Timelocked Bribing

Majid Khabbazian[*]
mkhabbazian@ualberta.ca
University of Alberta

Tejaswi Nadahalli[*]
tejaswin@ethz.ch
ETH Zürich

Roger Wattenhofer[*]
wattenhofer@ethz.ch
ETH Zürich

ABSTRACT

A Hashed Time Lock Contract (HTLC) is a central concept in cryptocurrencies where some value can be spent either with the preimage of a public hash by one party (Bob) or after a timelock expires by another party (Alice). We present a bribery attack on HTLC's where Bob's hash-protected transaction is censored by Alice's timelocked transaction. Alice incentivizes miners to censor Bob's transaction by leaving almost all her value to miners in general. Miners follow (or refuse) the bribe if their expected payoff is better (or worse). We explore conditions under which this attack is possible, and how HTLC participants can protect themselves against the attack. Applications like Lightning Network payment channels and Cross-Chain Atomic Swaps use HTLC's as building blocks and are vulnerable to this attack. Our proposed solution uses the hashpower share of the weakest known miner to derive parameters that make these applications robust against this bribing attack.

KEYWORDS

bitcoin, HTLC, bribe

1 INTRODUCTION

Bitcoin started the modern cryptocurrency revolution by removing trusted intermediaries and replacing them with a dynamic set of miners. These miners validate transactions and are paid by the system in the form of block rewards and also by transaction participants in the form of fees. Rational miners will always choose higher-fee transactions than lower-fee ones, and this behavior will get reinforced over time as block rewards decrease to zero [11]. This setup has often raised ([21] [18] [20]) the possibility of miners being bribed by transaction participants to favor one participant over the other. Typical bribing attacks envision the paying party (Alice) cheating the paid party (Bob) by Alice double-spending the same value in a separate transaction paying back to Alice. Miners are bribed by Alice to include the double-spending transaction in the blockchain by forking it and orphaning the block with the first transaction, thereby cheating Bob of the payment from the first transaction. These bribery attacks, however, operate at a block level because, to be cheated, Bob needs to be convinced that the first transaction is buried in the blockchain by k blocks (in Bitcoin, $k = 6$). Before this happens, Bob should ideally not honor the first transaction, but monitor the public Bitcoin blockchain. If a transaction where Alice double-spends the same bitcoins back to herself is seen, and Bob's transaction is abandoned in an orphaned block, Bob should not honor Alice's first transaction by not giving Alice the goods and services that were promised.

A more sophisticated concept of transactions exists where Bob *does* want Alice to pay the transaction value back to herself, but only after some time has elapsed. During this time, Bob reserves the option of getting paid himself from the same payment source. This complex transaction structure is the building block for financial contracts like escrows, payment channels, atomic swaps, etc. The required time delay is implemented using a blockchain artefact called *timelocks*. A rudimentary version of timelocks (nLocktime) was in the first Bitcoin implementation by Satoshi Nakamoto in 2009 [22]. More sophisticated timelocks that lock transactions, specific bitcoins, or specific script execution paths were added later [14] [26] [12]. Bitcoin script allows for timelocks to be combined with hashlocks in an OR condition to create a new kind of transaction called Hash Timelocked Transactions (HTLC). As we will see later, HTLC's open the possibility of transaction level bribing of miners where miners do not have to orphan mined blocks, but just have to ignore a *currently valid* transaction and wait for the timelocked bribe to become valid. Additionally, in this attack, the bribe is endogenous to the transactions and does not have to be implemented externally through public bulletin boards or other third party smart contracts. Bribery attacks that operate at a transaction level are far more insidious compared to block orphaning bribery attacks. Block orphaning attacks undermine the native cryptocurrency's trust with the larger community and could be detrimental to the briber's financial position in general. Transaction level bribery, on the other hand, targets specific contracts on the blockchain and could go unnoticed as the larger cryptocurrency system hums along.

1.1 HTLC

HTLC's are a type of smart contract that use preimage resistance of cryptographic hash functions, along with timelocks, to enable an escrow service. Say we have a buyer who has some bitcoin and wants to buy some goods/services from a seller. The buyer commits their bitcoin into a contract which is locked by an OR condition of:

- A cryptographic hash digest of a random secret preimage that the buyer knows, and will reveal to the seller once the buyer has possession of the goods/services. This sends the funds to an address the seller controls. The exchange of the preimage for the goods/services can be implemented in variety of ways, leading to different applications.
- A timelock after which the funds are sent back to the buyer. This is to ensure that the funds do not get locked in the contract if the seller aborts.

In Bitcoin's script-like pseudocode, an HTLC looks like this:

```
HTLC_TXN: {
  txid: HTLC_TXN_TXID
  vin: [{
    txid: SOURCE_TXN_ID that pays the buyer.
    scriptSig: <buyer's sig for SOURCE_TXN_ID>
  }]
  vout: [{
    value: <value>
    scriptPubKey:
      IF
```

[*]Authors are listed alphabetically.

```
        OP_HASH160 <digest> OP_EQUALVERIFY
          <seller_pubkey_1>
      OP_ELSE
        <delay> OP_CSV OP_DROP <buyer_pubkey_1>
      OP_ENDIF
      OP_CHECKSIG
  }]
}
```

This transaction is broadcast and is confirmed on the Bitcoin blockchain to a sufficient depth to be considered finalized. The seller then exchanges their goods and services for the preimage of the hash from the buyer. This exchange process is independent of the transaction itself. Each application that uses HTLC's has its own way of doing this exchange. For example, Atomic Swaps rely on another public blockchain to reveal the secret preimage. After the exchange is done, the seller will attempt to move the funds from HTLC_TXN to an address that the seller controls with a transaction like this:

```
SELLER_TXN: {
  txid: SELLER_TXN_TXID
  vin: [{
    txid: HTLC_TXN_TXID
    scriptSig: <seller_sig_1> <preimage> OP_TRUE
  }]
  vout: [{
    value: <value>
    scriptPubKey: <seller_pubkey_2 > OP_CHECKSIG
  }]
}
```

1.2 Bribing Attack

At this point, the buyer already has the goods/services for which the buyer commited the initial funds for. If the buyer acts in good faith and does nothing, there is no attack. If the buyer acts in bad faith, the buyer will try to censor the seller's transaction from being included in any future block. The buyer uses the refund arm of the HTLC_TXN and attempts to move the funds to *any* miner by leaving the output field empty:

```
BRIBE_TXN: {
  txid: BRIBE_TXN_TXID
  vin: [{
    txid: HTLC_TXN_TXID
    scriptSig: <buyer_sig_1> OP_FALSE
    sequence: <delay>
  }]
  vout: [{
    // Empty output. Entire amount
    // goes to the miner.
  }]
}
```

Note that the buyer can send an ϵ amount to themselves. This makes the bribe not just a griefing attack (where the attacker does not profit), but marginally profitable. Also note that SELLER_TXN and BRIBE_TXN spend the same UTXO and are inherently incompatible. If one of them is confirmed on the blockchain, the other becomes invalid.

Bitcoin's consensus rules govern what transactions can be included in a block by miners, but does not say anything about what transactions miners can or cannot ignore. It gives the benefit of the doubt to miners, allowing the possibility that miners have not seen a specific transaction because of network delays/failures. Miners could be (or not be) interested in a transaction because its fees are high (or low). In our attack scenario, miners sees SELLER_TXN and BRIBE_TXN at the same time. But as per the consensus rules, miners cannot include BRIBE_TXN immediately because it is timelocked. But crucially, there is no obligation to include the SELLER_TXN immediately either. As blocks go by, BRIBE_TXN becomes valid and can be included in the blockchain, and we have successfully censored SELLER_TXN and have gotten the goods and services for nothing.

In the following sections, we show how the two main applications of HTLC's: Lightning Payment Channels and Atomic Swaps, are both vulnerable to this bribing attack.

1.3 Payment Channels

Payment channels [13], [23] are a promising solution to the scalability problem in cryptocurrencies like Bitcoin and Ethereum, which have low transaction throughputs. Lightning Network's [23] payment channels rely on HTLC's to enforce the revocation of older commitment transactions. In our attack scenario, Alice and Bob have a payment channel that they have updated over time using many commitment transactions. Both Alice and Bob keep their own copy of the commitment transaction, where their copy can be broadcast by them, and will lock their side of the channel balance with an HTLC and the counterparty's side with a regular payment. This means that in the case of a channel closure, the broadcaster has to wait for his payment, but the counterparty can withdraw funds immediately. Without loss of generality, we can assume that in one such update (u_1), the entire channel balance was in Bob's favor, and Alice has zero balance in her favor. In a subsequent update (u_2), Alice delivers some goods/services to Bob, and after u_2, the entire channel balance is in Alice's favor and Bob has zero balance on his side of the channel. As a part of the Lightning Protocol, during u_2's negotiation, Bob gives Alice the preimage (p_1) of a hash that lets her punish him if u_1 ever makes it to the blockchain.

The briber (in our case, Bob) broadcasts an outdated commitment transaction u_1 (called Revoked Commitment Transaction in Lightning). This has one output which is an HTLC. He then follows it up by broadcasting the bribing transaction: BRIBE_TXN. Note that the BRIBE_TXN is timelocked and should be invalid till the timelock expires. The victim (Alice in our case), sees u_1 on the blockchain, and using her knowledge of the revocation preimage, sends the corresponding SELLER_TXN (called Breach Remedy Transaction in Lightning) to the pool of transactions to be included in the blockchain, Note that SELLER_TXN should be valid immediately as it has no timelock on it. But if all miners wait for the BRIBE_TXN's timelock to expire, and during that time ignore the SELLER_TXN, the bribing attack is successful. The amount that goes from the BRIBE_TXN to the miner does not matter to Bob because he already has the equivalent goods/services from Alice for that value. Therefore, he is bribing with what he has already spent.

Lightning Network uses HTLC's to also implement payment hops from, say, Alice to Bob through Carol - where Alice and Bob do not have a direct payment channel between each other, but both have a channel to Carol. HTLC's are used here to ensure that

Carol can use her channels to send funds from Alice to Bob without Carol's own funds being put at risk. Either the entire payment goes through from Alice to Bob through Carol (who gets the routing fees), or the entire payment is aborted, and all parties retain their own pre-payment balances. Using a series of messages [8], Alice, Bob, and Carol communicate using an off-chain protocol and negotiate a series of commitment transactions that each have an additional HTLC that sends the new payment from Alice to Bob through Carol. These HTLC's have a different payment specific secret preimage and its associated hash that locks the hashlock arm of the HTLC. They also have a lower timeout value (compared to the channel's timeout value) that refunds this particular payment back to the source in case any other node along the payment route aborts the payment. These hops do not affect the bribing attack model: an outdated commitment channels can still be broadcast by the briber and the victim has to respond.

1.4 Atomic Swaps

Atomic Swaps are a way to exchange cryptocurrencies between two separate public blockchain systems (say, between Bitcoin and Litecoin) without involving a trusted third party [16], [15]. TierNolan's classic Atomic Swap construction [3] relies on two `HTLC_TXN`'s to get around the trusted third party. Alice and Bob have their own `HTLC_TXN`'s in the blockchains whose assets they have. These `HTLC_TXN`'s will enable corresponding `SELLER_TXN`'s to the other party and `BUYER_TXN`'s to themselves. Alice initiates her side of the swap by publishing an HTLC on her blockchain which has a timelock of $2 \cdot t$ and hash of a secret preimage that only she knows. Bob accepts the swap by publishing his own HTLC on his blockchain with a timelock of $1 \cdot t$ and the same hash whose preimage he *does not* know. Alice then redeems Bob's HTLC by revealing her secret through a `SELLER_TXN` on Bob's blockchain. Bob's knowledge of this secret (by monitoring Bob's public blockchain) enables Bob to publish his own `SELLER_TXN` on Alice's blockchain, thereby completing the swap.

In the atomic swap described above, Alice can try to censor Bob's `SELLER_TXN` with her own `BRIBE_TXN` on her blockchain that lets her keep assets on Bob's blockchain, and leave most of her bribing profits on her own blockchain to miners. This way, Alice only profits if her attack succeeds, and has no possibility of a loss. Ideally, this should not be possible because Bob's `SELLER_TXN` is valid from the moment he gets to know of Alice's secret preimage, and Alice's `BRIBE_TXN` is invalid at that time. But if all miners are made aware of Alice's `BRIBE_TXN`, the bribing attack might succeed.

In atomic swaps, Bob would not agree to the swap if he sees Alice's opening HTLC being spent only by his `SELLER_TXN` and a `BRIBE_TXN` that leaves all the rewards to a miner. A "standard" atomic swap will instead have a `BUYER_TXN` that refunds the value back to the swap initiator. Alice can then send a followup `BRIBE_TXN` sending the value of this `BUYER_TXN` to the miner. For our purposes, if we have a `BUYER_TXN` and a `BRIBE_TXN` in sequence, we can consider it as a single `BRIBE_TXN`.

2 ANALYSIS

In this section, we analyze the parameters under which this bribing attack is successful. As Alice and Bob both have to agree on the HTLC for it to be valid, they can control these parameters to avoid the attack. The HTLC parameters are:

- T: denotes the number of blocks needed until the `BRIBE_TXN` becomes valid. This is the HTLC's timelock expressed in terms of number of blocks.
- f: fee offered by Alice to miners to confirm her `SELLER_TXN`.
- b: bribe offered by Bob to miners to confirm his `BRIBE_TXN`. Note that b is not explicitly called out in the transaction because all unclaimed outputs of a transaction go to the miner who confirms it. Typically, $b > f$.

There are parameters of the network that Alice and Bob do not control. These are the percentages of the total hashpower that identifiable miners control. Unidentifable miners are grouped in a catch-all group. Let there be n miners M_j, $1 \leq j \leq n$, each with a fraction p_j of the total hashpower.

2.1 Assumptions

- Miners are rational and choose the most profitable strategy on what transactions to include in their blocks while conforming to the consensus rules of Bitcoin. Their goal is to maximize expected payoff, and not mine altruistically. Rationality also implies that a miner will not chose a dominated strategy when they can choose one that is not.
- Relative hashpowers of miners is common knowledge. Currently, almost all Bitcoin blocks are mined by mining pools, and almost all of these blocks have an identifiable signature in the coinbase transaction that allows them to identify this relative share of hashpowers.
- The attacker and the victim of the bribery attack have no hashpower of their own.
- Timelocks are expressed in number of blocks, and we are thus operating in a setting where block generation is equivalent to clock ticks.
- Block rewards and fees generated by transactions external to our setting are constant and have no bearing on the attack itself.
- All miners can see timelocked transactions that are valid in the future. Currently, the most popular Bitcoin implementation, Bitcoin Core, does not allow timelocked transactions that are "valid in the future" to enter its pool. Consequently, it does not forward such transactions through the peer to peer network. This is not a consensus rule, but rather an efficiency gain whereby allowing only valid transactions to enter the pool and propagate across the peer to peer network reduces network and memory load. We assume that `SELLER_TXN` and `BRIBE_TXN` are visible to all miners immediately after they are broadcast by their respective parties. Also, some mining pools run "transaction accelerator" services where they cooperate with other mining pools to get visibility to transactions that pay an extra fee (on top of the blockchain fee). We assume that malicious buyers have access to such services.

2.2 Setting

We analyze this attack by modeling the sequence of blocks being mined as a (Markov) game, called the *bribing game*. A bribing game has n miners, and runs in $T + 1$ sequential stages. Stages represent periods between two mined blocks. In each stage, every miner has two possible actions: *follow* or *refuse* (corresponding to a miner excluding the SELLER_TXN from the miner's block template or not). After all miners play their action, a single miner is randomly selected as the leader of the stage. In other words, after all the miners have decided on their block template, a single miner wins the proof of work lottery and this miner's block extends the blockchain.

Let $B_1, B_2, \ldots, B_T$ be all the blocks that can include SELLER_TXN. Let B_{T+1} be the block that includes BRIBE_TXN. Note that BRIBE_TXN cannot be included in $B_1, B_2, \ldots, B_T$ as it's not valid then. Let $\mathcal{E}_{i,j}$ denote the event that miner j is selected as the leader of stage i. The events $\mathcal{E}_{i,j}$ are independent of each other and the actions taken by miners. $\mathcal{E}_{i,j}$ represents block B_i being mined by miner M_j. In addition, the *selection probability* of miner j for block i is given by:

$$\forall i, j \quad Pr(\mathcal{E}_{i,j}) = p_j,$$

which corresponds to the hashpower of miner M_j. Each stage is in one of two states: *active* and *inactive*. The game starts in an active stage (i.e., the first stage is active). Stage i, $i > 1$, becomes inactive if the leader of stage $i - 1$ had played the action *refuse* (correspond to including SELLER_TXN), or if stage $i - 1$ is already inactive. Therefore, if one stage becomes inactive, all the following stages become inactive. This intuitively makes sense because once SELLER_TXN is confirmed, it stays confirmed in subsequent blocks and more importantly, BRIBE_TXN is invalid after that. The payoffs for each stage i are determined by whether $1 \leq i \leq T$ or if $i = T + 1$.

- $1 \leq i \leq T$: If the leader plays *refuse*, the payoff is $f > 0$. If the leader plays *follow*, the payoff is 0. Non-leaders' payoff is always 0.
- $i = T + 1$: Leader's payoff is $b > 0$. Non-Leaders' payoff is 0.

Let us call a miner M_i *powerful* if $p_i \geq \frac{f}{b}$; otherwise we call M_i *weak*. Note that the bribing attack is successful if all miners follow the bribe (i.e., they always ignore SELLER_TXN). This corresponds to the strategy profile in which all miners play the action *follow* in all stages. Without loss of generality, there are two possible distributions of hashpowers among miners:

- All miners are powerful; i.e., $p_i \geq \frac{f}{b}$ for $1 \leq i \leq n$.
- At least one miner is weak; i.e, $\exists p_i$ s.t. $p_i < \frac{f}{b}$ for $1 \leq i \leq n$.

In the next sections, we analyze both of these distributions.

2.3 All miners are powerful

Lemma 2.1. *If all miners are* powerful *(i.e., $p_i \geq \frac{f}{b}$ for $1 \leq i \leq n$), then the strategy profile in which every miner plays* follow *in all stages is an equilibrium.*

Proof. Consider Miner i, and assume that all other miners follow the bribe in all stages. We show that following the bribe in all stages is the best response for Miner i as well. If Miner i follows the bribe in all stages, they will earn $p_i \cdot b$ in expectation. This is because, when all miners play *follow* in all stages, stage $T + 1$ will be active, and its leader, which is Miner i with probability p_i, earns b.

If Miner i plays *refuse* with non-zero probability in at least one stage. Let $x > 0$ be the probability that stage $T + 1$ becomes inactive as the result of Miner i's actions. In other words, x is the probability that Miner i plays *refuse* in a Stage $1 \leq i \leq T$ in which they are selected as the leader. Note that other miners cannot make stage $T+1$ inactive as they always play *follow* and only Miner i is including SELLER_TXN in their block template. The expected payoff of Miner i is, therefore, $x \cdot f + (1 - x) \cdot p_i \cdot b$, which is not more than $p_i \cdot b$, because $p_i \geq \frac{f}{b}$ and $x > 0$. □

Note that when all miners are powerful, the equilibrium shown in Lemma 2.1 (which favours bribery) exists no matter how large T is. As of this writing, the average fees for Bitcoin transactions since the beginning of 2019 is around 0.00003 BTC. The average balance held by a lightning channel is 0.026 BTC. If we use these values, we get the equilibrium stated in Lemma 2.1 exists if each miner has over 0.115% of the total hash power of the entire Bitcoin network. Due to the permissionless and anonymous nature of Bitcoin, however, we can never be sure that the weakest miner has a hash power above 0.115% of the total hash power. However, we can inspect the Bitcoin blockchain to guesstimate the distribution of hashpowers among known mining pools, and recommend channel parameters based on that. We treat this in more detail in section 3. Next, we consider the case where at least one miner is weak. We show that, in this case, the value of T matters.

2.4 One miner is weak

Recall that when a stage becomes inactive, all its followup stages become inactive as well. Moreover, all miners receive zero payoff in an inactive stage, irrespective of what they play. Note that, for every miner (weak or powerful), playing *follow* at state $T + 1$ is the strictly dominant strategy if stage $T + 1$ is active. This is because the expected payoff of a miner in an active stage $T + 1$ is $p_i b$ if they play *follow*, and $p_i f$ (which is smaller than $p_i b$) if they play *refuse*. In the next lemma, we show that in active stages other than stage $T + 1$, playing *refuse* is the strictly dominant strategy for weak miners.

Lemma 2.2. *In any active stage i, $1 \leq i \leq T$, playing* refuse *is the strictly dominant strategy for any weak miner.*

Proof. A miner earns b if stage $T + 1$ is active and this miner is selected as the leader of stage $T + 1$. Therefore, the probability that a Miner j earns b is at most p_j. From the definition of weakness, for Miner j, we have $p_j \cdot b < f$. So, if stage $T + 1$ is active, the weak miner gets an expected payoff less than f. Additionally, in stages $< T$, the probability that a miner earns f is strictly less than one, because, no matter how large T is, there is always a non-zero chance that the miner never get selected as a leader. Therefore, across all stages up to and including stage $T + 1$, the expected payoff of a weak miner is always strictly less than f.

Assume Miner j is weak (i.e., $p_j < \frac{f}{b}$), and plays *follow* in an active stage i, $1 \leq i \leq T$. We now show that playing *refuse* in stage i will improve her payoff. Suppose Miner j plays *refuse* instead of *follow* in the active stage i. If j is not selected as the leader of stage i, then the game remains the same as the case where j played *follow*. If j is selected as the leader, however, they will earn f. This is an improvement over the *expected payoff* of Miner j from the previous paragraph, which is strictly less than f.

□

2.5 The elimination of dominated strategies

By Lemma 2.2, playing *refuse* is the strictly dominant strategy for every weak miner; any other strategy is strictly dominated. Hence, we can simplify the analysis of the bribing game by eliminating strictly dominated strategies. Let us call a bribing game *safe* if after eliminating strictly dominated strategies, the only action left for each miner (strong or weak) in stage one is to play *refuse*. If every miner plays *refuse* in stage one, the game is effectively over as other stages become inactive after that (with `BRIBE_TXN` being invalid after stage one).

By Lemma 2.1, if all the miners are powerful, the bribing game is not safe no matter how large T is. By the next theorem, however, the game is safe if there is at least one weak miner, and T is large enough.

THEOREM 2.3. *Suppose there is at least one weak miner, and*

$$T > \frac{\log \frac{f}{b}}{\log(1-p_w)}, \tag{1}$$

where p_w is the sum of the selection probabilities of weak miners. Then, the bribing game is safe.

PROOF. By Lemma 2.2, playing *refuse* is the strictly dominant strategy for every weak miner in each stage i, $1 \le i \le T$. By eliminating the dominated strategies of weak miners, we get a smaller game in which weak miners play *refuse* in every stage i, $1 \le i \le T$.

Consider a powerful miner u, and suppose u plays *follow* in stage 1. Let α be the probability that stage $T+1$ will be active. Since weak miners only play *refuse* in the first T stages, we get

$$\begin{aligned}\alpha &\le (1-p_w)^T \\ &\le (1-p_w)^{\frac{\log\left(\frac{f}{b(1-p_w)}\right)}{\log(1-p_w)}} \\ &= \frac{f}{b(1-p_w)},\end{aligned}$$

where $(1-p_w)^T$ is the probability that no weak miner is selected as a leader in the first T stages. Thus, the expected payoff of u at stage $T+1$ is less than

$$\frac{f}{b(1-p_w)} \cdot (1-p_w).b = f,$$

where $\frac{f}{b(1-p_w)}$ is an upper bound on the probability that stage $T+1$ is active, and $(1-p_w)$ is an upper bound on the probability that u is selected as the leader of stage $T+1$. Note that the probability that u earns f prior to stage $T+1$ is strictly less than one. Therefore, at the beginning of stage 1, the expected payoff of u is strictly less than f. Now, if u plays *refuse* (instead of *follow*) in the first stage, we will have two possibilities. First possibility is that u is selected as the leader of stage 1, in which case u earns f, which is strictly more than its expected payoff. In the second possibility where u is not selected as the leader of stage 1, the game remains identical to the original case were u plays *follow*. This implies that u is better off playing *refuse* in the first stage, which concludes the proof. We remark that this result does not imply that u is better off playing *refuse* in every stage. In fact, as the game proceeds to new stages, the expected payoff of u can change, and u may choose to play *follow*. □

A bribing game with parameters f and b may be safe for a significantly smaller T than what is given in Theorem 1. In its proof, we eliminated only strictly dominated strategies of weak miners. In principle, we can continue the process by eliminating strictly dominated strategies of powerful miners. To do so, we can first sort the powerful miners according to their selection probabilities. Starting with the powerful miner with the smallest selection probability, we can calculate the minimum number of initial stages in which the miner is strictly better off playing *refuse*. We then eliminate the strictly dominated strategies of that miner, and move to the next powerful miner. At the end of this iterated elimination process, if all miners play *refuse* in the first stage, then the game is proven to be safe.

This algorithm is described formally Algorithm 1. The `FIND_T` procedure receives as input a list of mining hashpowers (leader selection probabilities), and the values of parameters f and b. As output, it returns the lowest value of T such that all miners refuse the bribe in the first stage of the game. It uses the inner procedure `CALCULATE_BRIBERY_MATRIX` to determine the behavior of more powerful miners at each block when less powerful miners' strategies get dominated.

Example (Table 1): Let's take the case of 4 miners with hashpower shares $\mathbb{P} = [0.1, 0.2, 0.3, 0.4]$, $f = 11, b = 100$. Applying Theorem 2.3, we get an upper bound of T to be 21. Running the procedure `CALCULATE_BRIBERY_MATRIX` returns the matrix shown in Table 1, with "1" standing for *refuse* and "0" standing for *follow*. We now go through the actions of each miner.

Table 1: Bribery Matrix Example

Blocks	0.1	0.2	0.3	0.4
Block #1	1	1	1	1
Block #2	1	1	1	1
Block #3	1	1	1	1
Block #4	1	1	1	1
Block #5	1	1	1	1
Block #6	1	1	1	1
Block #7	1	1	1	1
Block #8	1	1	1	1
Block #9	1	1	1	1
Block #10	1	1	1	1
Block #11	1	1	1	1
Block #12	1	1	1	1
Block #13	1	1	1	1
Block #14	1	1	1	1
Block #15	1	1	1	1
Block #16	1	1	0	0
Block #17	1	0	0	0
Block #18	1	0	0	0
Block #19	1	0	0	0
Block #20	1	0	0	0
Block #21	1	0	0	0

```
procedure CALCULATE_BRIBERY_MATRIX($\mathbb{P}, f, b, T$)
    $\mathbb{B} \leftarrow [\,][\,]$    ▷ Bribery Matrix where B[i][j] represents whether $miner_i$ follows the bribe at $block_j$
    for $i \leftarrow 0$ to $length(\mathbb{P})$ do
        if $\mathbb{P}[i] < f/b$ then
            $\mathbb{B}[i] \leftarrow \underbrace{[1, 1, ...1]}_{T}$
        else
            $\mathbb{B}[i] \leftarrow \underbrace{[0, 0, ...0]}_{T}$
            for $t_i \leftarrow 1$ to $T$ do
                $P_h \leftarrow 1$
                for $t_j \leftarrow 1$ to $t_i$ do
                    $sum \leftarrow 0$
                    for $j \leftarrow 0$ to $i$ do
                        $sum \leftarrow sum + \mathbb{B}[j][t_j] \cdot \mathbb{P}[j]$
                    $P_h \leftarrow P_h * (1 - sum)$
                $expected\_bribe = P_h * \mathbb{P}[i] * b$
                if $f > expected\_bribe$ then
                    $\mathbb{B}[i][t_i] = 1$
    return B

procedure FIND_T($\mathbb{P}, f, b$)    ▷ P is the array of miners' hashpowers
    assert(at least 1 value in $\mathbb{P} > f/b$)
    $\mathbb{P} = sorted(\mathbb{P})$    ▷ Ascending
    $T = \lceil \frac{\log \frac{f}{b}}{\log(1-p_w)} \rceil$    ▷ From Theorem 2.3
    $\mathbb{B}$ = CALCULATE_BRIBERY_MATRIX($\mathbb{P}, f, b, T$)
    for $i \leftarrow 1$ to $T$ do
        for $j \leftarrow 0$ to $length(\mathbb{P})$ do
            if $\mathbb{B}[j][i] == 0$ then
                return $T - (i - 1)$
    return $T$
```

Algorithm 1: Iterated Removal of Dominated Strategies

The miner with hashpower 0.1 (p_0) will play *refuse* at every block because we have $T > \frac{\log \frac{f}{b}}{\log(1-p_w)}$. The miner with hashpower 0.2 (p_1) will play *refuse* as long as the expected bribe (payable at $T+1$) calculated at a particular block is lower than the fees that they would earn if they mine that block. In this case, $(1-p_w)^t \cdot p_1 \cdot b < f$ till $t = 6$ for values of $f = 11, b = 100, p_w = 0.1$. This means that p_1 will start playing *follow* as we get closer to $t = T$ (specifically when we are 5 blocks away from T). The miner with hashpower 0.3 (p_3) will play *refuse* along similar lines, by looking at the actions of miners p_0 and p_1 over the different blocks. One thing to notice is that at block #16, p_2 will act assuming that p_0 and p_1 will both play *refuse*. At block #17, p_2 will act assuming that p_0 will play *refuse* and p_1 will play *follow*. This is implemented in the algorithm by using the 0's and 1's in the bribery matrix and using them as factors in line #13 of the `CALCULATE_BRIBERY_MATRIX` procedure. This way, on line #13, we only use miners who play *refuse* at each block to calculate the expected bribe.

In the main procedure `FIND_T`, we then find the last block in which all miners play *refuse* and return that as the result. In the real world, we can give a 5-6 block cushion on top of this, and it will still be significantly lower than the upper bound of T.

3 SOLUTIONS

In the introduction, we pointed out that the two main applications of HTLC's: Lightning Channels and Atomic Swaps, are both vulnerable to this bribing attack. In this section, we first analyze the Bitcoin blockchain to get an estimate of the hashpower share of known mining pools. This lets us find parameters that can harden the HTLC constructions in each of these applications such that they are not vulnerable to the bribing attack. In the case of Atomic Swaps, to use these parameters, we propose a modification to the classic atomic swap protocol.

3.1 Mining Pools and their Hashpower Shares

We try to find the weakest known miners in the Bitcoin ecosystem by analyzing the miners of the 16000 blocks from Block #601000. We know the coinbase transaction indicators of larger mining pools. Using these, we can attribute mined blocks to known mining pools. Looking at these blocks, we can estimate each of these mining pools' share of the total hashpower based on how many blocks they have mined. Mining pools and their hashpower shares are shown in Table 2. We see that the weakest known pools are under 1% of the total hashpower, and this leads to our proposed fixes for both Lightning Channels and Atomic Swaps.

Table 2: Hashpower of 16000 blocks from block #601000

Mining Pool	Hashpower
PoolIn	16.8562%
F2Pool	15.1438%
BTC.com	13.2188%
AntPool	10.4563%
ViaBTC	6.3812%
Unknown	5.3438%
Huobi	4.7062%
58COIN	4.5375%
SlushPool	3.8625%
BTCTOP	3.6875%
BytePool	3.6812%
BitFury	3.0438%
CN6T	2.7625%
OKEX	2.3438%
OKPool	1.3375%
NovaBlock	1.1000%
SpiderPool	0.4562%
Bitcoin.com	0.3875%
NCKPool	0.2687%
UkrPool	0.2062%
Taal.com	0.2062%
BitclubNetwork	0.0063%
KanoPool	0.0063%

3.2 Lightning

In the Lightning Network specifications (specifically, from Bolt 2 [7]), we have the following parameters:

- *channel_reserve_satoshis*: Each side of a channel maintains this reserve so it always has something to lose if it were to try to broadcast an old, revoked commitment transaction. Currently, this is recommended to be 1% of the total value of the channel. This is the amount that the cheated party can utilize as extra fees without dipping into their own side of the channel.
- *to_self_delay*: This is the number of blocks that the counterparty's self outputs must be delayed in case a channel closes unilaterally from the counterparty's side. In one popular Lightning client: c-lightning [9], this is set by default to 144 blocks (approximtely 1 day). In another popular Lightning client: LND [10], it is scaled in a range from 1 day to 14 days based on the channel value.

We do not find any documented reasons on why these important parameters are set the way they are. Based on the analysis from Sections 2.4 and 2.5, and the distribution of hashpowers, we can formulate what these values ought to be. First, we note that *channel_reserve_satoshis* on the victim's side of this bribing attack can be used by the victim to increase their fees to thwart the attack. We posit that *channel_reserve_satoshis* being at 1% is reasonable, given that there are many known miners whose hashpower is less than 1% of the total hashpower of all miners. If it were lower than, say, 0.03%, as per Section 2.3, the channel would be always vulnerable to this bribing attack.

We then set $\frac{f}{b}$ to be 0.01, and calcualte the total *weak* hashpower to be 0.0145 (from Table 2). Based on Theorem 2.3, we get $T > 316$ blocks. This is substantially larger than the suggested default of *to_self_delay* at 144 blocks. So, if the channel operator is paranoid, they can set *to_self_delay* to this higher value of 316. We can plug in the hashpowers from Table 2 into Algorithm 1, with $f = 1$ and $b = 100$ and we get a value of $T = 35$ blocks. If the channel operator is *#reckless* and wants to eliminate strictly dominated strategies of stronger miners, they can open channels with this much lower timelock value.

3.3 Atomic Swaps

Atomic Swaps that have Bitcoin on one side need to take Bitcoin's block time of 10 minutes into account. Even if the other blockchain in question (say Litecoin) has faster block generation, till Bitcoin's transactions are not confirmed, the atomic swap in question cannot be considered executed. Commercial platforms like Komodo [4] use 15,600 seconds (26 blocks) as the HTLC's timelock value when they setup swaps between Bitcoin like currencies or ERC-20 style tokens. Other works [15], [24], [25] have suggested that a timelock period of 1 day (144 blocks) is a good default.

Based on Theorem 2.3, we get $\frac{f}{b} = 0.68$ at $T = 26$ blocks and $\frac{f}{b} = 0.122$ at $T = 144$ blocks. A fee to bribe ratio of 0.68 (for $T = 26$ blocks) is quite high. This suggests that $T = 26$ blocks does not provide enough security for reasonable values of fee to bribe ratios. At 144 blocks, we have a reasonable fee to bribe ratio of 0.122.

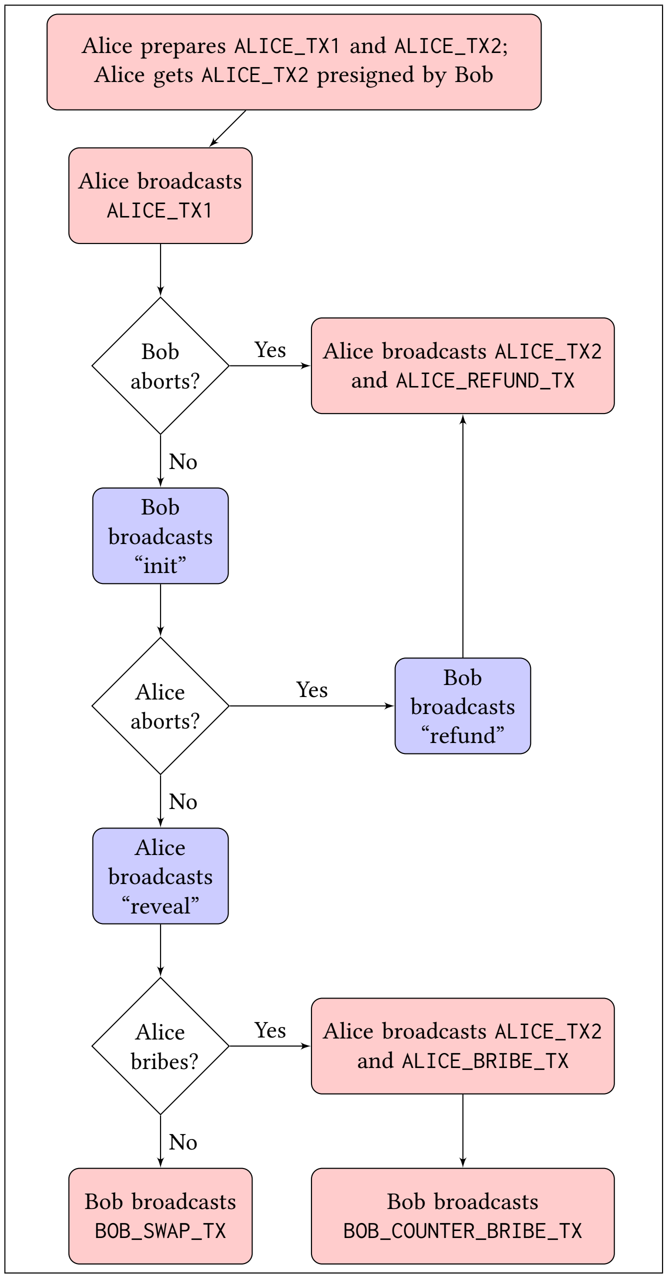
Unlike Lightning channel's *channel_reserve_satoshis*, due to its inherently asymmetric nature, there is no simple way to encode this extra fees in the atomic swap itself. Alice has to convince Bob upfront that she will not attempt the bribing attack when it is Bob's turn to redeem his side of the swap. One way of achieving this is for Bob to offer a lower value than what Alice wants. This way, if Alice attempts the bribery attack, Bob can increase his `SELLER_TXN` fees to the amount dictated by Theorem 2.3 or Algorithm 1. But if Alice does not attempt to bribe, this atomic swap setup is unfair to her as she is getting a lower value from Bob than what she is offering to Bob.

To solve this, we present an extension to the classic Atomic Swap protocol that allows a way for Alice to include extra fees in the swap for Bob to use to "counter-bribe" *only* if Alice attempts to bribe.

3.3.1 Risk Free Atomic Swap. Here, as with the classic protocol, Alice creates a (random) secret preimage and hashes it to get her "locking string". Alice creates a transaction that commits her swap amount such that Bob can claim this amount only if he knows the preimage. The "refund" part of this transaction, instead of sending the amount back to Alice after a timelock, sends it to a multisig controlled by both Alice and Bob. Alice also creates a second transaction that uses this multisig controlled output as its first input, and another unrelated input from Alice which adds the extra fees required to make the swap risk free. The total output of this second transaction is sent to Bob *only* if he has the secret preimage, or to Alice after a timelock. This pair of transactions is created by Alice; the second transaction is pre-signed by Bob and needs to be held by Alice before she broadcasts the first transaction.

Based on whether Alice or Bob abort the swap, or Alice bribes miners, a combination of the following transactions will be broadcast on the first blockchain by either Alice or Bob as depicted by the flow chart in Figure 2. Transactions on the second blockchain are unchanged from the classic Atomic Swap protocol. In the flow chart, they are represented as just "init", "refund", and "reveal". Transactions of the first and second blockchains are in the red and blue boxes respectively.

```
ALICE_TX1: {
  txid: ALICE_TX1_TXID,
  vin: [{
    txid: PREV_TX1_TXID // Pays amount X to Alice
    scriptSig: <Alice's sig>
  }]
  vout: [{
    value: X
    scriptPubKey:
      IF
        OP_HASH160 <digest> OP_EQUALVERIFY
        <bob_pubkey_first_exit> OP_CHECKSIG
      OP_ELSE
        2 <alice_pubkey_1> <bob_pubkey_1> 2
        OP_CHECKMULTISIG
      OP_ENDIF
  }]
}


ALICE_TX2: {
  txid: ALICE_TX2_TXID,
  vin: [{
    txid: ALICE_TX1_TXID
    scriptSig: 0 <alice_sig_1> <bob_sig_1> OP_FALSE
  }, {
    txid: PREV_TX2_TXID // Pays F to Alice
    scriptSig: <Alice's sig>
  }]
  vout: [{
    value: X + F
    scriptPubKey:
      IF
        OP_HASH160 <digest> OP_EQUALVERIFY
        <bob_pubkey_second_exit>
      OP_ELSE
        `to_alice_delay`
        OP_CSV
        OP_DROP
        <alice_pubkey_2>
      OP_ENDIF
      OP_CHECKSIG
  }]
}

ALICE_BRIBE_TX: {
  txid: ALICE_BRIBE_TXID
  vin: [{
    txid: ALICE_TX2_TXID
    scriptSig: <alice_sig_2> OP_FALSE
    sequence: `to_alice_delay`
  }]
  vout: [{
    value: 0 // Leaves X + F as bribe to miners.
  }]
}

ALICE_REFUND_TX: {
  txid: ALICE_REFUND_TXID
  vin: [{
    txid: ALICE_TX2_TXID
    scriptSig: <alice_sig_2> OP_FALSE
    sequence: `to_alice_delay`
  }]
  vout: [{
    value: X + F // Refund to Alice
    scriptPubKey: <alice_pubkey_refund > OP_CHECKSIG
  }]
}

BOB_SWAP_TX: {
  txid: BOB_SWAP_TXID
  vin: [{
    txid: ALICE_TX1_TXID
    scriptSig: <bob_sig_first_exit> <preimage> OP_TRUE
  }]
  vout: [{
    value: X // No extra fees for Bob
    scriptPubKey: <bob_pubkey_swap> OP_CHECKSIG
  }]
}

BOB_COUNTER_BRIBE_TX: {
  txid: BOB_COUNTER_BRIBE_TXID
  vin: [{
    txid: ALICE_TX2_TXID
    scriptSig: <bob_sig_second_exit> <preimage> OP_TRUE
  }]
  vout: [{
    value: X // Leaves F to the miners
    scriptPubKey: <bob_pubkey_swap> OP_CHECKSIG
  }]
}
```


Alice prepares ALICE_TX1 and ALICE_TX2; Alice gets ALICE_TX2 presigned by Bob
Alice broadcasts ALICE_TX1
Bob aborts?
Yes
Alice broadcasts ALICE_TX2 and ALICE_REFUND_TX
No
Bob broadcasts "init"
Alice aborts?
Yes
Bob broadcasts "refund"
No
Alice broadcasts "reveal"
Alice bribes?
Yes
Alice broadcasts ALICE_TX2 and ALICE_BRIBE_TX
No
Bob broadcasts BOB_SWAP_TX
Bob broadcasts BOB_COUNTER_BRIBE_TX

Algorithm 2: Risk Free Atomic Swap; red = first blockchain; blue = second blockchain;

The second blockchain transactions are unchanged from the classic Atomic Swap protocol. This is because, unlike Lightning channels, in an Atomic Swap, only the swap initiator (in this case, Alice) can attempt to cheat by bribing the first blockchain's miners after she claims her side of the swap on the second blockchain. So, the modification to the classic swap that brings in the "counter bribe fees" is done only on Alice's side of the swap as shown above with the intermediate multisig.

4 RELATED WORK

There have been many proposed attacks on cryptocurrencies which attempt to censor specific transactions. One of the earlier ones discussed on bitcointalk.org was that of *feather forking* [21]. In this attack, a miner wants to censor a specific transaction and announces on some public bulletin board that they will not add blocks on top of any block that contains this specific transaction. If this miner has a reasonable chance of getting a block, other rational miners will follow them instead of mining "normally" and thereby getting the fees of the censored transaction. This attack has been used in other contexts as well [19]. Feather forking relies on censoring a transaction with the cooperation of a miner by forking the blockchain away from a block that contains the said transaction. In our attack, the attacker is not necessarily a miner, and does not have to convey any out of band information about their intentions to other miners. Just releasing the bribing transaction into the network constitutes the bribe, and if all miners see this transaction, it can be effective based on the numerical parameters that govern it.

Miners can be incentivized to fork the Bitcoin blockchain with "Whale Transactions" [18]. Here, the attacker waits for a target transaction to be confirmed to a sufficient depth to get the corresponding goods and services from their victim. After that, the attacker tries to fork the blockchain by successively broadcasting transactions that have high fees (whale transactions) and also reverse the target transaction. These whale transactions are then included in blocks of the blockchain fork that rational miners might follow. The authors evaluate the relationship between confirmation depth, the attacker's secret mining lead, the attacker's hashpower, the whale transaction fees and whether these attacks are profitable. External smart contracts on platforms like Ethereum can be used [20] to incentivize Bitcoin miners to abandon the honest blockchain suffix and mine on top of a briber's fork.

Most of these attacks rely on attackers being able to incentivize rational miners to orphan a reasonable length suffix of the blockchain. If done after the primary transaction has been thought confirmed by the victim, the attack succeeds. Given that most proof-of-work cryptocurrencies have a probabilistic notion of finality, these attacks are feasible. On the other hand, Bitcoin has seen fewer and fewer orphan blocks over time [6], and the possibility of this kind of attack is considerably lower now than they were in, say, 2015.

On the other hand, blocking specific transactions (and not orphaning entire blocks) needs the censorship details to be distributed to all miners. One approach involves the use of external smart contract platforms [27] [17]. In these attacks, for example, Bitcoin transactions could be censored using smart contracts published on the Ethereum platform. The attacker would create an Ethereum smart contract that has guaranteed rewards if a miner follows a specific block template of transactions (that excludes the victim's transaction). There are many other flavors of these attacks which make use of the full potential of smart contracts to create proper incentives for miners to deviate from the normal mining protocol.

Another class of censorship attacks in Bitcoin target specific transactions in the mempool. These attacks force the victim to monitor the "global mempool" along with the blockchain. Most interested parties operate their own Bitcoin nodes, and monitor their local mempool for attacks. In Transaction Pinning [2], an attacker chains multiple low fee-rate transactions to a target transaction to make the package unprofitable to mine. The victim can use CPFP carve-outs [1] to bump up the fee-rate of the censored transaction and still get it confirmed by a miner. To enable this, Lightning Channels will allow so called "anchor outputs" [5] to let either counterparty bump up their fees.

5 CONCLUSION

In this work, we observe that HTLC's are vulnerable to an "in-band" bribing attack where the HTLC initiator (buyer, in our case) can receive goods and services offline and then bribe their way out of paying the counterparty (seller, in our case). This bribe can only work if the "time value" of waiting for the bribe is worthwhile for all miners. A rather self-evident observation is that when the timelock on the bribe expires and the bribe transaction is still valid, it will be claimed in the immediate next block as the fee on it is considerably higher than normal transaction fees. Additionally, stronger miners are likely to mine any specific block - and therefore more likely to mine the block in which the bribe is valid and available. Therefore, we posit that weaker miners will ignore the bribe altogether and will attempt to mine the seller's transaction while the timelock holds and the fee on the seller's transaction is good enough. This leads us to the relationship between the fee to bribe ratio and the distribution of miners' hashpowers. Based on this analysis, we propose Lightning Channel parameters that make them resistant to this kind of bribing attack. In Atomic Swaps, our analysis also proposes a fee for the victim to safeguard themselves. To enable that, we propose a modification to the classic Atomic Swap protocol that can bring in this fee into the swap and still keep the it fair for both parties.

REFERENCES

[1] [n.d.]. CPFP Carve-out. https://bitcoinops.org/en/topics/cpfp-carve-out/.
[2] [n.d.]. Transaction Pinning. https://bitcoinops.org/en/topics/transaction-pinning/.
[3] 2013. Atomic Swaps. https://bitcointalk.org/index.php?topic=193281.msg2224949.
[4] 2018. Atomic Swaps Explained: The Ultimate Beginner's Guide. https://komodoplatform.com/atomic-swaps/. [Accessed: 2020-05-07].
[5] 2019. Anchor Outputs. https://github.com/lightningnetwork/lightning-rfc/pull/688.
[6] 2019. An orphan block on the bitcoin (BTC) blockchain. https://en.cryptonomist.ch/2019/05/28/orphan-block-bitcoin-btc-blockchain/.
[7] Bolt Authors. [n.d.]. Lightning Network Specifications, Bolt 2. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md. [Accessed: 2020-05-07].
[8] Bolt Authors. [n.d.]. Lightning Network Specifications, Bolt 3. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md. [Accessed: 2020-05-07].
[9] C-Lightning authors. [n.d.]. c-lightning - a Lightning Network implementation in C. https://github.com/ElementsProject/lightning. [Accessed: 2020-05-07].
[10] LND authors. [n.d.]. LND: The Lightning Network Daemon. https://github.com/lightningnetwork/lnd. [Accessed: 2020-05-07].
[11] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. 2015. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 104–121.
[12] BtcDrak, Mark Friedenbach, and Eric Lombrozo. 2015. BIP112: CHECKSEQUENCEVERIFY. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki [Accessed: 2020-05-07].
[13] Christian Decker and Roger Wattenhofer. 2015. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*. Springer, 3–18.

[14] Mark Friedenbach, BtcDrak, Nicholoas Dorier, and kinoshitajona. 2015. BIP68: Relative lock-time using consensus-enforced sequence numbers. https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki [Accessed: 2020-05-07].

[15] Runchao Han, Haoyu Lin, and Jiangshan Yu. 2019. On the Optionality and Fairness of Atomic Swaps. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT '19)*. Association for Computing Machinery, New York, NY, USA, 62–75. https://doi.org/10.1145/3318041.3355460

[16] Maurice Herlihy. 2018. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. ACM, 245–254.

[17] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gazi, Sarah Meiklejohn, and Edgar Weippl. 2019. Pay-To-Win: Incentive Attacks on Proof-of-Work Cryptocurrencies. Cryptology ePrint Archive, Report 2019/775. https://eprint.iacr.org/2019/775.

[18] Kevin Liao and Jonathan Katz. 2017. Incentivizing blockchain forks via whale transactions. In *International Conference on Financial Cryptography and Data Security*.

[19] Antonio Magnani, Luca Calderoni, and Paolo Palmieri. 2018. Feather forking as a positive force: incentivising green energy production in a blockchain-based smart grid. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. ACM, 99–104.

[20] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. 2018. Smart contracts for bribing miners. Cryptology ePrint Archive, Report 2018/581. https://eprint.iacr.org/2018/581.

[21] Andrew Miller. 2013. Feather-forks: enforcing a blacklist with sub-50% hash power. https://bitcointalk.org/index.php?topic=312668.0.

[22] Satoshi Nakamoto. 2009. Bitcoin Core Source Code, Version 0.1.0. https://bitcointalk.org/index.php?topic=68121.0.

[23] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments.

[24] BitMEX Research. [n.d.]. Atomic Swaps and Distributed Exchanges: The Inadvertent Call Option. https://blog.bitmex.com/atomic-swaps-and-distributed-exchanges-the-inadvertent-call-option/. [Accessed: 2020-05-07].

[25] Dan Robinson. 2019. HTLCs Considered Harmful. https://cyber.stanford.edu/sites/g/files/sbiybj9936/f/htlcs_considered_harmful.pdf. [Accessed: 2020-05-07].

[26] Peter Todd. 2014. BIP68: CHECKLOCKTIMEVERIFY. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki [Accessed: 2020-05-07].

[27] Fredrik Winzer, Benjamin Herd, and Sebastian Faust. 2019. Temporary Censorship Attacks in the Presence of Rational Miners. In *IEEE Security & Privacy on the Blockchain (IEEE S & B)*. https://eprint.iacr.org/2019/748.