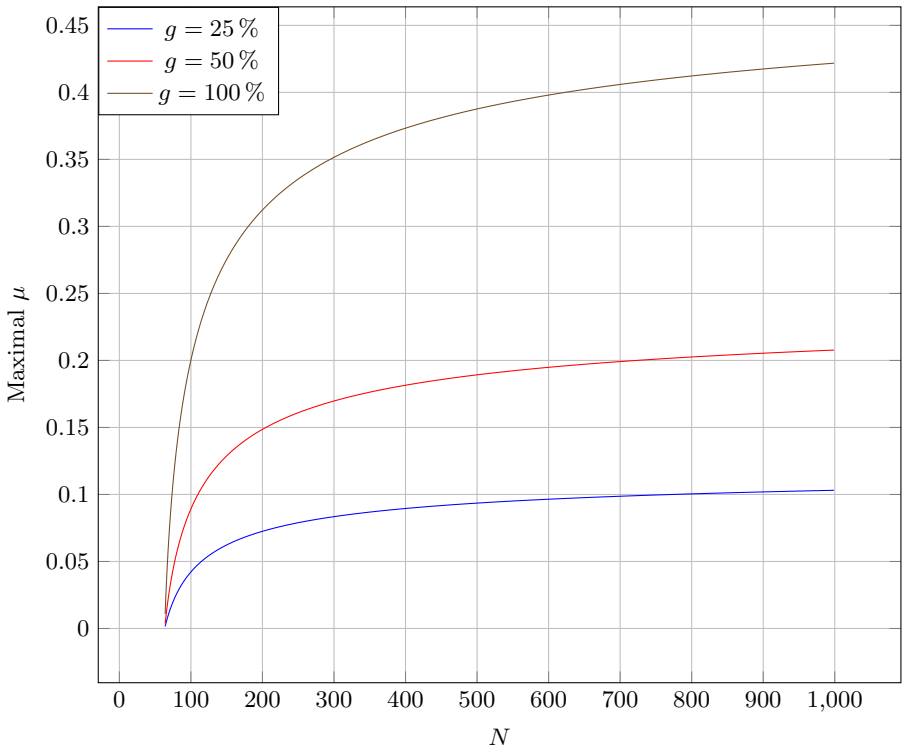

Fig. 17. Influence of N on the summing method for different g. The measure taken to observe it is the maximal hashrate an adversary can own such that $p_{\text{success}} < 2^{-50}$.

A downside however is that we have to assume that the adversary is not too powerful for the system to handle. The client, once having received the proof, knows N and hence knows what the most powerful adversary it can handle is. If N is too small, which is left at the client's discretion, the security of the system can no more be guaranteed. For this reason, we will also describe mitigations to this attack while only assuming that $\mu < \frac{1}{2}$.

5.3 Randomly sampling more blocks

The first idea in order to mitigate the attack is to lower its probability of success such that the adversary would have to wait a unreasonable time to have a non-overwhelming probability of failure. The client can either continue to sample from FlyClient's optimised sampling distribution or use another one.

Sampling more accordingly to FlyClient's sampling distribution Using the former, sampling more blocks does not help. Indeed, let us consider an adversary with 50 % of the hashrate minimising their cost. If they wait for a week once they mined the merging block, that is, they set $n - m = 144 \times 7$, then their probability of success when sampling 670 blocks (which is a reasonable value according to FlyClient's original paper), is around 0.9224. In order to have the same probability of success as the one in the original FlyClient paper, that is 2^{-50}, the client would have to sample around 240 000 blocks, which would be a huge loss in efficiency.

Sampling more uniformly The problem is that the adversary can potentially have a very short fork located anywhere in the chain. Since we cannot gain any information on it, another solution can be to randomly sample until either a fake block is found or an inconsistency between the merging block and the block before is found. In this case, an adversary which maximises their probability of success would include no fake block in their fork. Hence, the client's only way to catch them is to sample both the merging block and the block before. However, this solution does not scale either. Indeed, let us assume that the client, when sampling a block, also asks for both the block that follows and the one that precedes, so that it only has to sample either the merging block or the block before. Assuming independent queries, then if the clients does q queries, given that every block has a probability p of being sampled, the probability of success of the adversary is given by, using the same reasoning as in subsection A.4:

$$p_{\text{success}} = (1 - 2\,p)^q \, \left[2\,(1 - p)^q - (1 - 2\,p)^q\right]. \tag{12}$$

For a blockchain of size n, the client is as efficient as an SPV client if it performs $\frac{n}{3}$ queries, since it samples three blocks at a time. However, even if it performs n queries, p being equal to $\frac{1}{n}$, the probability of success of the adversary can be approximated by:

$$p_{\text{success}} \approx 2\,\mathrm{e}^{-3} - \mathrm{e}^{-4} \approx 0.081. \tag{13}$$

Hence, this method is both less efficient and less secure than an SPV client.

5.4 Deterministically sampling more blocks

Rather than trying to decrease the adversary's probability of success, one can try to increase their minimal cost so that this cost becomes unreasonable. A way to do so is asking the prover to provide the client with the k blocks that follow and the k blocks that precede the block containing the transaction to be verified. This forces the prover to have a fork with length at least $k+1$ valid blocks, which can be particularly difficult. Theorem 7, which we prove in subsection A.7 shows how the cost of an adversary using the direct setup increases with k.

Theorem 7. *The minimal cost that an adversary must pay to run the attack while having to provide k valid blocks after and k valid blocks before the block*

containing the transaction verifies:

$$\mathbb{E}\left[C|\mu\right] \geqslant \left[1 + k\left(1 - \mu\right)\right] R. \tag{14}$$

Furthermore, with probability $1 - \left[\mu\ (2 - \mu)\right]^k$ *and assuming the adversary runs the most optimal strategy, this inequality is actually an equality.*

As a side-effect, using this technique also lowers the probability of success of the attack. For instance, if the client uses $k = 30$, it increases the proof size by approximately 9 % but sets the adversary's minimal cost to run the attack to at least $16\,R$ instead of R, while still being efficient. Note that since the very first block has a valid MMR root by definition, there is no point in asking for an MMR proof of every sampled block: only the last sampled block has to be further verified like this. If this block is a legacy block, then it will be verified the same way FlyClient handles legacy blocks. This greatly reduces the additional data required for this additional check a prover has to provide the client with.

5.5 Implementation cost and security of the considered countermeasures

Three methods arise from the previous analysis, each coming with its drawbacks:

- The Binary Search is interactive;
- Filtering on legacy blocks works in a stronger model;
- Deterministically sampling more blocks transforms the security from cryptographic to cryptoeconomic.

Note that at the exception of the Binary Search, these methods are built on top of FlyClient, which ensures their security, were FlyClient to be implemented as a hard or soft fork. Similarly, the security of the Binary Search approach has been discussed in the original FlyClient paper [2]. The previous section having shown that these methods are secure under a velvet fork, they are secure in all possible considered ways of deploying FlyClient.

A consideration that can be taken into account when choosing one of these three countermeasures is the cost of implementation. Namely, we have to evaluate the cost overhead of the mitigation compared to a vanilla implementation of FlyClient. Note that all these mitigations check for the consistency between the previous block reference and the MMR proofs of inclusion when it can.

Cost overhead of the Binary Search If the client is implemented using the Binary Search, then the adversary has no point in using a *chain-sewing* attack since it would be easily detected. Hence, their best guess is to create a fork according to the original FlyClient use case. While FlyClient samples around $\lceil 50\,\ln(n) \rceil$ blocks, this method only samples $2\,\lceil \log_2(n) \rceil + q$ blocks. Since the probability of catching the adversary grows exponentially with q, q grows logarithmically with $n - m$ to achieve a given probability of success. Hence, this method does not incur a huge cost overhead for the client, and can even be more efficient than vanilla FlyClient, at the cost of non-interactivity.

Cost overhead of filtering on legacy blocks If the client filters on legacy block using the summing method, then it does not sample more blocks than vanilla FlyClient. However, an additional counter must be increased to count how many legacy blocks were included in the proofs. Counting this number of legacy blocks can be done efficiently while parsing the proof and hence induce a negligible cost overhead.

Cost overhead of deterministically sampling more blocks Finally, if the client deterministically samples more blocks, then the cost overhead is sampling k more blocks, where k is a pre-determined constant fixed when implementing the client depending on the desired induced cost for an adversary to run an attack. Note however that only the MMR proof of the last block must be verified, the client then just needs to recursively verify the PoW and previous block reference of the other $k - 1$ blocks.

6 Conclusion

In this paper, we described a powerful attack against FlyClient when naively deployed as a velvet fork, allowing adversaries owning less than half of the total hashrate to perform double-spend transactions and to print coins at a constant cost. We described the different strategies an adversary have at their disposal and provide a comprehensive analysis on these. Finally, we proposed countermeasures to keep the protocol non-interactive and efficient, while ensuring the security of the protocol.

All in all, the benefits of deploying FlyClient as a velvet fork are to be confronted with the constraints which come with it, namely the higher number of blocks to provide the client with and the additional security measures that have to be deployed along with it. While certain measures do not increase the number of blocks to sample, they still increase the amount of computations that have to be done to check the proof, which may result in higher cost for the provers, were FlyClient to be deployed in the cross-chain setting. Furthermore, there is no evidence that no other attack on the protocol deployed under a velvet fork is possible. While the benefits of using a velvet fork seem clear, the potential requirement for an honest updated majority along with the possible security flaws it incurs make its relevance way more debatable.

Finally, it is to be denoted that our attack extensively uses the adversary's ability to declare any block as a legacy one. An even more efficient way to prevent these kinds of attacks would be to have some kind of "meta-PoW", allowing updated miners to differentiate a legacy block from a malicious one. This however would require once again to assume that a majority of updated miners is honest.

7 Acknowledgements

We would like to thank D. Zindros and A. Polydouri for having shared with us their insights about *chain-sewing* attacks and the feedback they gave on this paper.

Furthermore, we would like to thank N. Stifter for having proofread this paper, making it clearer and more precise than it used to be.

References

1. blockchain.com: https://www.blockchain.com/charts/difficulty, accessed on 11/06/20
2. Bünz, B., Kiffer, L., Luu, L., Zamani, M.: Flyclient: Super-light clients for cryptocurrencies. Cryptology ePrint Archive, Report 2019/226 (2019), https://eprint.iacr.org/2019/226
3. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. Cryptology ePrint Archive, Report 2014/765 (2014), https://eprint.iacr.org/2014/765
4. Interlay: Polkadot btc-relay spec (2020), https://interlay.gitlab.io/polkabtc-spec/btcrelay-spec/, accessed on 02/06/20
5. Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work. Cryptology ePrint Archive, Report 2017/963 (2017), https://eprint.iacr.org/2017/963
6. Kiayias, A., Polydouri, A., Zindros, D.: The velvet path to superlight blockchain clients (5 2020)
7. Lai, Y.T., Prestwich, J., Konstantopoulos, G.: Flyclient - consensus-layer changes (2019), https://zips.z.cash/zip-0221
8. Lan, R., Upadhyaya, G., Tse, S., Zamani, M.: Horizon: A gas-efficient, trustless bridge for cross-chain transactions (2021)
9. Lerner, S.D.: Hawkclient: Building a fully decentralized bridge between rsk and ethereum, https://blog.rsk.co/noticia/hawkclient-building-a-fully-decentralized-bridge-between-rsk-and-ethereum/, accessed on 08/01/21
10. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Cryptography Mailing list at https://metzdowd.com (03 2009)
11. theguardian: Bitcoin's market value exceeds $1tn after price soars, https://www.theguardian.com/technology/2021/feb/19/bitcoins-market-value-exceeds-1tn-after-price-soars-to-above-54000, accessed on 15/03/21
12. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: Sok: Communication across distributed ledgers. Cryptology ePrint Archive, Report 2019/1128 (2019), https://eprint.iacr.org/2019/1128
13. Zamyatin, A., Avarikioti, Z., Perez, D., Knottenbelt, W.J.: Txchain: Efficient cryptocurrency light clients via contingent transaction aggregation. In: Garcia-Alfaro, J., Navarro-Arribas, G., Herrera-Joancomarti, J. (eds.) Data Privacy Management, Cryptocurrencies and Blockchain Technology. pp. 269–286. Springer International Publishing, Cham (2020)

14. Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.J.: Xclaim: Trustless, interoperable cryptocurrency-backed assets. Cryptology ePrint Archive, Report 2018/643 (2018), https://eprint.iacr.org/2018/643
15. Zamyatin, A., Stifter, N., Judmayer, A., Schindler, P., Weippl, E., Knottenbelt, W.J.: (short paper) a wild velvet fork appears! inclusive blockchain protocol changes in practice. Cryptology ePrint Archive, Report 2018/087 (2018), https://eprint.iacr.org/2018/087

A Proofs

A.1 Theorem 1

Proof. First of all, according to the Bitcoin backbone protocol [3], mining a block is a memoryless process. Hence, the adversary has no interest in continuing to mine on top of a block to mine the forking block if the honest network has already found the corresponding honest block. Hence, the adversary's best strategy is to try to mine the forking block before the honest network does and to retry on top of the following honest block if they fail.

Since the probability that they find a block before the main chain is μ, they will spend in average $\frac{1}{\mu}$ blocks mining on their fork in order to mine the forking block.

The adversary then wants to mine the merging block before the honest network does, allowing themselves at most $\overline{F}$ fails. Hence, F follows a geometric distribution of parameter μ and support $\mathbb{N}$. As such, we have:

$$\mathbb{P}\left[F \leqslant \overline{F} \middle| \mu\right] = 1 - (1-\mu)^{\overline{F}+1}. \tag{15}$$

Since N follows a geometric distribution of parameter $\mathbb{P}\left[F \leqslant \overline{F}\right]$ and support $\mathbb{N}$, we have:

$$\mathbb{E}\left[N \middle| \mu, \overline{F}\right] = \frac{(1-\mu)^{\overline{F}+1}}{1-(1-\mu)^{\overline{F}+1}}. \tag{16}$$

Finally, the number of fake blocks that the adversary includes in their fork is given by:

$$\mathbb{E}\left[F \middle| \mu, F \leqslant \overline{F}\right] = \sum_{k=0}^{\overline{F}} k \, \frac{\mathbb{P}\left[(F=k) \cap \left(F \leqslant \overline{F}\right)\right]}{\mathbb{P}\left[F \leqslant \overline{F}\right]} \tag{17}$$

which leads to:

$$\mathbb{E}\left[F \middle| \mu, F \leqslant \overline{F}\right] = \frac{(1-\mu)\left[1-(1-\mu)^{\overline{F}}\left(\overline{F}\,\mu+1\right)\right]}{\mu\left[1-(1-\mu)^{\overline{F}+1}\right]}. \tag{18}$$

Since all the random variables are considered independent, this leads to the following average cost:

$$\mathbb{E}\left[C \middle| \mu, \overline{F}\right] = \left[\mathbb{E}\left[N \middle| \mu, \overline{F}\right]\left(\frac{1}{\mu}+\overline{F}+1\right)+\frac{1}{\mu}+\mathbb{E}\left[F \middle| \mu, F \leqslant \overline{F}\right]\right] R\,\mu - R\,(1-\mu) \tag{19}$$

where $\frac{1}{\mu} + \overline{F} + 1$ corresponds to the number of blocks mined on the main chain on average during a failed attempt and $\frac{1}{\mu} + \mathbb{E}\left[F \middle| \mu, F \leqslant \overline{F}\right]$ corresponds to the number of blocks mined on the adversary's fork in average during their successful attempt. This finally leads to:

$$\mathbb{E}\left[C \middle| \mu, \overline{F}\right] = \left(\frac{1}{\mu}\left[1 + \frac{1}{1-(1-\mu)^{\overline{F}+1}}\right] - 1\right) R\,\mu - R\,(1-\mu). \tag{20}$$

This expression can finally be reduced to:

$$\mathbb{E}\left[C \middle| \mu, \overline{F}\right] = \frac{R}{1-(1-\mu)^{\overline{F}+1}}. \tag{21}$$

A.2 Theorem 2

Proof. Let us consider that $F = 0$. Then, the only way the client has to detect the attack is to sample the merging block, which it does with probability $\frac{\ln\left(1-\frac{1}{n-m}\right)}{\ln(\delta)}$ in the constant difficulty case, according to the FlyClient original paper [2].

Now, let us consider $F \geqslant 1$. There are two disjoint ways using which the client can detect the attack, by sampling a fake block, or by sampling the merging block and having to provide the previous block hash in the MMR proof of inclusion, which is the case if m is odd. We will assume that m is odd with probability $\frac{1}{2}$, since the adversary does not control the parity of F, and redoing the attack is quite costly for only a little gain in probability. Since the probability that the client samples a block in $\mathsf{C}[m-F:F]$ in the constant difficulty case is given by $\frac{\ln\left(\frac{n-m}{n-m+F}\right)}{\ln(\delta)}$, the probability that the attack fails in this case is given by $\frac{\ln\left(\frac{n-m}{n-m+F}\right)}{\ln(\delta)} + \frac{\ln\left(1-\frac{1}{n-m}\right)}{2\,\ln(\delta)}$.

There is a corner case however. In order for the probabilities to make sense, $n-m$ cannot be too small if F is large. If we want to have no fake block in the $\delta\,n$ last part of the chain, then F must satisfy $F + 3 < (1-\delta)\,n$, since the genesis, the block that contains the transaction to be verified and the merging block must be included as well, which is equivalent to $\delta\,(F+3) < (1-\delta)\,\delta\,n$. Furthermore, in order for no fake block nor the merging block to be sampled with probability 1, we must have $\delta\,n < n - m - 1$. Finally, this leads to:

$$F < \frac{(n-m)\,(1-\delta) - 1}{\delta} - 2. \tag{22}$$

A simpler way to get this inequation is simply to solve for $\frac{\ln\left(\frac{n-m-1}{n-m+F}\right)}{\ln(\delta)} < 1$. That said, if this inequation is not satisfied, then the probability of a fake block being sampled is equal to 1. Hence, the probability of success of the attack is nil in this case.

A.3 Theorem 3

Proof. According to the Bitcoin Backbone protocol [3], we can model the mining process with a memoryless one. For this reason, we will work in rounds, where, at each round, the adversary mines a block with probability μ. If they fail, the honest miners mine a block, which happens with probability $1-\mu$.

The adversary's best strategy is to try to mine $\mathsf{C}[f+1]'$ until they manage to do it before the honest miners do. This takes the adversary, in average, $\frac{1}{\mu}$ blocks to do. In order to compute the probability of succeeding, let us consider three distinct cases:

- **The adversary wins the first two rounds**. This happens with probability μ^2. In this case, the adversary has to wait until $\mathsf{C}[f+1]$, $\mathsf{C}[f+2]$ and $\mathsf{C}[f+3]$ are mined on the main chain, so that they can include the hash of $\mathsf{C}[f+3]$ in the previous block reference of $\mathsf{C}[m]$. They then succeed with probability μ.
- **The adversary wins one out of the two first rounds.** This happens with probability $2\,\mu\,(1-\mu)$. Then either the adversary wins the next round, wait for the main chain and mine the merging block, which happens with probability μ^2, or they lose the first round, which happens with probability $1-\mu$. In this case, the adversary and the main chain are at the same level: the adversary has no longer a block in advance. The adversary either succeeds by winning the next 2 rounds, or by losing the first and winning the next two. In total, the adversary succeeds with probability $\mu^2+(1-\mu)\,\left[\mu^2+(1-\mu)\,\mu^2\right]$.
- **The adversary loses the first two rounds.** This happens with probability $(1-\mu)^2$. There are now two cases. If the adversary loses the next round, which happens with probability $1-\mu$, they have no choice but to win the next three rounds, which happens with probability μ^3. If they win the first round however, then we're back on the situation where the adversary's fork and the main chain have equal length, whose probability has been computed previously. In total, the adversary succeeds with probability $(1-\mu)\,\mu^3+\mu\,\left[\mu^2+(1-\mu)\,\mu^2\right]$.

Putting things together, the adversary succeeds to put the attack in place with probability:

$$p(\mu)=\mu^3+2\,\mu^3\,(1-\mu)\,\left(\mu^2-3\,\mu+3\right)+(1-\mu)^2\,\mu^3\,(3-2\,\mu) \tag{23}$$

which leads to:

$$p(\mu)=\mu^3\,\left(-4\,\mu^3+15\,\mu^2-20\,\mu+10\right). \tag{24}$$

Hence, the average adversary's cost $\mathbb{E}\left[C|\mu\right]$ is equal to:

$$\left[\left(\frac{1}{\mu^3\,\left(-4\,\mu^3+15\,\mu^2-20\,\mu+10\right)}-1\right)\,\left(\frac{1}{\mu}+3\right)+\frac{1}{\mu}+2\right]\,R\,\mu-(1-\mu)\,R \tag{25}$$

which boils down to:

$$\mathbb{E}\left[C|\mu\right]=\left[\frac{3\,\mu+1}{\mu^3\,\left(10-20\,\mu+15\,\mu^2-4\,\mu^3\right)}-1\right]\,R. \tag{26}$$

A.4 Theorem 4

Proof. In order for the attack to fail, $\mathsf{C}[m]$ has to be sampled. Furthermore, in order to detect the inconsistency between $\mathsf{C}[f+3]'$ and the previous block reference in $\mathsf{C}[m]$, either $\mathsf{C}[f+3]'$ has to be sampled, or m has to be odd. Indeed, if m is odd, then the adversary has to provide the hash of $\mathsf{C}[f+3]'$ in the MMR proof of inclusion of $\mathsf{C}[m]'$. However, the adversary can start to mine on top of $\mathsf{C}[f]$ only if f is even, so that m is even too. This does not increase the adversary's average cost since they will be mining on the main chain while waiting for f to be even.

Hence, the attack fails if both $\mathsf{C}[f+3]'$ and $\mathsf{C}[m]$ are sampled. Let us denote by A_N (respectively B_N) the event "$\mathsf{C}[m]'$ (respectively $\mathsf{C}[m]$) is not sampled if N blocks are sampled by the client". In particular, we have:

$$\mathbb{P}[A_1] = 1 - p_{m-1} \tag{27}$$

and:

$$\mathbb{P}[B_1] = 1 - p_m. \tag{28}$$

"The adversary succeeds" is an event equivalent to $A_N \cup B_N$, N being equal to q in our case. Plus, we have:

$$\mathbb{P}[A_N \cup B_N] = \mathbb{P}[A_N] + \mathbb{P}[B_N] - \mathbb{P}[A_N \cap B_N]. \tag{29}$$

We approximate FlyClient's sampling by independent queries for blocks, as described in the FlyClient original paper. Thus, we have:

$$\mathbb{P}[A_N] = (1 - p_{m-1})^N \tag{30}$$

and

$$\mathbb{P}[B_N] = (1 - p_m)^N. \tag{31}$$

Finally, using the same reasoning, we have:

$$\mathbb{P}[A_N \cap B_N] = (1 - p_m - p_{m-1})^N \tag{32}$$

which concludes the proof.

A.5 Theorem 5

Proof. Let N_H be the number of declared legacy blocks that an honest node includes in their proof. Similarly, we define N_A to be the number of declared legacy blocks that the adversary includes in their proof.

As a reminder, each block has a probability of μ to be mined by the adversary, a probability $g - \mu$ to be mined by an up-to-date honest node and a probability $1 - g$ to be mined by a non-up-to-date node. Putting things differently, an honest node only accepts a portion $g - \mu$ of the blocks in their proofs, the other being declared as legacy, while the adversary only accepts a portion μ of the blocks in their proofs. We only consider queries done after the forking block. Indeed,

before the forking block, the adversary and the honest node will declare the same number of legacy blocks. There's no point for the adversary to lie here, since it would only increase the number of legacy blocks they declare, which benefits the honest prover. Hence, N_H follows a geometric distribution with parameter $g - \mu$ and support $\mathbb{N}$, while N_A follows a geometric distribution with parameter μ and support $\mathbb{N}$.

Note that N_H and N_A are not independent. Indeed, it is impossible to have $N_H = N_A$, since this would mean that both provers accepts the same block to be valid, which is not possible after the forking block. We are to compute the joint distribution of (N_H, N_A). Let be $(i, j) \in \mathbb{N}^2$. We have:

$$\mathbb{P}\left[N_H = i, N_A = j\right] = \mathbb{P}\left[N_H = i | N_A = j\right] \mathbb{P}\left[N_A = j\right]. \tag{33}$$

$\mathbb{P}\left[N_A = j\right]$ is known since we know that N_A follows a geometric distribution with parameter μ and support $\mathbb{N}$. Hence, we are to determine $\mathbb{P}\left[N_H = i | N_A = j\right]$. For this proof only, $\mathsf{C}[0]$ will denote the block queried by the client. Then $\mathbb{P}\left[N_H = i | N_A = j\right]$ is the probability that $N_H = i$ knowing that $\mathsf{C}[0 : j]$ are either legacy or have been mined by an up-to-date miner and that $\mathsf{C}[j]$ has been mined by the adversary. Thus, we have three distinct cases.

If $i < j$: For $i = 0$, this probability is equal to the probability that $\mathsf{C}[0]$ is a honest block (that is, mined by an up-to-date miner) knowing that it is either honest or legacy. Hence:

$$\mathbb{P}\left[N_H = 0 | N_A = j\right] = \frac{g - \mu}{1 - \mu}. \tag{34}$$

Then, knowing $\mathbb{P}\left[N_H = i | N_A = j\right]$ is given by the probability that $\mathsf{C}[0 : i]$ are all legacy blocks and that $\mathsf{C}[i]$ is an honest block, knowing that they aren't adversary-mined blocks. Putting things together, we have:

$$\mathbb{P}\left[N_H = i | N_A = j\right] = \frac{g - \mu}{1 - \mu} \left(\frac{1 - g}{1 - \mu}\right)^i. \tag{35}$$

If $i = j$: This case cannot happen, for the reasons detailed above. Hence:

$$\mathbb{P}\left[N_H = i | N_A = j\right] = 0. \tag{36}$$

If $i > j$: This case happens if every block in $\mathsf{C}[0 : j]$ are legacy blocks, knowing they are not adversary-mined. We also know that $\mathsf{C}[j]$ is adversary-mined. We then have to have $\mathsf{C}[j + 1 : i : t]$o be either legacy or adversary blocks. Putting things together, we have:

$$\mathbb{P}\left[N_H = i | N_A = j\right] = \left(\frac{1 - g}{1 - \mu}\right)^j (1 - g + \mu)^{i-j-1} (g - \mu). \tag{37}$$

Putting things together, we get:

$$\mathbb{P}\left[N_H = i | N_A = j\right] = \begin{cases} \frac{g-\mu}{1-\mu} \left(\frac{1-g}{1-\mu}\right)^i & \text{if } i < j \\ 0 & \text{if } i = j \\ \left(\frac{1-g}{1-\mu}\right)^j (1 - g + \mu)^{i-j-1} (g - \mu) & \text{if } i > j \end{cases}. \tag{38}$$

Hence:

$$\mathbb{P}\left[N_H = i, N_A = j\right] = \begin{cases} \mu\,(g-\mu)\,(1-g)^i\,(1-\mu)^{j-i-1} & \text{if } i < j \\ 0 & \text{if } i = j\,. \\ \mu\,(g-\mu)\,(1-g)^j\,(1-g+\mu)^{i-j-1} & \text{if } i > j \end{cases} \tag{39}$$

We are now able to compute $\mathbb{P}\left[N_H < N_A\right]$:

$$\mathbb{P}\left[N_H < N_A\right] = \sum_{i=0}^{+\infty} \sum_{j=i+1}^{+\infty} \mathbb{P}\left[N_H = i, N_A = j\right] \tag{40a}$$

$$= \sum_{i=0}^{+\infty} \sum_{j=i+1}^{+\infty} \mu\,(g-\mu)\,(1-g)^i\,(1-\mu)^{j-i-1} \tag{40b}$$

$$= (g-\mu) \sum_{i=0}^{+\infty} (1-g)^i \tag{40c}$$

$$= 1 - \frac{\mu}{g}. \tag{40d}$$

A.6 Theorem 6

Proof. For each query, we define N_{A_k} and N_{H_k} to be the number of declared legacy blocks included in respectively the adversary's proof and the honest prover's proof. Their distribution has been computed in subsection A.5. We then define S_N to be:

$$S_N \stackrel{\text{def}}{=} \frac{1}{N} \sum_{k=1}^{N} \left(N_{A_k} - N_{H_k}\right)\,. \tag{41}$$

By assumption, all the $N_{A_k} - N_{H_k}$ are i.i.d. Hence, the central limit theorem ensures that S_N can be approximated by a normal distribution with mean $\mathbb{E}\left[N_{A_k} - N_{H_k}\right]$ and variance $\mathbb{V}\left[N_{A_k} - N_{H_k}\right]$. Using linearity of the expectation, $\mathbb{E}\left[N_{A_k} - N_{H_k}\right]$ is easily found to be equal to:

$$\mathbb{E}\left[N_{A_k} - N_{H_k}\right] = \mathbb{E}\left[N_{A_k}\right] - \mathbb{E}\left[N_{H_k}\right] = \frac{g - 2\,\mu}{\mu\,(g-\mu)}\,. \tag{42}$$

We now are to compute $\mathbb{V}\left[N_{A_k} - N_{H_k}\right]$. We have:

$$\mathbb{V}\left[N_{A_k} - N_{H_k}\right] = \mathbb{V}\left[N_{A_k}\right] + \mathbb{V}\left[N_{H_k}\right] - 2\,\mathbb{V}\left[N_{A_k}, N_{H_k}\right] \tag{43a}$$

$$= \mathbb{V}\left[N_{A_k}\right] + \mathbb{V}\left[N_{H_k}\right] - 2\,\left(\mathbb{E}\left[N_{A_k}\,N_{H_k}\right] - \mathbb{E}\left[N_{A_k}\right]\,\mathbb{E}\left[N_{H_k}\right]\right)\,. \tag{43b}$$

Since N_{A_k} and N_{H_k} both follows geometric distribution with known parameter, every quantity here can easily be computed except $\mathbb{E}\left[N_{A_k}\,N_{H_k}\right]$. Hence, we are to compute it. We have:

$$\mathbb{E}\left[N_{A_k}\,N_{H_k}\right] = \sum_{i=0}^{+\infty} \sum_{j=0}^{+\infty} i\,j\,\mathbb{P}\left[N_{H_k} = i, N_{A_k} = j\right]\,. \tag{44}$$

Let us define s_1 to be:

$$s_1 \stackrel{\text{def}}{=} \sum_{i=1}^{+\infty} \sum_{j=i+1}^{+\infty} i\, j\, \mathbb{P}\left[N_{H_k} = i, N_{A_k} = j\right] \tag{45}$$

and s_2 to be:

$$s_2 \stackrel{\text{def}}{=} \sum_{i=1}^{+\infty} \sum_{j=1}^{i-1} i\, j\, \mathbb{P}\left[N_{H_k} = i, N_{A_k} = j\right]. \tag{46}$$

Since $\mathbb{P}\left[N_{H_k} = i, N_{A_k} = i\right] = 0$ for all $i \in \mathbb{N}$, we have:

$$\mathbb{E}\left[N_{A_k}\, N_{H_k}\right] = s_1 + s_2. \tag{47}$$

Now, we have:

$$s_1 = \sum_{i=1}^{+\infty} \sum_{j=i+1}^{+\infty} i\, j\, \mu\, (g-\mu)\, (1-g)^i\, (1-\mu)^{j-i-1} \tag{48a}$$

$$= \mu\, (g-\mu) \sum_{i=1}^{+\infty} i\, (1-g)^i \sum_{j=i+1}^{+\infty} j\, (1-\mu)^{j-i-1} \tag{48b}$$

$$= \mu\, (g-\mu) \sum_{i=1}^{+\infty} i\, (1-g)^i \sum_{j=1}^{+\infty} (i+j)\, (1-\mu)^{j-1} \tag{48c}$$

$$= \mu\, (g-\mu) \sum_{i=1}^{+\infty} i\, (1-g)^i \left[i \sum_{j=1}^{+\infty} (1-\mu)^{j-1} + \sum_{j=1}^{+\infty} j\, (1-\mu)^{j-1} \right] \tag{48d}$$

$$= \mu\, (g-\mu) \sum_{i=1}^{+\infty} i\, (1-g)^i \left(\frac{i}{\mu} + \frac{1}{\mu^2} \right) \tag{48e}$$

$$= (g-\mu) \left[\sum_{i=1}^{+\infty} i^2\, (1-g)^i + \frac{1}{\mu} \sum_{i=1}^{+\infty} i\, (1-g)^i \right] \tag{48f}$$

$$= (g-\mu) \left[\frac{(1-g)^2 + 1 - g}{g^3} + \frac{1-g}{\mu\, g^2} \right] \tag{48g}$$

$$= \frac{(g-\mu)\, (1-g)}{g^2} \left(\frac{2-g}{g} + \frac{1}{\mu} \right). \tag{48h}$$

Similarly:

$$s_2 = \sum_{i=1}^{+\infty} \sum_{j=1}^{i-1} i\, j\, \mu\, (g-\mu)\, (1-g)^j\, (1-g+\mu)^{i-j-1} \tag{49a}$$

$$= \mu\, (g-\mu) \sum_{j=1}^{+\infty} \sum_{i=j+1}^{+\infty} i\, j\, \left(1-g\right)^j\, (1-g+\mu)^{i-j-1} \tag{49b}$$

$$= \mu\,(g-\mu)\sum_{j=1}^{+\infty} j\,(1-g)^j\,\left(\frac{j}{g-\mu}+\frac{1}{(g-\mu)^2}\right) \tag{49c}$$

$$= \mu\,\left[\sum_{j=1}^{+\infty} j^2\,(1-g)^j + \frac{1}{g-\mu}\sum_{j=1}^{+\infty} j\,(1-g)^j\right] \tag{49d}$$

$$= \frac{\mu\,(1-g)}{g^2}\,\left(\frac{2-g}{g}+\frac{1}{g-\mu}\right) \tag{49e}$$

Hence, we have:

$$\mathbb{E}\left[N_{A_k}\,N_{H_k}\right] = \frac{1-g}{g^2}\,\left[2-g+\frac{(g-\mu)^2+\mu^2}{\mu\,(g-\mu)}\right] \tag{50a}$$

$$= \frac{g-g^2-g\,\mu+g^2\,\mu+\mu^2-g\,\mu^2}{g\,\mu\,(g-\mu)}\,. \tag{50b}$$

We are thus able to compute $\mathbb{V}\left[N_{A_k}, N_{H_k}\right]$:

$$\mathbb{V}\left[N_{A_k}, N_{H_k}\right] = \frac{g-g^2-g\,\mu+g^2\,\mu+\mu^2-g\,\mu^2}{g\,\mu\,(g-\mu)} - \frac{(1-\mu)\,(1-g+\mu)}{\mu\,(g-\mu)} \tag{51a}$$

$$= \frac{g-g^2-g\,\mu+g^2\,\mu+\mu^2-g\,\mu^2}{g\,\mu\,(g-\mu)} - \frac{g-g^2+g^2\,\mu-g\,\mu^2}{g\,\mu\,(g-\mu)} \tag{51b}$$

$$= \frac{\mu^2-g\,\mu}{\mu\,g\,(g-u)} \tag{51c}$$

$$= -\frac{1}{g}\,. \tag{51d}$$

Finally, this allows us to compute $\mathbb{V}\left[N_{A_k}-N_{H_k}\right]$:

$$\mathbb{V}\left[N_{A_k}-N_{H_k}\right] = \frac{1-\mu}{\mu^2}+\frac{1-g+\mu}{(g-\mu)^2}+\frac{2}{g}\,. \tag{52}$$

Hence, the distribution of S_N can be approximated by a normal distribution with mean $\frac{g-2\,\mu}{\mu\,(g-\mu)}$ and variance $\frac{1}{N}\left(\frac{1-\mu}{\mu^2}+\frac{1-g+\mu}{(g-\mu)^2}+\frac{2}{g}\right)$ according to the central limit theorem. In this case, the adversary succeeds if $S_N \leqslant -\frac{1}{N}$, which finally ensures, using this approximation:

$$p_{\text{success}} \approx \Phi\left(-\frac{\frac{g-2\,\mu}{\mu\,(g-\mu)}\,N+1}{\sqrt{N\,\left(\frac{1-\mu}{\mu^2}+\frac{1-g+\mu}{(g-\mu)^2}+\frac{2}{g}\right)}}\right)\,. \tag{53}$$

A.7 Theorem 7

Proof. The adversary's best strategy is to place the block containing the transaction at the beginning or at the end of their fork. By doing so, they can use

precedent or next blocks as part of their proof, and have only to provide the client with k additional valid blocks. Then, their best strategy is to try to mine $\mathsf{C}[f+1]'$, which takes them in average $\frac{1}{\mu}$ blocks, and then proceeding to mine $\mathsf{C}[f+2:f+k+2]'$. They have no choice but to sequentially mine all these blocks, more and more blocks being mined on the main chain meanwhile. As in subsection A.3, we consider the process of mining as a memoryless process, where at each round the adversary mines a block with probability μ. If they don't manage to mine this block, the honest miners mine a block on the main chain. Hence, the number of blocks being mined on the main chain while the adversary mines the k valid blocks is given by:

$$B = k + \max\left(\sum_{i=1}^{k} X_i - k, 0\right) \tag{54}$$

where each X_i follows a geometric distribution of parameter μ with support $\mathbb{N}$. Hence:

$$\mathbb{P}\left[\sum_{i=1}^{k} X_i \leqslant k\right] = \mathbb{P}\left[\bigcap_{i=1}^{k} (X_i \leqslant 1)\right] \tag{55}$$

which leads to:

$$\mathbb{P}\left[\sum_{i=1}^{k} X_i \leqslant k\right] = \left[\mu\,(2-\mu)\right]^k. \tag{56}$$

Furthermore, $\sum_{i=1}^{k} X_i$ follows a negative binomial distribution with parameters k and $1-\mu$. Hence, we have the following distribution for B:

$$\mathbb{P}\left[B = k+l\right] = \begin{cases} \left[\mu\,(2-\mu)\right]^k & \text{if } l = 0 \\ \binom{2\,k+l-1}{k+l}\,\mu^k\,(1-\mu)^{k+l} & \text{otherwise} \end{cases}. \tag{57}$$

Hence, $\mathbb{E}\left[B\right]$ seems to have no closed form. However, we can lower-bound B, and as such $\mathbb{E}\left[B\right]$ using:

$$B \geqslant \sum_{i=1}^{k} X_i \tag{58}$$

which leads to:

$$\mathbb{E}\left[B\right] \geqslant \frac{1-\mu}{\mu}\,k. \tag{59}$$

Hence, the following holds:

$$\mathbb{E}\left[C|\mu\right] \geqslant \left(\frac{1}{\mu} + \frac{1-\mu}{\mu}\,k + \frac{1}{\mu} - 1\right)\mu\,R - (1-\mu)\,R \tag{60}$$

which leads to:

$$\mathbb{E}\left[C|\mu\right] \geqslant \left[1 + k\,(1-\mu)\right]\,R. \tag{61}$$

Note that with probability $1 - [\mu\ (2 - \mu)]^k$, inequality 58 is actually an equality. Hence, we have:

$$\mathbb{E}\left[C|\mu\right] = \left[1 + k\,(1 - \mu)\right]\ R \tag{62}$$

with probability $1 - [\mu\ (2 - \mu)]^k$.

B Lemmas

B.1 Variable difficulty

Lemma 1. *Let d be a function from $[0\,;\,1]$ to $[0\,;\,1]$ which represents the cumulative difficulty of the protocol over time. FlyClient's sampling distribution over the block space is now given by:*

$$\forall x \in [0\,;\,1-\delta], s(x) = \frac{d'(x)}{[d(x) - 1]\ \ln(\delta)}. \tag{63}$$

Proof. Let $(a, b) \in [\![0\,;\,n\,(1-\delta)]\!]^2$ be two block heights, with $a < b$. The probability that a block in $\mathsf{C}[a : b]$ is sampled by FlyClient under a variable difficulty protocol is given by:

$$p_{a,b} = \int_{d\left(\frac{a}{n}\right)}^{d\left(\frac{b}{n}\right)} \frac{\mathrm{d}t}{(t-1)\ \ln(\delta)}. \tag{64}$$

Using integration by substitution, we have:

$$p_{a,b} = \int_{\frac{a}{n}}^{\frac{b}{n}} \frac{d'(x)}{[d(x) - 1]\ \ln(\delta)}\,\mathrm{d}x. \tag{65}$$

C Additional figures

C.1 Approximation of the sum of non-independent geometric variables by a normal distribution

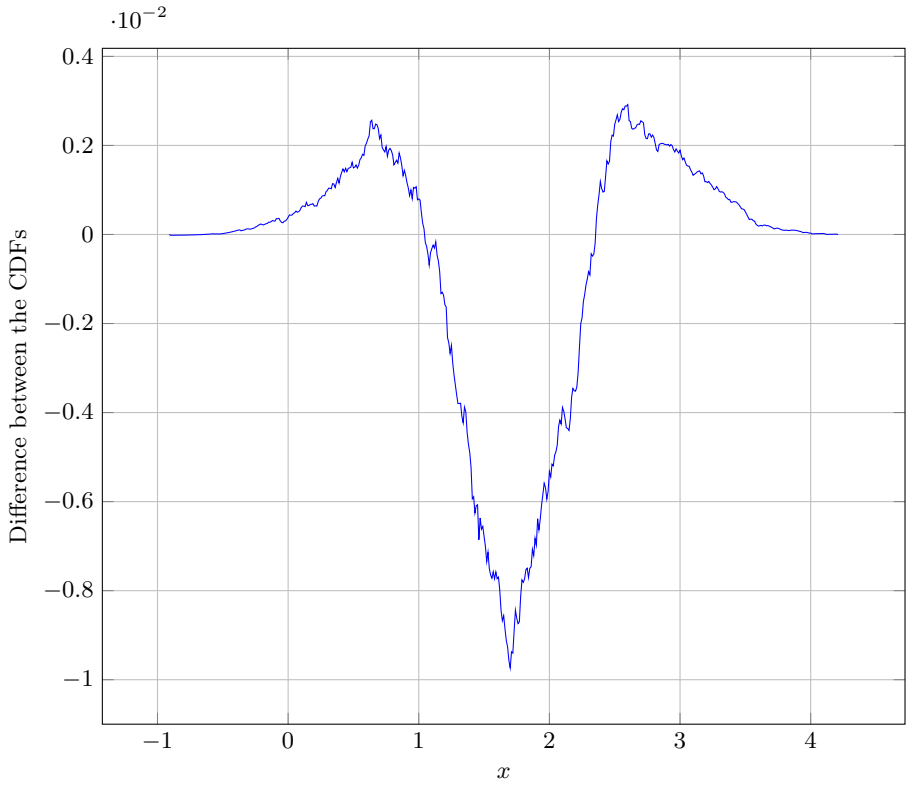

Fig. 18. Difference between the model CDF and the empiric CDF for $\mu = \frac{1}{5}$, $g = \frac{1}{2}$ and $N = 100$ using 100 000 samples.