MAD-HTLC: Because HTLC is Crazy-Cheap to Attack

Itay Tsabary  
Technion, IC3  
sitay@campus.technion.ac.il

Matan Yecheili  
Technion, IC3  
matany@campus.technion.ac.il

Ittay Eyal  
Technion, IC3  
ity@technion.ac.il

Abstract—Smart Contracts and transactions allow users to implement elaborate constructions on cryptocurrency blockchains like Bitcoin, Ethereum, and Libra. Many of these, including operational payment channels, use a building block called Hashed Time-Locked Contract (HTLC).

In this work, we distill from HTLC a specification (HTLC-Spec), and present an implementation called Mutual-Assured-Destruction Hashed Time-Locked Contract (MAD-HTLC). MAD-HTLC employs a novel approach of utilizing the existing blockchain operators, called miners, as part of the design. If a user misbehaves, MAD-HTLC incentivizes the miners to confiscate all her funds. We prove that MAD-HTLC satisfies HTLC-Spec with game-theoretic analysis and instantiate it on Bitcoin's operational blockchain.

Notably, current miner software makes only little effort to optimize revenue, since the advantage is relatively small. However, as the demand grows and other revenue components shrink, miners are more motivated to fully optimize their fund intake. By patching the standard Bitcoin client, we demonstrate such an optimization is easy to implement, making the miners natural enforcers of MAD-HTLC.

Finally, we show how vulnerable HTLC is to bribery attacks. An attacker can incentivize miners to prefer her transactions by offering high transaction fees. We demonstrate this can be easily implemented by patching the Bitcoin client, and use game-theoretic tools to qualitatively tighten the known cost bound of such bribery attacks.

I. INTRODUCTION

Blockchain-based cryptocurrencies like Bitcoin [1] and Ethereum [2] are monetary systems with a market cap of $270B [8]. They enable simple transactions of internal tokens and implementation of more elaborate smart contracts. The transactions create the smart contracts and interact with them. Entities called miners create data structures called blocks that contain transactions. They publish and order the blocks to form a blockchain, thus confirming the included transactions and achieving system progress. The system state is obtained by parsing the transactions according to the block order. Blockchain security relies on incentives, rewarding miners with tokens for carrying out their tasks.

A prominent smart-contract design pattern is the Hashed Time-Locked Contract (HTLC), set up for two participants, Alice and Bob. It asserts that Alice gets tokens for presenting a hash preimage of a specific value before a certain timeout, otherwise Bob gets them. A variety of more elaborate smart-contract designs rely on HTLC as a building block [11]. These include high-frequency payment channels [4], [5], [6], [7], [8], [9], [10], atomic swaps [11], [12], [13], [14], [15], contingent payments [16], [17], [18], [19], [20], and cryptocurrency vaults [21], [22], [23], [24]. We identify the specification required by the variety of contracts using HTLC and call it HTLC-Spec.

Unfortunately, HTLC is vulnerable to bribery attacks [25], [26], [27]. Winzer et al. [28] showed that Bob can bribe miners to ignore Alice’s transactions until the timeout elapses. This allows Bob to obtain the HTLC tokens while depriving Alice of them, even if Alice published the preimage.

In this work we provide a secure implementation for HTLC-Spec, and further analyze HTLC susceptibility.

We begin by describing the model for an underlying blockchain mechanism like that of Libra [29], Ethereum or Bitcoin [11]. The system’s state is a set of contracts; each contract comprises a token amount and a predicate; transactions redeem contract tokens by providing inputs that satisfy their predicates. Users publish transactions initiating new contracts, assigning them with the redeemed tokens while also offering some as fees. In each round one miner adds a block with a transaction to the chain and receives its fee.

We proceed to present MAD-HTLC, our HTLC-Spec implementation ([14]). MAD-HTLC relies on the novel idea that miners are also participants in a smart contract execution, and thus their interests should be taken into account. MAD-HTLC utilizes miners as enforcers of its correct execution, allowing and incentivizing them to seize its contract tokens in case of any bribery attempt. That, in turn, incentivizes Alice and Bob to refrain from such attempts and to interact with MAD-HTLC as intended.

In addition to the preimage specified by HTLC-Spec, which we denote preimga, MAD-HTLC uses a second preimage, preimgb, known only to Bob. MAD-HTLC comprises a main deposit contract (MH-Dep) and an auxiliary collateral contract (MH-Col), which work as follows. MH-Dep has three so-called redeem paths. First, it allows Alice to redeem it with a transaction including the predefined preimage, preimga. Alternatively, it allows Bob to redeem it after a timeout with a transaction including another preimage, preimgb. This is essentially the specification, but MH-Col provides another option, allowing any miner to redeem it herself with a transaction including both preimga and preimgb.

Now, if both Alice and Bob try to redeem MH-Dep, then their transactions must reveal preimages preimga and preimgb, respectively. Any miner can then simply take these preimages and issue her own transaction that uses the third redeem path to seize the tokens for herself. Specifically, if Alice tries to redeem MH-Dep, then Bob is assured that he cannot do so –
if he tried to redeem the tokens then the miners would get them instead. Assuming Bob is benign, i.e., rational but prefers to act honestly for the same reward, then this construction is sufficient to satisfy HTLC-Spec. But we can do better.

If Bob is spiteful, then he will prefer to reduce Alice’s reward if it doesn’t affect his. When Alice knows preimg\textsubscript{a}, and tries to redeem MH-Dep, Bob cannot redeem it as well, but he can publish a redeem transaction nonetheless allowing the miners to collect the tokens instead of Alice.

We strengthen MAD-HTLC such that Bob is strictly incentivized to refrain from such deviant behavior with the auxiliary contract MH-Col. It can be redeemed only after the same timeout as MH-Dep, either by a transaction of Bob, or by any miner that provides both preimg\textsubscript{a} and preimg\textsubscript{b}. Now, if Alice knows preimg\textsubscript{a}, then she can redeem MH-Dep and Bob can redeem MH-Col. If instead Bob contends with Alice for MH-Dep, both still lose the MH-Dep for the miners; but now both preimg\textsubscript{a} and preimg\textsubscript{b} are revealed, allowing miners to seize the MH-Col tokens as well. Bob is therefore strictly incentivized not to contend, allowing Alice to receive the tokens as required.

MAD-HTLC utilizes the mutual assured destruction [30] principle: If a party misbehaves then all parties lose everything. Although penalizing the well-behaved party as well, this mechanism design [31] technique ensures rational players act as intended.

We prove that MAD-HTLC satisfies HTLC-Spec by showing its desired behavior is the best response of the various players ([1], [V]). We formalize its execution as a game played by Alice, Bob and the miners, all seeking to maximize their revenue and minimize their expenses. We analyze the game and show that the prescribed behavior is incentive-compatible.

MAD-HTLC can be trivially implemented in rich smart-contract languages like Libra’s Move [32] and Ethereum’s Solidity [33]. We prove the efficacy of MAD-HTLC by implementing it in the less expressive Bitcoin Script [34], smart-contract language ([IV]). We deploy it on Bitcoin’s main network, and show it bears negligible overhead compared to the prevalent HTLC.

MAD-HTLC relies on miners’ rationality. However, as of today, default cryptocurrency clients only offer basic optimization. Changes in miners’ revenue structure will make better optimizations more important ([II-B]). To demonstrate miners can easily enhance transaction choice optimization once they choose to do so, we patch the standard Bitcoin client [35] to allow for easy additions of elaborate logic. In particular, we implement the logic enabling miners to benefit from enforcing the correct execution of MAD-HTLC.

We conclude by revisiting the security of the prevalent HTLC implementation and refine previous results [28] regarding its vulnerability to bribing attacks ([VII]).

We show that HTLC is vulnerable even in blockchains with limited Script-like languages. We analyze the miners’ behavior as a game. Each suffix of the game can be analyzed as a subgame, and all players have perfect knowledge of the system state. Bob can take advantage of this setting to incentivize miners to withhold Alice’s transaction until the timeout. He makes withholding the single subgame perfect equilibrium in the game. This proves that in the presence of rational miners the attack-cost lower bound of Winzer et al. [28] is tight. Here miners only have to be non-myopic for the attack to succeed, a simple optimization we implement by patching the standard Bitcoin client with merely 150 lines of code.

In summary, we make the following contributions:

- We formalize the specification HTLC-Spec of the prevalent HTLC contract,
- present MAD-HTLC that satisfies HTLC-Spec utilizing rational miners as participants,
- implement MAD-HTLC, deploy it, and evaluate its overhead,
- prove MAD-HTLC is incentive compatible,
- prove HTLC is vulnerable to bribery attacks in limited smart-contract environments, and
- qualitatively tighten the bound of Winzer et al. [28] and implement the required rational miner behavior.

Open Source and Responsible Disclosure: We intend to complete a responsible-disclosure process with prominent blockchain development groups before making this public. We intend to open source our code, subject to security concerns of the community.

II. RELATED WORK

We are not aware of previous work that utilized miners’ natural incentives to use them as participants in smart contract design.

We review previous work on bribing attacks in blockchains ([II-A]), then detail exhibited and postulated mining behavior with respect to transaction selection ([III-B]), and present systems and applications reliant on HTLC-Spec ([II-C]).

A. Bribery Attacks.

Winzer et al. [28] present attacks that delay confirmation of specific transactions until a given timeout elapses. Specifically, one of their attacks applies to HTLC where Bob delays the confirmation of Alice’s redeeming transaction until he can redeem it himself. Their presented attack requires a predicates available only with a richer smart contract languages like Ethereum’s Solidity [33] and Libra’s Move [32] but not Bitcoin’s Script [34]. In contrast, our attack works well, as we demonstrate by implementation. It therefore applies a wider range of systems [35], [37], [38].

They present two results regarding the attack costs. First, they show that Bob’s attack cost for making miner’s collaboration with the attack a Nash-equilibrium grows linearly with the size of the smallest miner. However, all miners not cooperating with the attack is also a Nash equilibrium. Therefore, they analyze Bob’s cost for making the attack a dominant strategy, i.e., each miner is incentivized to cooperate irrespective of the other miners’ strategies. This bound grows linearly with relative miner sizes, and exponentially with the HTLC timeout.
Our analysis improves this latter bound by taking into account the miners all know the system state and each others’ incentives. This insight allows us to use the subgame perfect equilibrium solution concept, a refinement of Nash-equilibrium suitable for games of dynamic nature. We consider the game played by rational participants aware of the game dynamics, and show that a linear-in-miner-size cost (as in [28]) suffices for the existence of a unique subgame perfect equilibrium.

Other work (E.g., [25], [26], [27]) analyzes bribing attacks on the consensus mechanism of cryptocurrency blockchains. Unlike this work, bribes in these papers compete with the total block reward (not just a single transaction’s fee) and lead miners to violate predefined behavior. These attacks are therefore much more costly and more risky than the bribery we consider, where a miner merely prioritizes transactions for confirmation.

B. Transaction-Choice Optimization

MAD-HTLC incentivizes rational entities to act in a desired way. It relies on the premise that all involved parties, namely Alice, Bob and the miners are rational. Specifically, that they monitor the blockchain state and issue transactions accordingly.

Indeed, previous work [28], [47], [48], [49], [50], [51] shows that this premise is prominent, and that system users and miners engage in carefully-planned transaction placing, manipulating their publication times and offered fees to achieve their goals. Other work [52], [53], [54], [55], [56], [57], [58] asserts that the profitability of such actions is expected to rise as the underlying systems mature, enabling constructions such as MAD-HTLC which rely on the rationality assumption.

C. HTLC-Spec usage

A variety of smart contracts critically rely on HTLC-Spec. To the best of our knowledge, all utilize HTLC, making them vulnerable once miners behave rationally. We review some prominent examples to illustrate the usefulness of HTLC-Spec.

a) Off-chain state channels: A widely-studied smart contract construction [5], [8], [10], [9], [6], [59], [60], [12], [7], [61] with implementations on various blockchains [4], [62], [63], [64], [65], [66] is that of an off-chain channel between two parties, Alice and Bob.

The channel has a state that changes as Alice and Bob interact, e.g., pay one another by direct communication. In the simplest case, the state is represented by a settlement transaction that Bob can place on the blockchain. The settlement transaction terminates the channel by placing its final state back in the blockchain. The transaction initiates a HTLC with a hash image of Bob’s choice. Bob can redeem the contract after the timeout, or alternatively, Alice can redeem before the timeout if Bob had shared the preimage with her.

When Alice and Bob interact and update the channel state, Bob revokes the previous settlement transaction by sending his preimage to Alice. This guarantees that if Bob places a revoked settlement transaction on the blockchain, Alice can redeem the tokens within the timeout. Alternatively, if Alice becomes unresponsive, Bob can place the transaction on the blockchain and redeem the tokens after the timeout elapses.

Note that this scheme assumes synchronous access to the blockchain – Alice should monitor the blockchain, identify revoked-state transactions, and issue her own transaction before the revocation timeout elapses. To remove this burden, services called Watchtowers [67], [68], [69] offer to replace Alice in monitoring the blockchain and issuing transactions when needed. However, these also require the same synchronous access to the blockchain, and the placement of transactions is still at the hands of bribable miners. MAD-HTLC can be viewed as turning the miners themselves into watchtowers – watchtowers that directly confirm the transactions, without a bribable middleman.

b) Atomic swaps: These contracts enable token exchange over multiple blockchain systems [11], [12], [13], [14], [15], [70], where a set of parties transact their assets in an atomic manner, i.e., either all transactions occur, or none.

Consider two users, Alice and Bob, that want to have an atomic swap over two blockchains. Alice picks a preimage and creates a HTLC on the first blockchain with timeout $T_1$. Then, Bob creates a HTLC requiring the same preimage (Bob knows only its hash) and a timeout $T_2 < T_1$ on the second blockchain. Alice publishes a transaction on the second blockchain, revealing the preimage and claiming the tokens. Bob learns the preimage from Alice’s published transaction, and publishes a transaction of his own on the first blockchain. If Alice does not publish her transaction before $T_2$ elapses, then the swap is canceled.

c) Zero-knowledge contingent payments: These [16], [17], [18], [19], [20] are transactions allowing Bob to purchase a secret from Alice such that he learns the secret if and only if the payment is performed.

The construction is as follows. Bob creates a program that decides if its given input is the requested secret, and sends it to Alice.

Alice produces a zero-knowledge proof showing that $e$ is an encryption of an input positively evaluating the program and $h$ is the hash of the decryption key, and sends the proof along with $e$ and $h$ to Bob.

Bob then verifies the proof and creates a HTLC allowing Alice to redeem it with the preimage of $h$ until a timeout. If she does so, she receives the payment and Bob learns the preimage and can decrypt $e$ to get the requested data.

d) Cryptocurrency vaults: These designs [21], [22], [23], [24] prevent theft of cryptocurrency tokens in case of a compromised private key [21], [22], [23].

The center of the vault construction is a HTLC, initiated by a single user with a secret key and a secret preimage that are heavily protected, and another secret key that is easily accessible to the user. To access the vault tokens the user can either use the first key and the secret preimage (like Alice) for immediate access, or the second key, providing a delayed access (like Bob). If the second secret key is compromised by an attacker that tries to access the vault tokens, then the user
has time to access and protect the vault tokens prior to the theft.

III. MODEL

We start by describing the system participants and how they form a chain of blocks that contain transactions (III-A). Next, we explain how the transactions are parsed to define the system state (III-B). Finally, we detail the required contract specification HTLC-Spec (III-C).

A. Blockchain, Transactions and Miners

We assume an existing blockchain-based cryptocurrency system, facilitating transactions of internal system tokens among a set of entities. All entities have access to a PKI and a random-oracle as a hash function $H$.

The blockchain serves as an append-only ledger storing the system state. It is implemented as a linked list of elements called blocks. A subset of the entities are called miners, who aside from transacting tokens also extend the blockchain by creating new blocks. We refer to non-mining entities as users.

There is a constant set of $n$ miners. Each miner is associated with a number representing its relative block-creation rate, or mining power. Denote the mining power of miner $i$ by $\lambda_i$, where $\sum_1^n \lambda_i = 1$. Denote the minimal mining power by $\lambda_{\text{min}} = \min \lambda_i$. As in previous work [28], [74], [54], [75], we consider these rates to be common knowledge, since in practice miners can monitor the blockchain and infer them [76].

Block creation is a discrete-time, memoryless stochastic process. At each time step exactly one miner creates a block. As in previous work [10], [9], [54], we disregard miners deliberately [74], [77], [78] or unintentionally [79], [80], [81] causing transient inconsistencies (called forks in the blockchain literature). Blocks are indexed by their location in the blockchain, we denote the first block by $B_1$ and the $j$th block by $B_j$.

Transactions update the system state. An entity creates a transaction locally, and can publish it to the other entities. Transaction publication is instantaneous, and for simplicity we abstract this process by considering published transactions to be part of a publicly-shared data structure called the mempool. As in previous work [9], [10], [54], all entities have synchronous access to the mempool and the blockchain.

Unpublished and mempool transactions are unconfirmed, and are yet to take effect. Miners can include unconfirmed transactions of their choice when creating a block, thus confirming them and executing the stated token reassignment.

The system limits the number of included transactions per block, and for simplicity (and without loss of generality) we consider this limit to be one transaction per block.

The system progresses in steps. Each step $j$ begins with system entities publishing transactions to the mempool. Then, a single miner is selected at random proportionally to its mining power, that is, miner $i$ is selected with probability $\lambda_i$. The selected miner creates block $B_j$, either empty or containing a single transaction, and adds it to the blockchain. This confirms the included transaction, reassigning its tokens and awarding that miner with its fee. The system then progresses to the next step.

B. System State

The system state is a set of token and predicate pairs called contracts. Transactions reassign tokens from one contract to another. We say that a transaction redeems a contract if it reassigns its tokens to one or more new initiated contracts.

To redeem a contract, a transaction must supply values such that the contract predicate evaluated over them is true. Transactions that result in negative predicate value are invalid, and cannot be included in a block. We simply disregard such transactions.

We say that an entity owns tokens if she is the only entity able to redeem their contract, i.e., the only entity that can provide input data in a transaction that results in positive evaluation of the contract’s predicate.

Transactions reassign tokens as follows. Each transaction lists one or more input contracts that it redeems, each with its respective provided values. Each transaction also lists one or more output contracts that it initiates. A transaction is only valid if the aggregate amount in the output contracts is not larger than the amount in its redeemed input contracts. The difference between the two amounts is the transaction’s fee. The fee is thus set by the entity that creates the transaction.

The system state is derived by parsing the transactions in the blockchain by their order. Each transaction reassigns tokens thus updating the contract set. Transaction fees are reassigned to a contract supplied by the miner that included them in a block.

Two transactions conflict if they redeem the same contract. Both of them might be valid, but only one can be placed in the blockchain. Once one of them is confirmed, a block containing the other is invalid. We disregard such invalid blocks, and assume miners only produce valid blocks.

There is always at least one unconfirmed valid transaction in the mempool, and the highest offered fee by any mempool transaction is $f$, referred to as the base fee. Miners act rationally to maximize their received fees ([II-B]). Users are also rational, and prefer to offer the minimal sufficient fee for having their transactions confirmed.

Predicates have access to three primitives:

- **VerSig** $(sig; pk)$: validate that a digital signature $sig$ provided by the redeeming transaction (on the redeeming transaction, excluding the signature itself) matches a public key $pk$ specified in the contract.
- **VerPreImg** $(x; y)$: validate that a hash preimage $x$ provided by the redeeming transaction matches an image $y$ specified in the contract, i.e., that $H(x) = y$.
- **VerTimeout** $(T)$: validate that the transaction redeeming the contract is in a block at least $T$ blocks after the transaction initiating it.

A predicate can include arbitrary logic composing those primitives. In predicates that offer multiple redeem options via or conditions, we refer to each option as a redeem path.
We note that once a transaction is published its content becomes available to all entities. We say that an entity knows data if it is available to it.

C. HTLC-Spec Specification

We formalize as HTLC-Spec the following contract specification, used in variety of blockchain-based systems and algorithms [4], [7], [8], [9], [10], [17], [18], [19], [11], [12], [13]. HTLC-Spec is specified for two users, Alice and Bob. It is parametrized by a hash image and a timeout, and contains a certain deposit amount, $v^\text{dep}$. Alice gets the deposit if she publishes a matching preimage before the timeout elapses, otherwise Bob does.

In a blockchain setting, Alice and Bob redeem the deposit with a transaction that offers a fee. We assume the contract token amount $v^\text{dep}$ is larger than the base fee $f$, otherwise the contract is not applicable.

The redeeming transaction by Alice or Bob (according to the scenario) should require a fee negligibly larger than the base fee $f$. Specifically, the fee amount is independent of $v^\text{dep}$.

To construct HTLC-Spec, Alice and Bob choose the included hash image, the timeout, and the token amount, $v^\text{dep}$. Then either of them issues a transaction that generates the contract with $v^\text{dep}$ tokens and the parametrized predicate. Either Alice or Bob initially knows the preimage, depending on the scenario.

For simplicity, we assume that Alice either knows the preimage when the transaction initiating HTLC-Spec is confirmed on the blockchain, or she never does.

IV. MAD-HTLC DESIGN

We present MAD-HTLC, an implementation of HTLC-Spec. MAD-HTLC comprises two contracts — MH-Dep, the core implementation of the HTLC-Spec functionality, and MH-Col, an auxiliary collateral contract used to disincentivize spurious behavior by Bob.

Alice and Bob execute HTLC-Spec by initiating MH-Dep with $v^\text{dep} > f$ and MH-Col with $v^\text{col} > f$ tokens, respectively. We denote the block including the initiating transaction by $B_j$.

The intended way Alice and Bob should interact with MAD-HTLC is as follows. If Alice knows the predefined preimage $\text{preimg}_a$, she publishes a transaction $T_a^\text{dep}$ offering a fee $f_a^\text{dep} > f$ that redeems MH-Dep. She must publish $T_a^\text{dep}$ before the creation of block $B_{j+T−1}$. If Alice does not know the predefined preimage $\text{preimg}_a$, she does not publish any transaction.

Bob observes the published transactions in the mempool, watching for $T_a^\text{dep}$. If by the creation of block $B_{j+T−1}$ Alice did not publish $T_a^\text{dep}$ then Bob publishes $T_b^\text{dep+col}$ with a fee $f_b^\text{dep+col} > f$, revealing $\text{preimg}_b$, redeeming both $T_a^\text{dep}$ and MH-Col. If Alice did publish $T_a^\text{dep}$ by block $B_{j+T−1}$ then Bob publishes $T_b^\text{col}$ with a fee $f_b^\text{col} > f$, redeeming only MH-Col.

We now present the specifics of MH-Dep (IV-A) and MH-Col (IV-B).

A. MH-Dep

The MH-Dep contract is initiated with $v^\text{dep}$ tokens. Its predicate is parametrized with Alice’s and Bob’s public keys, $pk_a$ and $pk_b$, respectively; A hash image of the predefined secret $\text{img}_a = H(\text{preimg}_a)$ such that any entity other than Alice and Bob does not know $\text{preimg}_a$, and Alice or Bob know $\text{preimg}_a$ according to on the specific use case; Another hash image $\text{img}_b$ such that $H(\text{preimg}_a) = \text{img}_b$, where only Bob knows $\text{preimg}_b$; and a timeout $T$. The contract has three redeem paths, presented in Predicate 1. Table I shows the redeeming entity of MH-Dep.

In the first path (line 1), Alice can redeem MH-Dep by creating a transaction including $\text{preimg}_a$ and $\text{sig}_a$, a signature created using her secret key $sk_a$. Such a transaction can be included even in the next block $B_{j+1}$. This path is only available to Alice, since only she ever knows $sk_a$.

In the second path (line 2), Bob can redeem MH-Dep by creating a transaction including $\text{preimg}_b$, and a signature created using his secret key $sk_b$. Such a transaction can be included in a block at least $T$ blocks after MH-Dep’s initiation, that is, not earlier than block $B_{j+T}$. This path is only available to Bob, since only he ever knows $sk_b$.

In the third path (line 3), any entity can redeem MH-Dep by creating a transaction including both $\text{preimg}_a$ and $\text{preimg}_b$. A transaction taking the third redeem path does not require a digital signature, and can be included even in the next block $B_{j+1}$. This path is therefore available to any entity that knows both $\text{preimg}_a$ and $\text{preimg}_b$.

B. MH-Col

The MH-Col contract is initiated with $v^\text{col}$ tokens. Its predicate is parametrized with Bob’s public key $pk_b$; the hash image of the predefined secret $\text{img}_b = H(\text{preimg}_b)$ such that any entity other than Alice and Bob does not know $\text{preimg}_b$, and Alice and Bob know $\text{preimg}_b$ based on the specific use case; the hash image $\text{img}_b$ such that $H(\text{preimg}_b) = \text{img}_b$, where only Bob knows $\text{preimg}_b$; and a timeout $T$. It has two redeem paths, presented in Predicate 2. Table II shows the intended redeeming entity of MH-Col.

Both paths are constrained by the timeout $T$, meaning a redeeming transaction can only be included in a block at least $T$ blocks after the MH-Col initiation (line 1).
In the first path (line 2), Bob can redeem MH-Col by creating a transaction including a signature created using his secret key $sk_b$. Only Bob can redeem MH-Col using this path as he is the only one able to produce such a signature. This path allows Bob to claim the collateral tokens in case either he or Alice, but not both, publish a transaction redeeming MH-Dep.

The second path (line 3) allows any entity to redeem MH-Col by creating a transaction including both $preimg_a$ and $preimg_b$, not requiring any digital signature. This path allows miners to claim the MH-Col tokens in case Bob tries contesting Alice on redeeming MH-Dep, incentivizing him to refrain from doing so.

V. MAD-HTLC Analysis

To reason about MAD-HTLC’s security we must take into account miner behavior. Miners are the ones that confirm transactions, and they can behave in unintended and undesired ways to increase their gains. We formalize MAD-HTLC as a game played by Alice, Bob and the miners (\textit{V-A}), and show that they are all incentivized to act as intended (\textit{V-B}). We conclude by discussing mitigations in a stronger threat model, where either Alice or Bob have mining power (\textit{V-C}).

A. MAD-HTLC Game

The MAD-HTLC game begins when the MH-Dep and MH-Col are initiated in some block $B_j$. The game, which we denote by $\Gamma_{MH}$, comprises $T$ rounds, representing the creation of blocks $B_{j+1}, ..., B_{j+T}$. Each round begins with Alice and Bob publishing transactions, followed by a miner creating a block including a transaction of her choice.

Alice and Bob’s strategies are their choices of published transactions — which transactions to publish, when, and with what fee. Miner strategies are the choices of transaction to include in a block if they are chosen to create one.

To accommodate for the stochastic nature of the game, we consider entity utilities as the expected number of tokens they own at game conclusion, i.e., after the creation of $T$ blocks. Alice and Bob’s utilities depend on the inclusion of their transactions and their offered fees, and miner utilities on their transaction inclusion choices.

We present the game details (\textit{V-A1}) and the suitable solution concept (\textit{V-A2}).

1) Game Details: The game progresses in rounds, where each round comprises two steps. First, Alice, then Bob alternately publish transactions, until both do not wish to publish any more.

Note that all published transactions of the current and previous rounds are in the mempool. Since miners prefer higher fees, for the analysis we ignore any transaction $tx$ if there is another transaction $tx'$ such that both were created by the same entity, both redeem the same contracts, and $tx'$ pays a higher fee than $tx$ or arrives before $tx$.

Tokens are discrete, hence there is a finite number of fees Alice and Bob may offer, meaning this step is finite.

Then, a single miner is picked at random proportionally to its mining power and gets to create a block including a transaction of her choice, receiving its transaction fees. Each miner can also create a new transaction and include it in her block.

a) Subgames: The dynamic and turn-altering nature of the game allows us to define subgames, representing suffixes of $\Gamma_{MH}$. For any $k \in [1, T]$ we refer to the game starting just before round $k$ as the $k$’th subgame (Fig. 1).

Note that as miners create blocks and confirm transactions, the system state, including the state of MAD-HTLC, changes. Specifically, if MH-Dep is already redeemed, future blocks do not allow inclusion of conflicting transactions that redeem MH-Dep as well.

Hence, when considering MAD-HTLC states we need to distinguish whether MH-Dep is redeemable or irredeemable, which we denote by red and irred, respectively. We also note that MH-Col cannot be redeemed until the very last $T$’th subgame.

Consequently, each subgame $k \in [1, T]$ is defined by the number of remaining blocks to be created $k$, and the MH-Dep state $s \in \{\text{red, irred}\}$. We denote such a subgame by $\Gamma_{MH} (k, s)$.

We use $\cdot$ to denote sets of subgames. For example, we denote by $\Gamma_{MH} (\cdot, \text{red})$ the set of subgames where the contract state $s$ is red.

We refer to $\Gamma_{MH} (T, \cdot)$ as the final subgames, as once played, the full game $\Gamma_{MH}$ is complete. We refer to all other subgames as non-final.

The game begins when there are $T$ blocks to be created, Alice and Bob did not publish any transactions, and the MH-Dep is redeemable. Thus, the initial, complete game is $\Gamma_{MH} (1, \text{red})$.

Once the first round of a non-final subgame is complete, the system transitions to the subsequent subgame.

b) Actions: Alice and Bob’s actions are the publication of transactions in any $\Gamma_{MH} (\cdot, \cdot)$ subgame.

Alice can only redeem MH-Dep, hence has a single transaction of interest $tx_{a}^{c}$ offering fee of $f_{a}^{c}$ tokens. Note $tx_{a}^{c}$ has to outbid unrelated transactions and thus has to offer a fee $f_{a}^{c} > f$, however, cannot offer more tokens than the redeemed ones, so $f_{a}^{c} < v_{a}^{c}$. This transaction utilizes the first redeem path of MH-Dep, hence publishing it also publishes $preimg_{a}$.

Bob can redeem MH-Dep, MH-Col or both. We thus consider three actions of interest: $tx_{b}^{c}$, redeeming $tx_{b}^{c}$ while...
offering fee \( f^\text{dep}_b \); \( tx^\text{col}_b \), redeeming \( MH\text{-Col} \) while offering fee \( f^\text{col}_a \); and \( tx^\text{depcol}_b \), redeeming both \( tx^\text{dep}_b \) and \( MH\text{-Col} \) while offering fee \( f^\text{depcol}_b \). To redeem \( MH\text{-Dep} \) Bob uses the second redeem path, hence publishing transactions \( tx^\text{dep}_b \) or \( tx^\text{depcol}_b \) also publishes \( \text{preimg}_b \). Similarly to Alice’s fee considerations, Bob’s transactions have to outbid unrelated transactions, and cannot offer more tokens than they redeem. We get that \( f < f^\text{dep}_b < v^\text{dep} \), \( f < f^\text{col}_b < v^\text{col} \) and \( f < f^\text{depcol}_b < v^\text{dep} + v^\text{col} \).

A miner’s action is the choice of a transaction to include if she is chosen to create a block. First, a miner can include a transaction unrelated to \( MAD\text{-HTLC} \) in any \( \Gamma^MH (\cdot, \cdot) \) subgame.

She can also include any of the following transactions, assuming they were previously published, and as a function of the contract state:

- \( tx^\text{dep}_a \) if \( MH\text{-Dep} \) is redeemable, that is, in any \( \Gamma^MH (\cdot, \text{red}) \);
- \( tx^\text{col}_b \) if the timeout has elapsed, that is, in any \( \Gamma^MH (T, \cdot) \); and
- \( tx^\text{dep}_b \) or \( tx^\text{depcol}_b \) if the timeout has elapsed and \( MH\text{-Dep} \) is redeemable, that is, in any \( \Gamma^MH (T, \text{red}) \).

Conditioned on published transactions, a miner can also create and include the following transactions:

- Transaction \( tx^\text{dep}_a \), redeeming \( MH\text{-Dep} \) herself, using the third redeem path, and getting the \( v^\text{dep} \) tokens of \( MH\text{-Dep} \) as reward. This action is only possible if the miner knows both \( \text{preimg}_a \) and \( \text{preimg}_b \), and if \( MH\text{-Dep} \) is redeemable, that is, in any \( \Gamma^MH (\cdot, \text{red}) \) subgame where \( tx^\text{dep}_a \) and either of \( tx^\text{dep}_b \) or \( tx^\text{depcol}_b \) were published.
- Transaction \( tx^\text{col}_a \), redeeming \( MH\text{-Col} \) herself, using the second redeem path, and getting the \( v^\text{col} \) tokens of \( MH\text{-Col} \) as reward. This action is only possible if the miner knows both \( \text{preimg}_a \) and \( \text{preimg}_b \), and the timeout has elapsed, that is, in any \( \Gamma^MH (T, \text{red}) \) subgame where \( tx^\text{dep}_a \) and either of \( tx^\text{dep}_b \) or \( tx^\text{depcol}_b \) were published.
- Transaction \( tx^\text{depcol}_a \), redeeming both \( MH\text{-Dep} \) and \( MH\text{-Col} \) herself, using the respective third and second redeem paths, and getting the \( v^\text{col} + v^\text{dep} \) tokens of \( MH\text{-Dep} \) and \( MH\text{-Col} \) as reward. This action is only possible if the miner knows both \( \text{preimg}_a \) and \( \text{preimg}_b \), the \( MH\text{-Dep} \) is redeemable, and the timeout has elapsed, that is, in subgame \( \Gamma^MH (T, \text{red}) \) where \( tx^\text{dep}_a \) and either of \( tx^\text{dep}_b \) or \( tx^\text{depcol}_b \) were published.

We disregard actions that are trivially dominated \cite{44}, such as Alice and Bob sharing their secret keys or publishing the relevant preimages not via a transaction, or miners creating an empty block.

c) Strategy: A strategy \( \sigma \) is a mapping from each subgame to a respective feasible action, stating that an entity takes that action in the subgame. We call the strategy vector of all entities in a game a strategy profile, denoted by \( \sigma \).

d) Utility: Recall an entity’s utility is her expected accumulated token amount at game conclusion. We define the utility of an entity in a subgame as the expected token amount she accumulates within the subgame until its conclusion. We denote the utility of entity \( i \) when all entities follow \( \sigma \) in subgame \( \Gamma^MH (k, s) \) by \( u_i (\sigma, \Gamma^MH (k, s)) \).

2) Solution Concept: Note that block-creation rates, entity utilities and their rationality are all common knowledge, and that when choosing an action an entity is aware of the current system state. That means any subgame \( \Gamma^MH (k, s) \) is of perfect information \cite{83, 84}. We are thus interested in strategy profiles that are subgame perfect equilibria \cite{39, 40, 41, 42, 43, 44, 45, 46}. A strategy profile \( \sigma \) is a subgame perfect equilibrium in \( \Gamma^MH (k, s) \) if, for any subgame, no entity can increase her utility by deviating to a different strategy, where it knows how the other players would react based on their perfect knowledge. This implies that for each subgame, the actions stated by \( \sigma \) are a Nash equilibrium.

We say that a prescribed strategy profile is incentive compatible if it is a subgame perfect equilibrium, and the utility of each player is not lower than her utility in any other subgame perfect equilibrium. So an entity cannot deviate to increase her utility, and there are no other more favorable equilibria.

Our analysis utilizes the common technique of backward induction \cite{85, 86, 87, 46}, suitable for perfect-information finite games. Intuitively, to determine its action, a player analyzes the game outcome for each possible action, repeating the process recursively for each possible game suffix.

B. MAD-HTLC Incentive Compatibility

We now show that MAD-HTLC satisfies HTLC-Spec. For that, we show the prescribed behavior of MAD-HTLC is incentive compatible and implements the HTLC-Spec.

We first analyze Alice’s and Bob’s utilities when both follow the prescribed strategy, starting with the scenario where Alice knows the preimage \( \text{preimg}_a \).

Lemma V.1. In \( \Gamma^MH (1, \text{red}) \), if Alice knows \( \text{preimg}_a \) and Alice and Bob both follow the prescribed strategies, then miners’ best-response strategy leads to Alice redeeming \( MH\text{-Dep} \) and receiving \( v^\text{dep} - f^\text{dep}_a \) tokens, and Bob redeeming \( MH\text{-Col} \) for \( v^\text{col} - f^\text{col}_b \) tokens.

Proof. The prescribed strategy states that Alice publishes \( tx^\text{dep}_a \) during the first \( T - 1 \) rounds, and that Bob publishes \( tx^\text{col}_b \) in round \( T \).

Note that Bob does not publish \( tx^\text{dep}_b \) and \( tx^\text{depcol}_b \), hence miners do not know \( \text{preimg}_b \). The transactions \( tx^\text{dep}_a \) and \( tx^\text{col}_b \) offer \( f^\text{dep}_a \) and \( f^\text{col}_b \) fees, respectively, both greater than the base fee \( f \).

The induced subgames therefore enable miners to include \( tx^\text{dep}_a \) in one of the first \( T - 1 \) blocks, and including \( tx^\text{col}_b \) in the last one. Using backward induction shows the subgame perfect equilibrium is to include \( tx^\text{dep}_a \) in its published round, and \( tx^\text{col}_b \) in the last.

So both \( tx^\text{dep}_a \) and \( tx^\text{col}_b \) are included in blocks, and Alice and Bob get \( v^\text{dep} - f^\text{dep}_a \) and \( v^\text{col} - f^\text{col}_b \) tokens, respectively.

Lemma V.2. In \( \Gamma^MH (1, \text{red}) \), if Alice does not know \( \text{preimg}_a \) and Alice and Bob both follow the prescribed strategies, then
miners’ best-response strategy leads to Bob redeeming both MH-Dep and MH-Col for $v^{\text{dep}} + v^{\text{col}} - f_b^{\text{dep+col}}$ tokens, and Alice gets none.

**Proof.** As Alice does not know $\text{preimg}_a$, she does not publish any transaction, hence redeems no contract and receives no tokens.

By the prescribed strategy Bob publishes $tx_b^{\text{dep+col}}$, offering fee $f_b^{\text{dep+col}} > f$ and revealing $\text{preimg}_b$. However, $\text{preimg}_a$ is not published, so miners cannot redeem MH-Dep and MH-Col themselves.

Therefore, miners maximize their utility by including $tx_b^{\text{dep+col}}$ in the last round.

That means $tx_b^{\text{dep+col}}$ is included in a block, and Alice and Bob get 0 and $v^{\text{dep}} + v^{\text{col}} - f_b^{\text{dep+col}}$ tokens, respectively. □

We now consider potential deviations from the prescribed strategy, showing they are strictly dominated. We begin by showing that if Alice and Bob contend then the miners do not take their transactions in the last round.

**Lemma V.3.** In the last round of the game, i.e., subgame $\Gamma^\text{MH}(T, \cdot)$, if $tx_a^{\text{dep}}$ and either $tx_b^{\text{dep}}$ or $tx_b^{\text{dep+col}}$ are published then miners’ best-response strategy is not to include any of Alice’s or Bob’s transactions in this round.

**Proof.** Since Alice and Bob published their transactions, both $\text{preimg}_a$ and $\text{preimg}_b$ are available to all miners. Therefore, any miner can create a transaction redeeming MH-Dep and MH-Col herself.

If MH-Dep is irredeemable ($\Gamma^\text{MH}(T, \text{irred})$), then miners can create $tx_m^{\text{col}}$ and redeem MH-Col themselves in round $T$, getting $v^{\text{col}}$ tokens as reward. Alternatively, if $tx_b^{\text{col}}$ is published they can include it in a block, getting a fee of $f_b^{\text{col}}$ tokens. As $f_b^{\text{col}} < v^{\text{col}}$, including $tx_b^{\text{col}}$ is strictly dominated by including $tx_m^{\text{col}}$. In this case miners can also not include $tx_a^{\text{dep}}$ as the MH-Dep is irredeemable.

If MH-Dep is redeemable ($\Gamma^\text{MH}(T, \text{red})$), miners can also create $tx_m^{\text{dep+col}}$, include it in a block, and get $v^{\text{dep}} + v^{\text{col}}$ tokens in reward. Alternatively, they can include either $tx_a^{\text{dep}}$, $tx_b^{\text{dep}}$, $tx_b^{\text{dep+col}}$ or $tx_b^{\text{dep+col}}$ (whichever was published). However, any of these offers fees lower than $v^{\text{dep}} + v^{\text{col}}$, making them strictly-dominated by including $tx_m^{\text{dep+col}}$.

In either scenario including any of Alice’s or Bob’s transactions results with a strictly lower reward, hence miners avoid doing so. □

Now, consider Alice’s potential deviations. The following lemma shows these do not increase her utility.

**Lemma V.4.** In $\Gamma^\text{MH}(1, \text{red})$, Alice cannot increase her utility by deviating from the prescribed strategy.

**Proof.** First, if Alice does not know $\text{preimg}_a$, she can take no action, hence trivially complies with the prescribed strategy.

If Alice does know $\text{preimg}_a$, then her possible deviations are not publishing $tx_a^{\text{dep}}$ at all, or publishing it only in the last round $T$. Not publishing $tx_a^{\text{dep}}$ at all is strictly dominated — she gets no tokens; if she instead abides by the prescribed strategy then she cannot get a lower revenue but can get more, e.g., if Bob also follows the prescribed strategy (Lemma V.1).

The inclusion of $tx_b^{\text{dep}}$ in the last block depends on what transactions Bob publishes throughout the game (Lemma V.3). That is, if Bob published either $tx_a^{\text{dep}}$ or $tx_b^{\text{dep+col}}$ then miners’ best-response is not to include $tx_a^{\text{dep}}$, and Alice gets no tokens. Otherwise, miners’ best response is to include the transaction that offers the highest fee, which can be either $tx_a^{\text{dep}}$ or another, resulting with Alice receiving $v^{\text{dep}} - f_a^{\text{dep}}$ and zero tokens, respectively.

So, Alice cannot gain, and in several scenarios strictly lose, by deviating from her prescribed strategy. □

We now consider Bob’s potential deviation.

**Lemma V.5.** In $\Gamma^\text{MH}(1, \text{red})$, Bob cannot increase his utility by deviating from the prescribed strategy.

**Proof.** Consider all of Bob’s possible actions. His potential maximal utility is from having $tx_a^{\text{dep+col}}$ included, which he obtains by following the prescribed strategy in the scenario where Alice does not know $\text{preimg}_a$ (Lemma V.2). So, he has no incentive to deviate in this case.

Now, consider the case where Alice knows $\text{preimg}_a$, hence according to Lemma V.4 Bob publishes $tx_a^{\text{dep}}$ in the first $T - 1$ rounds.

Bob can publish $tx_a^{\text{dep}}$, $tx_b^{\text{dep+col}}$ and $tx_b^{\text{col}}$ throughout the game. If he publishes $tx_b^{\text{dep}}$ or $tx_b^{\text{dep+col}}$ in any round then none of his transactions are included (Lemma V.3) and he gets no reward. However, if he only publishes $tx_b^{\text{col}}$ then by Lemma V.1 he receives $v^{\text{col}} - f_b^{\text{col}} > 0$ tokens.

Not publishing $tx_b^{\text{col}}$ at all results with the minimal utility of 0, and an earlier publication still leads miners to include both $tx_b^{\text{col}}$ and $tx_a^{\text{dep}}$ (cf. V.5), obtaining the same utility as of the prescribed behavior. □

Following directly from Lemma V.4 and Lemma V.5 we obtain:

**Corollary 1.** The prescribed strategy of MAD-HTLC is a unique subgame perfect equilibrium, and as such, incentive compatible.

We are now ready to prove our main theorem:

**Theorem V.6.** MAD-HTLC satisfies HTLC-Spec.

**Proof.** The prescribed strategy of MAD-HTLC is incentive compatible (Corollary I), hence Alice and Bob abide by it.

By Lemma V.2 and Lemma V.1 the MAD-HTLC prescribed strategy results with the outcomes specified in HTLC-Spec.

Note that matching the HTLC-Spec, the prescribed strategy states the redeeming transaction fee should be negligibly larger than $f$, and is independent of $v^{\text{dep}}$. □
Irrational Miners: MAD-HTLC’s design deters Bob from unjustly contending MH-Dep – Bob knows rational miners will seize his funds if he acts dishonestly.

However, even in the presence of unsophisticated (irrational) miners, MAD-HTLC still satisfies HTLC-Spec. The common mining logic \cite{88, 89, 90, 91} as of today has miners myopically optimize for the next block. Since Bob’s transaction can only be confirmed in the last round, these miners will simply include Alice’s transaction, achieving the desired outcome.

Only miners that are sophisticated enough to be non-myopic but not sophisticated enough to take advantage of the third path would cooperate with the attack. But even in the presence of such miners, it is sufficient for one miner (or user) to take advantage of the third path during the $T$ rounds in order to thwart the attack.

C. Mining Alice or Bob

The analysis of MAD-HTLC’s incentive compatibility assumes that Alice and Bob have no mining capabilities or, equivalently, that they do not collude with any miner. Removing this assumption extends the game space considerably, and brings in timing and probability considerations that are outside the scope of this work, as the following examples show.

Consider a scenario where Alice publishes a transaction redeeming MH-Dep, and Bob is colluding with a miner to obtain the MH-Dep tokens. Bob can privately share $preimg_b$, enabling the miner to redeem MH-Dep herself using the third redeem path.

Alternatively, assume Alice knows $preimg_a$ and so should redeem MH-Dep. Alice colludes with a miner to obtain the MH-Col tokens as well. She deliberately stalls publishing $tx_{dep}$ until round $T$. Bob, believing she is defunct, publishes a transaction to redeem MH-Col. Having observed $preimg_b$ included in Bob’s transaction, Alice shares her own $preimg_a$ to the miner. Now the miner can redeem both MH-Dep but also MH-Col.

A possible solution to these can be creating multiple MH-Dep and MH-Col instances with the same secrets, but each with a different timeout. In the case of a miner redeeming either of these by providing $preimg_a$ and $preimg_b$, they become public, hence letting all miners redeem all future contracts. This requires the colluding miner to create the blocks matching the different timeouts, which requires significant mining power to achieve with non-negligible (in the number of timeouts) probability.

VI. MAD-HTLC Implementation

To demonstrate the efficacy of MAD-HTLC, we evaluate it on Bitcoin. We discuss the deployment of MAD-HTLC and its overhead (\S VI-A), and our implementation of a framework for implementing miner rationality (\S VI-B) used to facilitate the contract guarantees.

| Name | Script size [bytes] | Redeem path | Redeeming transaction size [bytes] |
|------|---------------------|-------------|-----------------------------------|
| HTLC | 99                  | Alice, Bob | 291, 259                           |
| MH-Dep | 129            | Alice, Bob, Miner | 323, 322, 282 |
| MH-Col | 88               | Bob, Miner  | 248, 241                           |

Table III: Script and redeeming transaction sizes.

A. MAD-HTLC Bitcoin Implementation

We implement MH-Dep and MH-Col in Bitcoin’s Script \cite{34} language. We also implement a version of the standard HTLC for reference. We bring the details in App. \[A\]

Bitcoin’s transaction fees are determined by the transaction byte sizes due to the block size limit. Our contracts use $P2SH$ \cite{92} (non SegWit addresses, that is, the initiating transactions contain only the hashes of the scripts they initiate rather than the scripts themselves. Each contract initiation within a transaction therefore requires 28 bytes: 8 bytes for the token amount, and 20 bytes for the predicate script hash.

The redeeming transactions provide the full predicate script and the required input data. Table \[III\] presents the script and redeeming transaction sizes of HTLC, MH-Dep and MH-Col. A transaction redeeming MH-Dep is about 50 bytes larger than one redeeming HTLC. At the current Bitcoin common fees \cite{94} and exchange rate \cite{95} implies an additional cost of a mere $0.02.

Including the auxiliary MH-Col implies an additional cost of about $0.10. However, for lack of clear incentive for spiteful behavior on Bob’s account, system designers might prefer to avoid including it to reduce the overhead.

Further size-reduction optimizations such as using SegWit transactions and merging multiple transactions can also be made, but are outside the scope of this work.

We note that for off-chain channels \cite{4, 5, 6, 7, 8, 9, 10, 59, 60, 12, 61, 67} the HTLC or MAD-HTLC is actually used on the blockchain only in a unilateral channel closure, indicating an abnormal behavior by either or both parties. In the common case it never touches the blockchain, and incurs no overhead for the additional security.

We deployed multiple three MH-Dep instances on the Bitcoin main net, and successfully redeemed them using the three redeem paths. We also deployed two MH-Col instances, successfully redeemed using the two paths. Table \[IV\] shows the IDs of the initiating and redeeming transactions.

B. Rational Miner Infrastructure

By default, cryptocurrency clients \cite{88, 89, 90, 91} only perform basic transaction-inclusion optimizations. The capacity of each block is typically bounded, e.g., by storage volume

\footnote{https://www.blockchain.com/}.
Table IV: MAD-HTLC Main-net experiment transaction id. in Bitcoin [96] and by processing overhead in Ethereum [97], and current optimization practice is to fill each block with transactions paying the highest fees.

This is a myopic approach, limited only to considering transactions that can be included in the next block. Specifically, Bitcoin Core, which is used by about 97% of current Bitcoin nodes [35], maintains a local mempool data structure that contains unconfirmed transactions whose timeouts (if any) have elapsed.

This implementation does not allow users to control transaction choices. We note that this limitation is not a consensus rule, but an implementation choice in the current software. Taking more elaborate considerations into account when choosing transactions is not a violation of legal miner behavior and does not affect the overall system’s specified behavior.

As noted (§II-B), optimizing the revenue from transactions is becoming more important for miners over time. To demonstrate the ease of achieving broader optimizations, including non-myopic considerations, we implemented rational miner infrastructure (RMI), an infrastructure allowing to easily incorporate any logic over Bitcoin Core received transactions.

RMI’s main design goal is to enable users to deploy their own optimization algorithms. For that, it comprises a patched C++ Bitcoin Core node, with added RPC commands, and a Python script (Figure 2, new components shaded), working as follows.

When the node receives a new transaction, instead of directly placing it in its mempool, it pushes the transaction to a designated new transaction queue. The Python script monitors this queue with a dedicated RPC, fetches new transactions and parses them. Then, based on the implemented optimization algorithm, it can instruct the node how to handle the transaction – insert it to the mempool, discard it, or keep it for future use. The Python script can also generate new transactions, and send them to the node.

The changes to Bitcoin Core include only 140 lines of code, allowing miners to easily incorporate rationality in Python script using the prevalent client implementation.

We implemented and locally tested the Python script for enforcing MAD-HTLC by taking advantage of the opportunistic miner (RMI), an infrastructure allowing to easily incorporate any logic over Bitcoin Core received transactions. This implementation required 350 lines of Python code.

VII. HTLC

The prevalent implementation of HTLC-Spec is a direct translation of the specification to a single contract called HTLC (§VII-A).

It relies on the premise that miners benevolently enforce the desired execution, namely include Alice’s transaction in a block before the timeout elapses. However, this assumption contradicts the core principle of cryptocurrency permissionless systems — miners operate to make profit [48], [54], [56], [49], [57], [52], [53], and include transactions that benefit their personal gains [47], [50], [51]. Specifically, Bob can incentivize miners with a bribe [28], [26], [27] to exclude Alice’s transaction until the timeout elapses, and then redeem the HTLC himself.

We analyze the security of HTLC by formalizing the game played among the entities (§VII-B), and showing how cheap Bob’s required bribe is (§VII-C). We show miner fee optimization is easy by implementing a bribery-accepting (i.e., rational) miner (§VII-D), and conclude by estimating the actual attack cost using numbers from operational contracts (§VII-E).

A. Construction

Alice and Bob execute HTLC-Spec by initiating a single HTLC initiated with \( v^{dep} \) tokens. The HTLC’s predicate is parameterized with Alice’s and Bob’s public keys, \( pk_a \) and \( pk_b \), respectively; A hash image of the predefined secret \( \text{preimg}_a = H(\text{preimg}_a) \) such that any entity other than Alice and Bob does not know \( \text{preimg}_a \) (Alice or Bob know \( \text{preimg}_a \) based on the specific use case); and a timeout \( T \).

The HTLC has two redeem paths, presented in Predicate [3]. In the first path (line 1), Alice can redeem it with a transaction including \( \text{preimg}_a \) and \( \text{sig}_a \), a signature with her secret key \( sk_a \). In the second path (line 2), Bob can redeem it with a transaction including \( \text{sig}_a \), a signature with his secret key \( sk_b \). This transaction can only be included in a block at

| Description | Transaction ID |
|-------------|---------------|
| Initiate MH-Dep | d03217526d145055860296cbca8f7462 |
| MH-Dep path 1 | 33c957b69f75e79d2d40a38504ce49a3 |
| MH-Dep path 2 | cd090c90af4ac6e2b58834fd96f177 |
| MH-Dep path 3 | 505c7feff6b2f5c66bf7f2cc5f6e37a |
| Initiate MH-Col | e4830db5680b348c61c51222f8d0b |
| MH-Col path 1 | 8169ab596536fd40db4f98e776c1b4 |
| MH-Col path 2 | 406b6b6b576d242c7b849f6a633 |

Figure 2: RMI design, new components shaded.
least $T$ blocks after HTLC’s initiation, that is, not earlier than block $B_{j+T}$.

Note that as only Alice and Bob know their respective secret keys, other entities cannot redeem the HTLC.

The intended way Alice and Bob should interact with HTLC is as follows. If Alice knows the predefined preimage $\text{preimg}_a$, she publishes a transaction $tx^h_a$ offering a fee $f^a_h > f$ that redeems the HTLC. She publishes this transaction right after the creation of block $B_j$, that is, before the creation of block $B_{j+1}$. If Alice does not know the predefined preimage $\text{preimg}_a$, she does not publish any transactions.

Bob observes the published transactions in the mempool, watching for $tx^h_a$. If by block $B_{j+T-1}$ Alice did not publish $tx^h_a$ then Bob publishes $tx^h_b$ with a fee $f^b_h > f$, redeeming the HTLC. If Alice did publish $tx^h_a$ by block $B_{j+T-1}$ then Bob does not publish any transactions.

B. HTLC Game

HTLC operation gives rise to a game, denoted by $\Gamma^H$, played among Alice, Bob, and the miners. It is a very similar to that of the MAD-HTLC game (§V-A), and we therefore present only the differences.

a) Subgames: The game state is simply the state of the HTLC, which can be either redeemable or irredeemable, denoted by red and irred, respectively.

The game begins when there are $T$ blocks to be created, Alice and Bob did not publish any transactions, and the HTLC is redeemable. Thus, the initial, complete game is $\Gamma^H (1, \text{red})$.

b) Actions: Alice can redeem the HTLC with a transaction $tx^h_a$, offering $f^a_h$ tokens as fee. Note $tx^h_a$ has to outbid unrelated transactions and thus has to offer a fee $f^a_h > f$, however, cannot offer more tokens than the redeemed ones, so $f^a_h < v_{\text{dep}}$. Alice redeems HTLC using the first redeem path, so $tx^h_a$ can be confirmed in any round.

Bob can redeem HTLC with a transaction $tx^h_b$, offering $f^b_h$ tokens as fee. Similarly, $f^b_h$ is bounded such that $f < f^b_h < v_{\text{dep}}$. Bob redeems HTLC using the second redeem path, so $tx^h_b$ can only be confirmed in the last round.

A miner can take the following actions:

- Include a transaction unrelated to HTLC in any $\Gamma^H (\cdot, \cdot)$ subgame.
- Include $tx^h_a$ if the HTLC is redeemable, that is, in any $\Gamma^H (\cdot, \text{red})$ subgame.
- Include $tx^h_b$ if the timeout has elapsed and the HTLC is redeemable, that is, in any $\Gamma^H (T, \text{red})$ subgame.

C. Bribe Attack Analysis

We now show the HTLC prescribed strategy (§VII-A) is not incentive compatible. Specifically, we show that if Alice commits to the prescribed strategy, then Bob strictly gains by publishing a conflicting transaction as well, outbidding Alice’s fee, thus incentivizing miners to exclude Alice’s transaction and include his instead.

So, let Alice publish $tx^h_a$ with fee $f^a_h$ in the first round, and Bob publish a transaction $tx^h_b$ with fee $f^b_h > f^a_h = f - \frac{f - f_{\text{lim}}}{\lambda_{\text{max}}} + f$.

Focusing on miner actions, we show through a series of lemmas they are incentivized to include $tx^h_a$ and to exclude $tx^h_b$, resulting with lower utility for Alice, higher utility for Bob, and a violation of the HTLC-Spec.

Denote by $\bar{\sigma}$ the best response strategy of miners in this setting.

First, we show miner utilities for subgames where the HTLC is irredeemable.

**Lemma VII.1.** For any $k \in [1, T]$, the utility of miner $i$ in subgame $\Gamma^H (k, \text{irred})$ is $u_i (\bar{\sigma}, \Gamma^H (k, \text{irred})) = \lambda_i (T - k + 1) f$.

**Proof.** Since HTLC is irredeemable, the only available action for miners is to include an unrelated transaction, yielding a reward of $f$.

Consider any $\Gamma^H (k, \text{irred})$ subgame. There are $T - k + 1$ remaining blocks to be created, and miner $i$ creates any of them with probability $\lambda_i$. This scenario can be viewed as a series of $T - k + 1$ Bernoulli trials with success probability $\lambda_i$. The number of successes is therefore Binomially distributed, and the expected number of blocks miner $i$ creates is $\lambda_i (T - k + 1)$. The reward for each block is $f$, so miner $i$’s utility is $u_i (\bar{\sigma}, \Gamma^H (k, \text{irred})) = \lambda_i (T - k + 1) f$.

We now consider miner utilities for $\Gamma^H (\cdot, \text{red})$ subgames, where the HTLC is redeemable. We begin with the final subgame $\Gamma^H (T, \text{red})$, creating block $B_{j+T}$.

**Lemma VII.2.** Choosing to include $tx^h_a$ is a unique subgame perfect equilibrium in $\Gamma^H (T, \text{red})$, and miner $i$’s utility when doing so is $u_i (\bar{\sigma}, \Gamma^H (T, \text{red})) = \lambda_i f^a_h$.

**Proof.** In the $\Gamma^H (T, \text{red})$ subgame, the miner that creates the block has three transactions to pick from: She can include an unrelated transaction for the base fee $f$, $tx^h_a$ for $f^a_h$, or $tx^h_b$ for $f^b_h$.

If $f^b_h > f^a_h$, then $f^b_h > f^a_h > f$. If $f^a_h < f^b_h$, then $f^a_h > f$. That means including $tx^h_a$ yields strictly greater reward than all other actions, thus being a unique subgame perfect equilibrium in this subgame.

Miner $i$ creates the block with probability $\lambda_i$, and so her expected profit, i.e., utility, is $u_i (\bar{\sigma}, \Gamma^H (T, \text{red})) = \lambda_i f^a_h$.

We now move on to consider any earlier ($k \in [1, T - 1]$) subgame (Blocks $B_{j+1}$ to $B_{j+T-1}$) where the HTLC is redeemable.

**Lemma VII.3.** For any $k \in [1, T - 1]$, the unique subgame perfect equilibrium is that every miner includes an unrelated transaction in $\Gamma^H (k, \text{red})$, and miner $i$’s utility when doing so is $u_i (\bar{\sigma}, \Gamma^H (k, \text{red})) = \lambda_i ((T - k) f + f^b_h)$.
Proof. Note that in $\Gamma_H^i (k, \text{red})$ there are two actions available, either include an unrelated transaction and receive $f$ reward, or include $tx^a_i$ and receive $f^h_i$ reward.

Consider any miner $i$. Denote by $\lambda^i_k$ the accumulated block-creation rates of miners, excluding miner $i$, that choose to include an unrelated transaction in $\Gamma_H^i (k, \text{red})$. Therefore, the accumulated probabilities of miners that choose to include $tx^a_i$, excluding miner $i$, is $1 - \lambda^i_k - \lambda_i$.

If miner $i$ chooses to include an unrelated transaction then either of the following occurs. First, with probability $\lambda_i$, miner $i$ gets to create a block, includes an unrelated transaction and receives a reward of $f$. The subsequent subgame is $\Gamma_H^i (k + 1, \text{red})$. Alternatively, with probability $\lambda^i_k$ another miner that includes an unrelated transaction gets to create a block, miner $i$ gets no reward and the subsequent subgame is $\Gamma_H^i (k + 1, \text{red})$. Finally, with probability $1 - \lambda^i_k - \lambda_i$ another miner that includes $tx^a_i$ gets to create a block, miner $i$ gets no reward and the subsequent subgame is $\Gamma_H^i (k + 1, \text{irred})$.

Therefore, miner $i$’s utility when including an unrelated transaction in these subgames is

$$u_i (\bar{\sigma}, \Gamma_H^i (k, \text{red})) =$$

$$\lambda_i \cdot (f + u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{red}))) +$$

$$\lambda^i_k \cdot u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{red})) +$$

$$(1 - \lambda_i - \lambda^i_k) \cdot u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{irred})) \quad (1)$$

Similarly, if miner $i$ chooses to include $tx^a_i$ than either of the following occurs. First, with probability $\lambda_i$, miner $i$ gets to create a block, includes $tx^a_i$ and receives a reward of $f$. The subsequent subgame is $\Gamma_H^i (k + 1, \text{irred})$. Alternatively, with probability $\lambda^i_k$ another miner that includes an unrelated transaction gets to create a block, miner $i$ gets no reward and the subsequent subgame is $\Gamma_H^i (k + 1, \text{irred})$. Finally, with probability $1 - \lambda^i_k - \lambda_i$ another miner that includes $tx^a_i$ gets to create a block, miner $i$ gets no reward and the subsequent subgame is $\Gamma_H^i (k + 1, \text{irred})$.

Therefore, miner $i$’s utility when including $tx^a_i$ in these subgames is

$$u_i (\bar{\sigma}, \Gamma_H^i (k, \text{red})) =$$

$$\lambda_i \cdot (f^a + u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{red}))) +$$

$$\lambda^i_k \cdot u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{red})) +$$

$$(1 - \lambda_i - \lambda^i_k) \cdot u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{irred})) \quad (2)$$

To prove the lemma we need to show that for any $k \in [1, T - 1]$ the utility from including an unrelated transaction (Eq. 1) exceeds that of including $tx^a_i$ (Eq. 2). This reduces to showing that

$$f + u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{red})) >$$

$$f^a + u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{irred})) \quad (3)$$

which we do inductively.

a) Base: First, consider $k = T - 1$. Using Lemma VII.2 and Lemma VII.1 we get the condition presented in Eq. 3 is

$$f + \lambda_i f^a > f^a + \lambda_i f$$

or alternatively,

$$f^h > \frac{\lambda_i f}{\lambda_i - \lambda} + f \quad (4)$$

Since $\lambda_{\min} \leq \lambda_i$ and $f^h > \frac{\lambda_i f}{\lambda_{\min}} + f$, the condition (Eq. 3) holds, meaning that in any subgame perfect equilibrium miner $i$ is strictly better by including an unrelated transaction in subgame $\Gamma_H^i (T - 1, \text{red})$.

Therefore, all miners choose to include unrelated transactions in such subgames, meaning $\lambda^i_k = 1 - \lambda_i$ and $1 - \lambda_i - \lambda^i_k = 0$. Therefore, miner $i$’s utility (Eq. 1) is $u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{red})) = \lambda_i ((T - k) + f^h_i)$.

b) Assumption: Consider any $k \in [1, T - 2]$ and assume that the claim holds for $k + 1$. That is, the unique subgame perfect equilibrium in subsequent games $\Gamma_H^i (k + 1, \text{red})$ is for all miners to include an unrelated transaction, and the utility of miner $i$ when doing so is $u_i (\bar{\sigma}, \Gamma_H^i (k + 1, \text{red})) = \lambda_i ((T - k) + f^h_i)$.

c) Step: Using the inductive assumption and Lemma VII.1 the condition of Eq. 3 translates to

$$f + \lambda_i ((k + 1) f + f^h_i) > f^a_i + \lambda_i (k + 1) f$$

or alternatively,

$$f^h > \frac{\lambda_i f}{\lambda_i - \lambda} + f \quad (5)$$

Again, since $\lambda_{\min} \leq \lambda_i$ and $f^h > \frac{\lambda_i f}{\lambda_{\min}} + f$, the condition (Eq. 5) holds, meaning that in the subgame perfect equilibrium miner $i$’s strict best response is to include an unrelated transaction in subgame $\Gamma_H^i (k, \text{red})$.

Since all miners include unrelated transactions, we get $\lambda^i_k = 1 - \lambda_i$ and $1 - \lambda_i - \lambda^i_k = 0$. Therefore, miner $i$’s utility (Eq. 1) is $u_i (\bar{\sigma}, \Gamma_H^i (k, \text{red})) = \lambda_i ((T - k) + f^h_i)$.

We conclude with the main theorem regarding HTLC susceptibility to bribing attacks.

Theorem VII.4. Alice’s prescribed behavior of HTLC allows Bob to bribe miners at a cost of $\frac{\lambda_i f}{\lambda_{\min}} + f$.

The proof follows directly from Lemmas VII.2 and VII.3 which show that if Alice naively follows the prescribed strategy then subgame perfect equilibrium of the initial sub-game is for all miners to place unrelated transactions until round $T$ and then place Bob’s transaction.

Note that by Theorem VII.4 the bribing cost required to attack HTLC is independent in $T$, meaning that simply increasing the timeout does contribute to HTLC’s security.

Of course once Alice sees an attack is taking place she can respond by increasing her fee. We conclude by showing that by paying a high fee dependent on $v_{dep}$, Alice can prevent the attack. We note that such a high fee is in violation of the HTLC-Spec.

Corollary 2. Bob cannot bribe the miners in this manner if Alice’s $tx^a_i$ offers at least $f^a_i > \lambda_{\min} (v_{dep} - f) + f$.

Proof. In order to achieve the attack, Bob should make placing unrelated transactions until $T$ and placing his transaction at $T$ a subgame perfect equilibrium. As we have seen (Eq. 5), the
threshold to incentivize the smallest miner is $f_h > \frac{\lambda_h - f}{\lambda_{\text{min}}} + f$. Recall the fee $f_h$ of the bribing transaction $\alpha_h$ is upper bounded by the HTLC tokens $v^{\text{dep}}$. Therefore, to achieve the attack it must hold that $v^{\text{dep}} > \frac{\lambda_h - f}{\lambda_{\text{min}}}$. By choosing $f_h > \lambda_{\text{min}} \left(\frac{v^{\text{dep}} - f}{f} + 1\right)$, Alice can prevent Bob from paying a fee adhering to the bounds.

Myopic Miners: This bribery attack relies on all miners being rational, hence considering their utility at game conclusion instead of myopically optimizing for the next block. If a portion of the miners are myopic and any of them gets to create a block during the first $T - 1$ rounds, that miner would include Alice’s transaction and Bob’s bribery attempt would have failed.

In such scenarios the attack succeeds only with a certain probability — only if a myopic miner does not create a block in the first $T - 1$ rounds. The success probability therefore decreases exponentially in $T$. Hence, to incentivize miners to support the attack, Bob has to increase his offered bribe exponentially in $T$.

We note a certain amount might be sufficient to bribe a rational miner in some game suffixes but not in others.

The exact analysis relies on assumptions on the mining power distribution, which is outside the scope of this work. Notably, for the simpler case when all other miners are myopic, miner $i$ is incentivized to support the attack only when it is her dominant strategy, matching the upper bound of Winzer et al. [28].

D. Bribery-Accepting Rational-Miner Implementation

Aside from the RMI mechanism (\{VI-B\}), we also implemented a simpler Bitcoin Core patch supporting the mentioned bribe attack on HTLC. This patch required 150 lines of C++ code and no additional external modules.

When the patched client receives transactions with an unexpired timeout (waiting transactions) it stores them in a data structure instead of discarding them. When creating a new block, the client first checks if any of the timeouts have elapsed, and if so, moves the relevant transactions to the mempool.

When receiving conflicting transactions, instead of accepting the first and discarding the second, it accepts the transaction that offers a higher fee. In case of a conflict with a waiting transaction, it chooses based on the condition described in Theorem VII.4.

The simplicity of this patch demonstrates that miners can trivially achieve non-myopic transaction selection optimization.

E. Real-World Numbers

We conclude this section by presenting three examples of HTLC being used in running systems, and show the substantial costs to make them resistant against bribery attacks.

Table V presents for each example the HTLC tokens $v^{\text{dep}}$, the base fee $f$, and the ratio of required tokens for bribery resistance (Theorem VII.4) and the base fee $\lambda_{\text{min}} \left(\frac{v^{\text{dep}} - f}{f} + 1\right)$.

| Name                          | $v^{\text{dep}}$ | $f$     | $\lambda_{\text{min}} \left(\frac{v^{\text{dep}} - f}{f} + 1\right)$ |
|-------------------------------|------------------|---------|---------------------------------------------------------------------|
| Lightning channel (BTC) [99], [100] | 2.684            | 2.22e-6 | 1.21e4                                                              |
| Litecoin atomic swap (LTC) [101] | 1.337            | 3.14e-4 | 435.7                                                               |
| Contingent ZK root (BTC) [16], [102], [103] | 0.1              | 0.002   | 10.5                                                                |

Table V: HTLC bribery resistance cost examples.

To estimate the base fee we conservatively take the actual paid fee, which is an upper bound. We conservatively estimate $\lambda_{\text{min}} = 0.01$ [76]; miners with lower mining power are less likely due to economy-of-scale [98].

The first use-case is of a Bitcoin Lightning channel [99], [100], where the required fee to secure the contract against a bribery is $1.21e4$ times what was actually paid. The second use-case is of a Litecoin atomic swap [101], requiring 436 times higher fee to be secured against bribes. The third use-case is of a contingent ZK proof demo [16], [102], [103], requiring 10.5 times higher fee to be secured. We note the latter relatively-low ratio is due to the low $v^{\text{dep}}$.

VIII. CONCLUSION

We introduce a novel approach of utilizing miner’s rationality to secure smart contracts, and use it to design MAD-HTLC, a contract implementing HTLC-Spec. We show with game-theoretic analysis that MAD-HTLC is secure, unlike the prevalent HTLC, which is vulnerable to cheap bribery attacks when miners behave rationally — we qualitatively tighten the known bound. We demonstrate the efficacy of our approach by implementing and executing MAD-HTLC on Bitcoin. We also demonstrate the practicality of implementing a rational miner by patching the standard Bitcoin client.

Both the attack against HTLC and the secure alternative MAD-HTLC have direct impact on a variety of contracts using the HTLC-Spec design pattern. As miners’ incentives to act rationally increase, those systems will become vulnerable and can directly adopt MAD-HTLC as a plug-in alternative.

IX. ACKNOWLEDGMENTS

This research was supported by the Israel Science Foundation (grant No. 1641/18), an IC3 research grant, Engima MPC Inc. and the Technion Hiroshi Fujiwara cyber-security research center.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008. [Online]. Available: http://www.bitcoin.org/bitcoin.pdf

[2] V. Buterin, “A next generation smart contract & decentralized application platform,” 2013. [Online]. Available: https://www.ethereum.org/pdfs/EthereumWhitePaper.pdf

[3] cryptoslate.com. (2020) Cryptocurrency market capitalizations. [Online]. Available: https://cryptoslate.com/cryptocurrency/

[4] J. Poon and T. Dryja, “The bitcoin lightning network: Scalable off-chain instant payments,” 2016.
[56] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan, “On the instability of bitcoin without the block reward,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016.

[57] I. Tsabary, A. Spiegelman, and I. Eyal, “Heb: Hybrid expenditure blockchain,” arXiv, pp. arXiv–1911. 2019.

[58] D. Easley, M. O’Hara, and S. Basu, “From mining to markets: The evolution of bitcoin transaction fees,” Journal of Financial Economics, vol. 134, no. 1, pp. 91–109, 2019.

[59] R. Khalil and A. Gervais, “Revive: Rebalancing off-blockchain payment networks,” in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 439–453.

[60] L. Gudeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, “Sok: Off the chain transactions.” IACR Cryptology ePrint Archive, vol. 2019, p. 360, 2019.

[61] L. Aumayr, O. Erozy, A. Erwig, S. Faust, K. Hostáková, M. Maffe, P. Moreno-Sanchez, and S. Riahi, “Generalized bitcoin-compatible channels.”

[62] Blockstream. (2020) Blockstream lightning implementation. [Online]. Available: https://blockstream.com/lightning/

[63] L. Labs. (2020) Lightning labs lightning implementation. [Online]. Available: https://lightning.engineering/

[64] Raiden. (2020) Acinq lightning implementation. [Online]. Available: https://acinq.co/

[65] Raiden. (2020) Raiden network. [Online]. Available: https://raiden.network/

[66] O. Network. “Omg network blockchain design.” [Online]. Available: https://docs.omg.network/blockchain-design

[67] G. Avraktiou, O. S. T. Litos, and R. Wattenhofer, “Cerberus channels: Incentivizing watchtowers for bitcoin,” Financial Cryptography and Data Security (FC), 2020.

[68] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, “Pisa: Arbitration outsourcing for state channels,” in Proceedings of the 1st ACM Conference on Advances in Financial Technologies, 2019, pp. 16–30.

[69] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, “Outpost: A responsive lightweight watchtower,” in Proceedings of the 1st ACM Conference on Advances in Financial Technologies, 2019, pp. 31–40.

[70] E. Wagner, A. Volker, F. Fuhrmann, R. Mattzutt, and K. Wehrle, “Dispute resolution for smart contract-based two-party protocols,” in 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2019, pp. 422–430.

[71] S. Ullah, “Trade.io reports theft of 7.5 million in funds from cold storage.” [Online]. Available: https://betahedz.com/trade-io-reports-theft-of-7-5-million-in-funds-from-cold-storage/

[72] Andrew Norry, “The history of the mt gox hack: Bitcoins biggest heist.” [Online]. Available: https://blockonomi.com/mt-gox-hack/

[73] Bloomberg, “How to steal 500 million in cryptocurrency.” [Online]. Available: https://fortune.com/2018/01/31/coincheck-hack-how/

[74] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in Financial Cryptography and Data Security, 2014.

[75] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, “Optimal selfish mining strategies in Bitcoin,” in Financial Cryptography and Data Security, 2016.

[76] blockchain.info, “Bitcoin mining pools.” [Online]. Available: https://blockchain.info/poolso

[77] andes, “Bitcoin’s kryptonite: The 51% attack.” June 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=12335

[78] M. Mirkin, Y. Ji, I. Pang, A. Klages-Mundt, I. Eyal, and A. Jules, “Bodos: Blockchain denial of service,” arXiv preprint arXiv:1912.07497, 2019.

[79] J. A. Garay, A. Kiayias, and N. Leonados, “The Bitcoin backbone protocol: Analysis and applications,” in Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2015.

[80] R. Pass, L. Seeman, and A. Shelat, “Analysis of the blockchain protocol in asynchronous networks,” in Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2017, pp. 643–673.

[81] L. Kiffer, R. Rajaraman, and A. Shelat, “A better method to analyze blockchain consistency,” in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 729–744.

[82] J-F. Mertens and A. Neyman, “Stochastic games,” International Journal of Game Theory, vol. 10, no. 2, pp. 53–66, 1981.

[83] Y. Shoham and K. Leyton-Brown, Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press, 2008.

[84] M. J. Osborne and A. Rubinstein, A course in game theory. MIT press, 1994.

[85] E. Zermelo, “Uber eine anwendung der mengenlehre auf die theorie des schachspiels,” in Proceedings of the fifth international congress of mathematicians, vol. 2. II, Cambridge UP, Cambridge, 1913, pp. 501–504.

[86] R. J. Aumann, “Backward induction and common knowledge of rationality,” Games and Economic Behavior, vol. 8, no. 1, pp. 6–19, 1995.

[87] M. M. Kamiński, “Backward induction: Merits and flaws,” Studies in Logic: Grammar and Rhetoric, vol. 50, no. 1, pp. 9–24, 2017.

[88] B. Core, “Bitcoin core.” [Online]. Available: https://bitcoin.org/en/

[89] G. Ethereum, “Geth.” [Online]. Available: https://github.com/ethereum/go-ethereum

[90] P. Wallet, “Parity.” [Online]. Available: https://github.com/ethereumjs/parity

[91] E. S. Exchange, “Ethereum client transaction ordering.” [Online]. Available: https://ethereum.stackexchange.com/questions/6107/what-is-the-default-ordering-of-transaction-during-mining-in-ether-

[92] G. Andresen. (2013) Bip 16 : Pay to script hash. [Online]. Available: https://blockstream.com/lightning/

[93] P. W. Eric Lombrozo, Johnson Lau. (2017) Bip 141 : Segregated witness (consensus layer). [Online]. Available: https://bitcointalk.org/index.php?topic=12435

[94] blockchain.info, “Bitcoin transaction fees,” 2020. [Online]. Available: https://www.blockchain.com/charts/mempool-state-by-fee-level

[95] CoinDesk. (2020) Bitcoin price. [Online]. Available: https://www.coindesk.com/price/bitcoin

[96] B. Glossary, “Bitcoin glossary.” [Online]. Available: https://developer.bitcoin.org/glossary

[97] E. G. Station, “Whats the maximum ethereum block size?” [Online]. Available: https://ethgasstation.info/blog/ethereum-block-size/

[98] N. Arnosti and S. M. Weinberg, “Bitcoin: A natural oligopoly,” arXiv preprint arXiv:1811.08572, 2018.

[99] CoinDesk. (2020) Bitcoin lightning top capacity channel. [Online]. Available: https://lnd.io

[100] B. Core, “Bitcoin core.” [Online]. Available: https://bitcoin.org/en/

[101] insight.litecore. (2017) Litecoin atomic swap. [Online]. Available: https://lightcore.org/txt/6c27eafbadb861f3be3be1e7aac7b2c6524cc3ef47800fca60eab36747037

[102] J. Clark and A. Essex, Financial cryptography and data security. Springer, 2016.

[103] BitBlockstream. (2016) First contingent zkp transaction. [Online]. Available: https://blockstream.info/txt/2005541391d1e3c3f6ec499ebd6b06e01c706ebc897293a7f6a2625e39ef23f

APPENDIX A

BITCOIN IMPLEMENTATION

Fig.3 shows the Bitcoin Script implementation of MH-Dep, MH-Col and HTLC. It also presents the required input data for each redeem path.

Script is stack-based, and to evaluate input data and a contract the latter is concatenated to the former, and then executed: constants are pushed into the stack, instructions operate on the stack. For a successful evaluation the stack must hold exactly one element with value 1 after all operations are executed.

a) MH-Dep: The script expects either two or three data elements. It hashes the first two and checks if they match imga and imgb.

If the first matches imga, but the second does not match imgb, (redeem path 1), then the script verifies the existence of a third
| MH-Dep          | MH-Col          | HTLC          |
|-----------------|-----------------|---------------|
| OP_HASH160      | T               | OP_HASH160    |
| img_a           | OP_CHECKSEQUENCEVERIFY | img_a         |
| OP_EQUAL        | OP_DROP         | OP_EQUAL      |
| OP_SWAP         | OP_HASH160      | img_b         |
| img_b           | OP_EQUAL        | OP_IF         |
| OP_IF           | OP_IF           | pk_a          |
| T               | OP_EQUAL        | OP_ELSE       |
| OP_IF           | OP_HASH160      | OP_ELSE       |
| OP_EQUAL        | OP_DROP         | T             |
| OP_IF           | img_b           | OP_ENDIF      |
| OP_ELSE         | OP_IF           | OP_DROP       |
| OP_VERIFY       | OP_HASH160      | pk_b          |
| pk_a            | OP_EQUAL        | OP_ENDIF      |
| OP_CHECKSIG     | OP_ELSE         | OP_CHECKSIG   |
| OP_ENDIF        | OP_ELSE         | OP_CHECKSIG   |
|                | OP_CHECKSIG     |               |
|                | OP_ENDIF        |               |

Redeem path | Input data  | Redeem path | Input data  | Redeem path | Input data  |
|------------|-------------|-------------|-------------|-------------|-------------|
| 1          | sig_a, OP_0, preimg_a | 1           | sig_b, OP_0 | 1           | sig_a, preimg_a |
| 2          | sig_b, preimg_b, OP_0 | 2           | preimg_b, preimg_a |             |              |
| 3          | preimg_b, preimg_a    |             |             |             |              |

Figure 3: MH-Dep, MH-Col and HTLC Script implementation.

data element, and that it is a signature created with Alice’s secret key.

If the first does not match $img_a$, but the second matches $img_b$ (redeem path 2), then the script verifies the existence of a third data element, and that it is a signature created with Bob’s secret key. It also verifies the timeout has elapsed.

If both the first and the second data elements match $img_a$ and $img_b$ (redeem path 3), respectively, then the script expects no third data element and evaluates successfully.

b) MH-Col: The script expects exactly two data elements. It begins by verifying timeout has elapsed, and then hashes the first element and checks if it matches $img_a$.

If not (redeem path 1), the script then verifies the second data is a signature created with Bob’s secret key. Otherwise (redeem path 2), the script hashes the second data element and verifies it matches $img_b$.

c) MH-Col: The script expects exactly two data elements. It hashes the first element and checks if it matches $img_a$.

If it does (redeem path 1), the script then verifies the second data is a signature created with Alice’s secret key. Otherwise (redeem path 2), the script hashes verifies the timeout has elapsed, and that the second data element is a signature created with Bob’s secret key.