2021 IEEE Symposium on Security and Privacy (SP)

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

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

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

Alex Manuskin
ZenGo-X
alex@manuskin.org

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

2021 IEEE Symposium on Security and Privacy (SP) | 978-1-7281-8934-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SP40001.2021.00080

***Abstract*—*Smart Contracts* and *transactions* allow users to implement elaborate constructions on *cryptocurrency blockchains* like Bitcoin and Ethereum. Many of these constructions, including operational payment channels and atomic swaps, 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 *MAD-HTLC*'s security using the UC framework and game-theoretic analysis. We demonstrate *MAD-HTLC*'s efficacy and analyze its overhead by instantiating it on Bitcoin's and Ethereum's operational blockchains.

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 optimization is easy to implement, making the miners natural enforcers of *MAD-HTLC*.

Finally, we extend previous results regarding *HTLC* vulnerability to *bribery attacks*. An attacker can incentivize miners to prefer her transactions by offering high *transaction fees*. We demonstrate this attack 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 in presence of rational miners. We identify bribe opportunities occurring on the Bitcoin and Ethereum main networks where a few dollars bribe could yield tens of thousands of dollars in reward (e.g., $2 for over $25K).

I. Introduction

Blockchain-based cryptocurrencies like Bitcoin [1] and Ethereum [2] are monetary systems with a market cap of $400B [3]. 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, $\mathcal{A}$, and Bob, $\mathcal{B}$ (§II). It asserts that $\mathcal{A}$ gets tokens for presenting a hash preimage of a specific value before a certain timeout, otherwise $\mathcal{B}$ gets them. A variety of more elaborate smart-contract designs rely on *HTLC* as a building block. These include high-frequency payment channels [4]–[10], atomic swaps [11]–[15], contingent payments [16]–[20], and vaults [21]–[24]. We identify the specification required by the variety of contracts using *HTLC* and call it *HTLC-Spec*.

Unfortunately, *HTLC* is vulnerable to *incentive manipulation attacks* [25]–[27]. Winzer et al. [28] showed that $\mathcal{B}$ can bribe miners using designated smart contracts to ignore $\mathcal{A}$'s transactions until the timeout elapses. Similarly, Harris and Zohar [29] show that $\mathcal{B}$ can delay $\mathcal{A}$'s transaction confirmation by overloading the system with his own transactions. Both of these allow $\mathcal{B}$ to obtain the *HTLC* tokens while depriving $\mathcal{A}$ of them, even if $\mathcal{A}$ published the preimage.

In this work, we provide a secure implementation of *HTLC-Spec*, and further analyze *HTLC*'s susceptibility to bribery.

We begin by describing the model (§III) for an underlying blockchain mechanism like that of Bitcoin or Ethereum. 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 (§IV). *MAD-HTLC* relies on the fact that miners can participate 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 $\mathcal{A}$ and $\mathcal{B}$ to refrain from such attempts and to interact with *MAD-HTLC* as intended. To the best of our knowledge, this is the first work to utilize miner incentives in this way.

In addition to the preimage specified by *HTLC-Spec*, which we denote pre_a, *MAD-HTLC* uses a second preimage, pre_b, known only to $\mathcal{B}$. *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 $\mathcal{A}$ to redeem it with a transaction including the predefined preimage, pre_a. Alternatively, it allows $\mathcal{B}$ to redeem it after a timeout with a transaction including the other preimage, pre_b. This is essentially the specification, but *MH-Dep* provides another option, allowing any user, and specifically any miner, to redeem it herself with a transaction including both pre_a and pre_b.

DOI 10.1109/SP40001.2021.00080

Now, if both $\mathcal{A}$ and $\mathcal{B}$ try to redeem *MH-Dep*, then their transactions must reveal preimages pre_a and pre_b, 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 $\mathcal{A}$ tries to redeem *MH-Dep*, then $\mathcal{B}$ is assured that she cannot do so – if she tries to redeem the tokens then the miners would get them instead. Assuming $\mathcal{B}$ 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 $\mathcal{B}$ is spiteful, then he will prefer to reduce $\mathcal{A}$'s reward if it does not affect his. When $\mathcal{A}$ knows pre_a and tries to redeem *MH-Dep*, $\mathcal{B}$ cannot redeem it as well, but he can publish a redeeming transaction nonetheless allowing the miners to collect the tokens instead of $\mathcal{A}$.

We strengthen *MAD-HTLC* such that $\mathcal{B}$ 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 $\mathcal{B}$, or by any miner that provides both pre_a and pre_b. Now, if $\mathcal{A}$ knows pre_a then she can redeem *MH-Dep* and $\mathcal{B}$ can redeem *MH-Col*. If instead $\mathcal{B}$ contends with $\mathcal{A}$ for *MH-Dep*, both still lose the *MH-Dep* for the miners; but now both pre_a and pre_b are revealed, allowing miners to seize the *MH-Col* tokens as well. $\mathcal{B}$ is therefore strictly incentivized not to contend, allowing $\mathcal{A}$ to receive the *MH-Dep* tokens as required.

This means the *MAD-HTLC* construction is secure against the known incentive manipulation attacks [25]–[29] – $\mathcal{B}$ cannot incentivize miners to exclude $\mathcal{A}$'s transaction and getting his confirmed instead.

MAD-HTLC utilizes the *mutual assured destruction* [30], [31] principle: If a party misbehaves then all parties lose everything. Although penalizing the well-behaved party as well, this mechanism design [32] technique ensures rational players act as intended.

To prove the security of *MAD-HTLC* (§V), we first bound the possible leakage and interactions of system entities using the UC framework [33]. These interactions include the setup, initiation and redeeming of *MAD-HTLC*. Then, we formalize *MAD-HTLC* as a game played by $\mathcal{A}$, $\mathcal{B}$ and the miners, where the action space comprises the aforementioned possible interactions. We model all parties as rational non-myopic players, and show the prescribed behavior is *incentive-compatible* [34].

We prove the efficacy of *MAD-HTLC* by implementing it both in the less expressive Bitcoin Script [35] and in the richer Ethereum Solidity [36] smart-contract languages (§VI). We deploy it on Bitcoin's and Ethereum's main networks, and show it bears negligible overhead (e.g., 2.2e-6 BTC) compared to the secured amount (e.g., 2.6 BTC). Specifically for payment-channels [4]–[10], this negligible overhead is only incurred in the abnormal case of a dispute.

MAD-HTLC relies on miners non-myopically optimizing their transaction choices, often referred to as *Miner Extractable Value* (*MEV*) [37]–[39]. While such optimizations are common in the Ethereum network, as of today, Bitcoin's default cryptocurrency client only offers basic optimization. Changes in miners' revenue structure will make better optimizations more important. To demonstrate miners can easily enhance transaction choice optimization once they choose to do so, we patch the standard Bitcoin client [40] to create *Bitcoin-MEV infrastructure*, allowing 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 then revisit 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, and bribe can be made using the built-in transaction fee mechanism. We analyze miners' behavior as a game for the *HTLC* timeout duration. Each suffix of the game can be analyzed as a subgame, and all players have perfect knowledge of the system state. $\mathcal{B}$ can take advantage of this setting to incentivize miners to withhold $\mathcal{A}$'s transaction until the timeout, making this the single *subgame perfect equilibrium* [41]. So, in presence of rational non-myopic miners the required bribe cost is independent of the timeout duration. This matches the lower bound, and qualitatively tightens the exponential-in-timeout upper bound, both presented by Winzer et al. [28].

In our Bitcoin-compatible attack variation, 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. We identify several potential bribe opportunities on the Bitcoin and Ethereum main networks, including examples of a few dollars bribe would have yielded tens of thousands of dollars in payout (e.g., payment channel where bribe of \$2 could have yielded payout of \$25K).

We conclude by discussing future directions (§VIII), including attacks and mitigations in a weaker model where $\mathcal{A}$ or $\mathcal{B}$ have mining capabilities, and using *MAD-HTLC* to reduce latency in systems using *HTLC-Spec*.

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 miners as participants;
- prove *MAD-HTLC* is secure and incentive compatible;
- implement, deploy, and evaluate *MAD-HTLC* on the Bitcoin and Ethereum main networks;
- patch the prevalent Bitcoin Client to create Bitcoin-MEV infrastructure, and to specifically support enforcing correct *MAD-HTLC* execution;
- 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 completed a responsible-disclosure process with prominent blockchain development groups. We intend to open source our code, subject to security concerns of the community.

II. Related Work

We are not aware of prior work utilizing miners' incentives to use them as enforcers of correct smart contract execution. We review previous work on bribing attacks in blockchains (§II-A), detail exhibited and postulated mining behavior with respect to transaction selection (§II-B), and present systems and applications using *HTLC-Spec* (§II-C).

A. Bribery Attacks

Winzer et al. [28] present attacks that delay confirmation of specific transactions until a given timeout elapses. Their attacks apply to *HTLC* where $\mathcal{B}$ delays the confirmation of $\mathcal{A}$'s redeeming transaction until he can redeem it himself. Their presented attack requires predicates available only in rich smart contract languages like Ethereum's Solidity [36], [42] and Libra's Move [43], [44], but not Bitcoin's Script [35]. Specifically, the attack requires setting a *bribe contract* that monitors what blocks are created and rewards participants accordingly.

In contrast, our attack variation works with Bitcoin's Script as well, as we demonstrate by implementation. It therefore applies a wider range of systems [45]–[47].

Winzer et al. [28] present two results regarding the attack costs. First, they show that $\mathcal{B}$'s attack cost for making miner's collaboration with the attack a Nash-equilibrium grows linearly with the *size* (i.e., the relative mining capabilities) of the smallest miner. However, all miners not cooperating with the attack is also a Nash equilibrium. Therefore, they analyze $\mathcal{B}$'s cost for making the attack a dominant strategy, i.e., to incentivize a to support the attack 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* [31], [34], [41], [48]–[54] solution concept, a refinement of Nash-equilibrium suitable for games of dynamic nature. We consider the game played by non-myopic 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 [25]–[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.

A recent and parallel work [55] also suggests using Bitcoin's fee mechanism to attack *HTLC*. It assumes miners below a certain hash-rate threshold are myopic (sub-optimal) while those above it are non-myopic; it presents safe timeout values given Bitcoin's current hash-rate distribution. In this work, we assume all miners are non-myopic and prove that in this model the attack costs are independent of the timeout. We also present *MAD-HTLC*, which is secure against these attacks with both myopic and non-myopic miners.

B. Transaction-Selection Optimization

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

Indeed, previous work [28], [37], [56]–[61] shows 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 [62]–[68] asserts the profitability of such actions is expected to rise as the underlying systems mature, enabling constructions such as *MAD-HTLC*, which rely on these optimizations.

C. HTLC-Spec usage

A variety of smart contracts [16]–[24] critically rely on *HTLC-Spec*. To the best of our knowledge, all utilize *HTLC*, making them vulnerable once miners optimize their transaction choices. We review some prominent examples.

a) Off-chain state channels: A widely-studied smart contract construction [5]–[10], [12], [69]–[71] with implementations on various blockchains [4], [72]–[76] is that of an *off-chain channel* between two parties, $\mathcal{A}$ and $\mathcal{B}$.

The channel has a *state* that changes as $\mathcal{A}$ and $\mathcal{B}$ interact, e.g., pay one another by direct communication. In the simplest case, the state is represented by a *settlement transaction* that $\mathcal{B}$ can place on the blockchain. The settlement transaction terminates the channel by placing its final state back in the blockchain. The transaction initiates an *HTLC* with a hash digest of $\mathcal{B}$'s choice. $\mathcal{B}$ can redeem the contract after the timeout or, alternatively, $\mathcal{A}$ can redeem it before the timeout if $\mathcal{B}$ had shared the preimage with her.

When $\mathcal{A}$ and $\mathcal{B}$ interact and update the channel state, $\mathcal{B}$ *revokes* the previous settlement transaction by sending his preimage to $\mathcal{A}$. This guarantees that if $\mathcal{B}$ places a revoked settlement transaction on the blockchain, $\mathcal{A}$ can redeem the tokens within the timeout. Alternatively, if $\mathcal{A}$ becomes unresponsive, $\mathcal{B}$ 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 – $\mathcal{A}$ 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* [77]–[79] offer to replace $\mathcal{A}$ 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]–[15], [80], where a set of parties transact their assets in an atomic manner, i.e., either all transactions occur, or none.

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

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 digital signature scheme [81] with a security parameter μ. Additionally, they have access to a hash function $H : \{0,1\}^* \to \{0,1\}^\mu$, mapping inputs of arbitrary length to outputs of length μ. We assume the value of μ is sufficiently large such that the standard cryptographic assumptions hold: the digital signature scheme is existentially unforgeable under chosen message attacks (*EU-CMA*) [81], [82], and that H upholds preimage resistance [83], [84].

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 a number representing its relative block-creation rate, or *mining power*. Denote the mining power of miner i by λ_i, where $\sum_{i=1}^{n} \lambda_i = 1$. Denote the minimal mining power by $\lambda_{\text{min}} = \min_i \lambda_i$. As in previous work [28], [64], [85], [86], these rates are common knowledge, as in practice miners can monitor the blockchain and infer them [87].

Block creation is a discrete-time, memoryless stochastic process. At each time step exactly one miner creates a block. As in previous work [4], [9]–[11], [64], we disregard miners deliberately [85], [88], [89] or unintentionally [90]–[92] causing transient inconsistencies (called *forks* in the 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], [64], 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 to simplify presentation 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 her mining power, i.e., miner i is selected with probability λ_i. The selected miner creates block b_j, either empty or containing a single transaction, and adds it to the blockchain. This confirms the 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 input 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 confirming miner.

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 ones.

There is always at least one unconfirmed valid transaction in the mempool [64], [66], [68], [93], [94], 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 (see §II-B). Users are also rational, and offer the minimal sufficient fee for having their transactions confirmed.

Predicates have access to three primitives of interest:

- $\mathsf{vSig}\,(sig; pk)$: validate that a digital signature *sig* provided by the transaction (on the transaction, excluding *sig*) matches a public key *pk* specified in the contract.
- $\mathsf{vPreImg}\,(pre; dig)$: validate that a preimage *pre* provided by the transaction matches a hash digest *dig* specified in the contract, i.e., that $H(pre) = dig$.
- $\mathsf{vTime}\,(T)$: validate that the transaction trying to redeem 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]–[14], [17]–[19]. *HTLC-Spec* is specified for two users, $\mathcal{A}$ and $\mathcal{B}$. It is parameterized by a hash digest and a timeout, and contains a certain *deposit* amount, v^{dep}. $\mathcal{A}$ gets the deposit if she publishes a matching preimage before the timeout elapses, otherwise $\mathcal{B}$ does.

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

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

To construct *HTLC-Spec*, $\mathcal{A}$ and $\mathcal{B}$ choose the included hash digest, the timeout, and the token amount, v^{dep}. Then either of them issues a transaction that generates the contract with v^{dep} tokens and the parameterized predicate. Either $\mathcal{A}$ or $\mathcal{B}$ initially knows the preimage, depending on the scenario.

For simplicity, we assume that $\mathcal{A}$ 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 sub contracts[1] — *MH-Dep*, the core implementation of the *HTLC-Spec* functionality, and *MH-Col*, an auxiliary contract for collateral, used to disincentivize spiteful behavior by $\mathcal{B}$.

MAD-HTLC includes additional variables and parameters along those of *HTLC-Spec*, facilitating its realization. It includes two preimages, pre_a and pre_b; the former corresponds to the preimage of *HTLC-Spec*; the latter is an addition in *MAD-HTLC*, chosen by $\mathcal{B}$, used in the various redeem paths. It also includes the *HTLC-Spec* deposit token amount v^{dep}, but also utilizes v^{col} collateral tokens.

Essentially, *MH-Dep* lets either $\mathcal{A}$ redeem v^{dep} with preimage pre_a, or $\mathcal{B}$ after the timeout with preimage pre_b, or any party with both preimages pre_a and pre_b. *MH-Col* has v^{col} redeemable only after the timeout, either by $\mathcal{B}$, or by any party with both preimages pre_a and pre_b.

We present protocol $\Pi_{\text{mad-htlc}}$ for setup, initiation and redeeming of a *MAD-HTLC* (§IV-A), and detail the specifics of *MH-Dep* (§IV-B) and *MH-Col* (§IV-C).

A. Protocol $\Pi_{\textit{mad-htlc}}$

Recall that *HTLC-Spec* is used in several scenarios differing in which party chooses the preimage, when that chosen preimage is shared, and who initiates the contract on the blockchain (§II-C). However, in all scenarios, once the contract is initiated, $\mathcal{A}$ can redeem v^{dep} by publishing the preimage before the timeout elapses, and $\mathcal{B}$ can redeem them only after.

So, there are several variants for any protocol that implements *HTLC-Spec*, and we focus on the variant where $\mathcal{B}$ picks the first preimage pre_a, potentially shares it with $\mathcal{A}$, either $\mathcal{A}$ or $\mathcal{B}$ can initiate the contract on chain, and either can redeem it using the various redeem paths. This corresponds to the *off-chain payment channels* scenario (§II-C).

Protocol $\Pi_{\text{mad-htlc}}$ (Protocol 1) progresses in phases, and is parameterized by the timeout T and the token amounts v^{dep} and v^{col}. First, in the setup phase, $\mathcal{B}$ randomly draws (denoted by $\xleftarrow{R}$) the two preimages pre_a and pre_b. He then derives their respective hash digests $dig_a \leftarrow H(pre_a)$ and $dig_b \leftarrow H(pre_b)$, shares dig_a and dig_b with $\mathcal{A}$. Upon $\mathcal{A}$'s confirmation $\mathcal{B}$ creates a transaction tx_{init} that initiates a *MAD-HTLC* with parameters $T, dig_a, dig_b, v^{\text{dep}}, v^{\text{col}}$ and shares tx_{init} with $\mathcal{A}$.

In the following $\mathsf{initiation}$ phase, $\mathcal{B}$ can share pre_a with $\mathcal{A}$. Additionally, either $\mathcal{A}$ or $\mathcal{B}$ can publish tx_{init} to the mempool, allowing miners to confirm it and initiate the *MAD-HTLC*.

In the final $\mathsf{redeeming}$ phase, once the *MAD-HTLC* is initiated, $\mathcal{A}$ and $\mathcal{B}$ can redeem v^{dep} and v^{col} from *MH-Dep* and *MH-Col*, respectively. Specifically, $\mathcal{A}$ redeems v^{dep} only if she received pre_a from $\mathcal{B}$, and otherwise $\mathcal{B}$ redeems v^{dep}. Either way, $\mathcal{B}$ redeems v^{col}.

B. MH-Dep

The *MH-Dep* contract is initiated with v^{dep} tokens. Its predicate is parameterized with $\mathcal{A}$'s and $\mathcal{B}$'s public keys, pk_a and pk_b, respectively; a hash digest of the predefined preimage $dig_a = H(pre_a)$ such that any entity other than $\mathcal{A}$ and $\mathcal{B}$ does not know pre_a, and $\mathcal{A}$ or $\mathcal{B}$ know pre_a according to on the specific use case; another hash digest dig_b such that $H(pre_b) = dig_b$, where only $\mathcal{B}$ knows pre_b; and a timeout T. The contract has three redeem paths, denoted by *dep-*$\mathcal{A}$, *dep-*$\mathcal{B}$ and *dep-*$\mathcal{M}$, and presented in Predicate 1. Table I shows the possible redeeming entities of *MH-Dep*.

In the *dep-*$\mathcal{A}$ path (line 1), $\mathcal{A}$ can redeem *MH-Dep* by creating a transaction including pre_a and sig_a, a signature created using her secret key sk_a. Such a transaction can

[1] Separating *MAD-HTLC* into two sub contracts is for Bitcoin compatibility; these can be consolidated to a single contract in blockchains supporting, richer smart-contract languages, see §VI.

Protocol $\Pi_{\text{mad-htlc}}$ run by $\mathcal{A}$ and $\mathcal{B}$ details the setup, initiation and redeeming of a *MAD-HTLC* in the scenario where $\mathcal{B}$ picking pre_a. It is parameterized by timeout T, and token amounts v^{dep} and v^{col}.

setup

$\mathcal{B}$ draws $pre_a \xleftarrow{R} \{0,1\}^{\mu}, pre_b \xleftarrow{R} \{0,1\}^{\mu}$ and sets $dig_a \leftarrow H(pre_a), dig_b \leftarrow H(pre_b)$. Then $\mathcal{B}$ sends dig_a and dig_b to $\mathcal{A}$ for confirmation. Afterwards $\mathcal{B}$, compiles a transaction tx_{init} that initiates a *MAD-HTLC* (both *MH-Dep* and *MH-Col*) with $dig_a, dig_b, T, v^{\text{dep}}, v^{\text{col}}$ as parameters and shares it with $\mathcal{A}$. tx_{init} is not published yet.

initiation

$\mathcal{B}$ can send pre_a to $\mathcal{A}$. If so, $\mathcal{A}$ expects to receive pre such that $dig_a = H(pre)$, and ignores other values.
Either $\mathcal{A}$ or $\mathcal{B}$ publish tx_{init} to the mempool, and it is eventually included in a block b_j, initiating *MAD-HTLC*.

redeeming

If $\mathcal{A}$ had received pre_a, she creates and publishes tx_a^{dep}, a transaction redeeming *MH-Dep* using the *dep-$\mathcal{A}$* redeem path.
$\mathcal{B}$ waits for the creation of block b_{j+T-1}. If by then $\mathcal{A}$ did not publish tx_a^{dep} then $\mathcal{B}$ publishes $tx_b^{\text{dep+col}}$, redeeming both *MH-Dep* and *MH-Col* through *dep-$\mathcal{B}$* and *col-$\mathcal{B}$* redeem paths, respectively. If $\mathcal{A}$ did publish tx_a^{dep} then $\mathcal{B}$ publishes tx_b^{col}, redeeming only *MH-Col* using the *col-$\mathcal{B}$* redeem path.

Protocol 1: $\Pi_{\text{mad-htlc}}$

Predicate 1: *MH-Dep*

Parameters: $pk_a, pk_b, T, dig_a, dig_b$
MH-Dep $(pre_1, pre_2, sig) :=$
1 $(\mathsf{vPreImg}(pre_1; dig_a) \wedge \mathsf{vSig}(sig; pk_a)) \vee$ // dep-$\mathcal{A}$
2 $(\mathsf{vPreImg}(pre_2; dig_b) \wedge \mathsf{vSig}(sig; pk_b) \wedge \mathsf{vTime}(T)) \vee$ // dep-$\mathcal{B}$
3 $(\mathsf{vPreImg}(pre_1; dig_a) \wedge \mathsf{vPreImg}(pre_2; dig_b))$ // dep-$\mathcal{M}$

Table I: Possible redeeming entity of *MH-Dep*.

	pre_b *published*	pre_b *not published*
pre_a *published*	Any entity	$\mathcal{A}$
pre_a *not published*	$\mathcal{B}$	—

Table II: Possible redeeming entity of *MH-Col*.

	pre_b *published*	pre_b *not published*
pre_a *published*	Any entity	$\mathcal{B}$
pre_a *not published*	$\mathcal{B}$	$\mathcal{B}$

be included even in the next block b_{j+1}. This path is only available to $\mathcal{A}$, since only she ever knows sk_a.

In the *dep-$\mathcal{B}$* path (line 2), $\mathcal{B}$ can redeem *MH-Dep* by creating a transaction including pre_b and sig_b, 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 $\mathcal{B}$, since only he ever knows sk_b.

In the *dep-$\mathcal{M}$* path (line 3), any entity can redeem *MH-Dep* by creating a transaction including both pre_a and pre_b. A transaction taking this 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, and specifically to any miner, that knows both pre_a and pre_b.

C. *MH-Col*

The *MH-Col* contract is initiated with v^{col} tokens. Its predicate is parameterized with $\mathcal{B}$'s public key pk_b; the hash digest of the predefined secret $dig_a = H(pre_a)$ such that any entity other than $\mathcal{A}$ and $\mathcal{B}$ does not know pre_a, and $\mathcal{A}$ and $\mathcal{B}$ know pre_a based on the specific use case; the hash digest dig_b such that $H(pre_b) = dig_b$, where only $\mathcal{B}$ knows pre_b; and a timeout T. It has two redeem paths, denoted by *col-$\mathcal{B}$* and *col-$\mathcal{M}$*, and presented in Predicate 2. Table II shows the possible redeeming entities of *MH-Col*.

Predicate 2: *MH-Col*

Parameters: pk_b, T, dig_a, dig_b
MH-Col $(pre_1, pre_2, sig) :=$
1 $\mathsf{vTime}(T) \wedge$
2 $[\mathsf{vSig}(sig; pk_b) \vee$ // col-$\mathcal{B}$
3 $(\mathsf{vPreImg}(pre_1; dig_a) \wedge \mathsf{vPreImg}(pre_2; dig_b))]$ // col-$\mathcal{M}$

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 *col-$\mathcal{B}$* path (line 2), $\mathcal{B}$ can redeem *MH-Col* by creating a transaction including sig_b, a signature created using his secret key sk_b. Only $\mathcal{B}$ can redeem *MH-Col* using this path as he is the only one able to produce such a signature. This path allows $\mathcal{B}$ to claim the collateral tokens in case either he or $\mathcal{A}$, but not both, publish a transaction redeeming *MH-Dep*.

The *col-$\mathcal{M}$* path (line 3) allows any entity to redeem *MH-Col* by creating a transaction including both pre_a and pre_b, not requiring any digital signature. This path allows miners to claim the *MH-Col* tokens in case $\mathcal{B}$ tries contesting $\mathcal{A}$ on redeeming *MH-Dep*, thus disincentivizing his attempt.

V. *MAD-HTLC* Security Analysis

To prove the security of *MAD-HTLC* we first show what actions the participants can take to interact with it (§V-A). We prove with the UC framework [33] the security of the setup, initiation and redeeming of a *MAD-HTLC*. This analysis yields a set of conditions on which entity can redeem tokens from *MAD-HTLC*.

Then, we move to analyze how the entities should act to maximize their gains. We formalize the redeeming of an initiated *MAD-HTLC* as a game played by $\mathcal{A}$, $\mathcal{B}$ and the miners (§V-B), and show that they are all incentivized to act as intended (§V-C).

A. Setup, Initiation and Redeeming Transactions Security

Our first goal is to prove the setup and initiation of *MAD-HTLC* are secure and to show which valid transactions each participant can generate based on the mempool and blockchain state. We present an overview of the security claims and proofs, and bring the details in the extended report [95].

Like prior work [4], [9], [28], [31], [64], [81], [84], [96]–[107], we assume the blockchain and predicate security holds, including the digital signature scheme and the hash function.

We make the following observation: Transaction invalidity due to vTime is temporal; this predicate becomes true once sufficiently many blocks are created. In contrast, two valid transactions can conflict, so only one of them can be confirmed. We neglect both invalidity reasons and show which valid transactions can be created; clearly, any transaction that is invalid under this relaxation is also invalid without it. Additionally, we consider only transactions relevant to our protocol, ignoring unrelated transactions.

We formalize parties' ability to redeem the contract under this relaxation using the *rPred*() function (Eq. 1): Denote $path \in \{$*dep-$\mathcal{A}$*, *dep-$\mathcal{B}$*, *dep-$\mathcal{M}$*, *col-$\mathcal{B}$*, *col-$\mathcal{M}$*$\}$; $\mathcal{P}$ the redeeming party; $h_a = 1$ if the redeeming party has a suitable preimage for dig_a, and 0 otherwise; and $h_b = 1$ if the redeeming party has a suitable preimage for dig_b, and 0 otherwise. Then the relaxed contract predicate is expressed by the function $rPred\,(path, \mathcal{P}, h_a, h_b)$. We note redeeming transactions are published in the mempool, hence publish any included preimages.

$$rPred\,(path, \mathcal{P}, h_a, h_b) = \begin{cases} (\mathcal{P} = \mathcal{A}) \wedge h_a & path = dep\text{-}\mathcal{A} \\ (\mathcal{P} = \mathcal{B}) \wedge h_b & path = dep\text{-}\mathcal{B} \\ h_a \wedge h_b & path \in \{dep\text{-}\mathcal{M}, col\text{-}\mathcal{M}\} \\ \mathcal{P} = \mathcal{B} & path = col\text{-}\mathcal{B} \end{cases} \tag{1}$$

We move to consider the setup, initiation, and redeeming of a single (relaxed) contract with respect to the mempool and the blockchain. We focus on a *mempool and blockchain projection* (*mbp*) functionality of a relaxed *MAD-HTLC*, and we model it as a single ideal functionality, $\mathcal{G}_{\text{mbp}}$. This functionality captures the parameter setup of a single contract by $\mathcal{A}$ and $\mathcal{B}$, its initiation, and redeeming transaction validity due to the vPreImg and vSig predicates, disregarding conflicts and timeouts. To facilitate the vPreImg predicate and its underlying preimage-resistant hash function H, we model the latter as a *global random oracle* ideal functionality $\mathcal{H}$ [33], [108], [109]. We abstract away digital signatures by considering authenticated channels among parties and functionalities. We consider an adversary that learns messages sent to $\mathcal{G}_{\text{mbp}}$ and that messages were sent between parties but not their content. This modeling is similar to previous work [8], [9], [84], [98].

We then define the $(\mathcal{H}, \mathcal{G}_{\text{mbp}})$*-hybrid world* [33] (hereinafter, simply *the hybrid world*), where the $\mathcal{H}$ and $\mathcal{G}_{\text{mbp}}$ ideal functionalities reside. In this hybrid world we then define the *relaxed MAD-HTLC* (*rmh*) protocol Π_{rmh} that is similar to $\Pi_{\text{mad-htlc}}$ (Protocol 1), but (1) it is defined with $\mathcal{H}$ and $\mathcal{G}_{\text{mbp}}$; (2) it considers system entities other than $\mathcal{A}$ and $\mathcal{B}$, and specifically miners, represented as a third party $\mathcal{M}$; and (3) it disregards timeouts and transaction conflicts.

The transition from Π_{rmh} to $\Pi_{\text{mad-htlc}}$ is straightforward, and we bring Π_{rmh} in the extended report [95].

Then, our goal is to prove the following lemma, detailing the possible valid transactions the entities can create and publish.

Lemma 1. *Let there be a contract setup and initiated as described by Π_{rmh}, let* pub$_a$ *and* pub$_b$ *be indicators whether the preimages* pre$_a$ *and* pre$_b$ *were published in $\mathcal{G}_{mbp}$, respectively, and let* shared *indicate if $\mathcal{B}$ shared* pre$_a$ *with $\mathcal{A}$. So, initially* pub$_a$ $\leftarrow 0$*,* pub$_b$ $\leftarrow 0$ *and* shared $\leftarrow 0$*. Then, parties $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{M}$ can only create and publish the following valid redeeming transactions:*

- *$\mathcal{B}$ can publish a valid redeeming transaction using the* dep-$\mathcal{B}$*,* dep-$\mathcal{M}$*,* col-$\mathcal{B}$ *or* col-$\mathcal{M}$ *redeem paths. Doing so with either* dep-$\mathcal{M}$ *or* col-$\mathcal{M}$ *sets* pub$_a$ $\leftarrow 1$*, and with either* dep-$\mathcal{B}$*,* dep-$\mathcal{M}$*, or* col-$\mathcal{M}$ *sets* pub$_b$ $\leftarrow 1$*. In addition to transaction creation and publication, $\mathcal{B}$ can share* pre$_a$ *with $\mathcal{A}$ (and by doing so sets* shared $\leftarrow 1$*).*
- *If* pub$_a \vee$ shared $= 1$ *then $\mathcal{A}$ can publish a valid redeeming transaction using the* dep-$\mathcal{A}$ *redeem path (and by doing so she sets* pub$_a$ $\leftarrow 1$*). If* (pub$_a \vee$ shared) $\wedge$ pub$_b = 1$*, then $\mathcal{A}$ can publish a valid redeeming transaction with either the* dep-$\mathcal{M}$ *or* col-$\mathcal{M}$ *redeem paths (and by doing so sets* pub$_a$ $\leftarrow 1$ *and* pub$_b$ $\leftarrow 1$*).*
- *If* pub$_a \wedge$ pub$_b = 1$ *then $\mathcal{M}$ can publish a valid redeeming transaction with either* dep-$\mathcal{M}$ *or* col-$\mathcal{M}$*.*

Ideal functionality $\mathcal{F}_{\text{rmh}}$ in the ideal world represents the setup, initiation and redeeming transaction publication of the contract for session id *sid*. It interacts with parties $\mathcal{A}, \mathcal{B}, \mathcal{M}$, and simulator Sim. It internally stores indicators $setup_a^{\text{mh}}$, $setup_b^{\text{mh}}$, $shared^{\text{mh}}$, $published^{\text{mh}}$, $init^{\text{mh}}$, pub_1^{mh} and pub_2^{mh}, all with initial value of 0.

- Upon receiving (setup-B, *sid*) from $\mathcal{B}$ when $setup_b^{\text{mh}} = 0$, set $setup_b^{\text{mh}} \leftarrow 1$ and leak (setup-B, *sid*) to Sim.
- Upon receiving (setup-A, *sid*) from $\mathcal{A}$ when $setup_b^{\text{mh}} = 1 \wedge setup_a^{\text{mh}} = 0$, set $setup_a^{\text{mh}} \leftarrow 1$ and leak (setup-A, *sid*) to Sim.
- Upon receiving (share, *sid*) from $\mathcal{B}$ when $setup_a^{\text{mh}} = 1 \wedge shared^{\text{mh}} = 0$, set $shared^{\text{mh}} \leftarrow 1$, and leak (share, *sid*) to Sim.
- Upon receiving (publish, *sid*) from either $\mathcal{A}$ or $\mathcal{B}$ when $setup_a^{\text{mh}} = 1 \wedge published^{\text{mh}} = 0$, set $published^{\text{mh}} \leftarrow 1$, and leak (publish, *sid*) to Sim.
- Upon receiving (init, *sid*) from $\mathcal{M}$ when $published^{\text{mh}} = 1 \wedge init^{\text{mh}} = 0$, set $init^{\text{mh}} \leftarrow 1$, and leak (init, *sid*) to Sim.
- Upon receiving (redeem, *sid*, *path*) from any party $\mathcal{P}$ such that $path \in \{dep\text{-}\mathcal{A}, dep\text{-}\mathcal{B}, dep\text{-}\mathcal{M}, col\text{-}\mathcal{B}, col\text{-}\mathcal{M}\}$ when $init^{\text{mh}} = 1$, set $pub_1^{\text{mh}} \leftarrow pub_1^{\text{mh}} \vee \big((\mathcal{P} = \mathcal{A}) \wedge shared^{\text{mh}} \wedge (path = dep\text{-}\mathcal{A})\big) \vee \big((\mathcal{P} = \mathcal{B}) \wedge path \in \{dep\text{-}\mathcal{A}, dep\text{-}\mathcal{M}, col\text{-}\mathcal{M}\}\big)$ and $pub_2^{\text{mh}} \leftarrow pub_2^{\text{mh}} \vee \big((\mathcal{P} = \mathcal{B}) \wedge path \in \{dep\text{-}\mathcal{B}, dep\text{-}\mathcal{M}, col\text{-}\mathcal{M}\}\big)$, denote $res^{\text{mh}} \leftarrow rPred\left(path, \mathcal{P}, pub_1^{\text{mh}}, pub_2^{\text{mh}}\right)$, leak (redeem, *sid*, *path*, $\mathcal{P}$) to Sim, and return res^{mh} to $\mathcal{P}$.
- Upon receiving (update, *sid*, i) for $i \in \{0, 1\}$ from Sim through the influence port, set $pub_i^{\text{mh}} \leftarrow 1$.

Functionality 1: $\mathcal{F}_{\text{rmh}}$ in the ideal world.

To prove Lemma 1 we consider an ideal world, where we define a *relaxed MAD-HTLC* ideal functionality $\mathcal{F}_{\text{rmh}}$ (Functionality 1) that implements the setup, initiation and redeeming of a relaxed *MAD-HTLC* contract.

$\mathcal{F}_{\text{rmh}}$ maintains indicators $setup_a^{\text{mh}}$, $setup_b^{\text{mh}}$, $shared^{\text{mh}}$, $published^{\text{mh}}$, $init^{\text{mh}}$, and pub_1^{mh} and pub_2^{mh}, corresponding to execution of *MAD-HTLC* (Protocol 1): $setup_a^{\text{mh}}$ and $setup_b^{\text{mh}}$ correspond to to $\mathcal{A}$ and $\mathcal{B}$ completing their setup; $shared^{\text{mh}}$ is set if $\mathcal{B}$ shared *pre$_a$* with $\mathcal{B}$; $published^{\text{mh}}$ and $init^{\text{mh}}$ indicate if the execution reached the initiation and redeeming phases, respectively; and pub_1^{mh} and pub_2^{mh} are set if *pre$_a$* and *pre$_b$* are published with a transaction.

$\mathcal{F}_{\text{rmh}}$ leaks messages to Sim and receives a special update instruction that sets either pub_1^{mh} or pub_2^{mh}. Looking ahead, this allows the simulator to notify $\mathcal{F}_{\text{rmh}}$ of a publication by a corrupted party.

The construction of $\mathcal{F}_{\text{rmh}}$ and the definition of *rPred* (Eq. 1) imply that the properties described by Lemma 1 trivially hold in the ideal world.

We then prove Π_{rmh} UC-realizes $\mathcal{F}_{\text{rmh}}$, i.e., for any PPT adversary Adv, there exists a PPT simulator Sim such that for any PPT environment $\mathcal{Z}$, the execution of Π_{rmh} in the hybrid world with Adv is computationally indistinguishable from the execution of $\mathcal{F}_{\text{rmh}}$ in the ideal world with Sim.

We prove the aforementioned by showing how to construct such a Sim for any Adv, and the derived indistinguishability towards $\mathcal{Z}$. Sim internally manages two preimages on its own, which are indistinguishable from to those chosen by $\mathcal{B}$ in the hybrid world: for an honest $\mathcal{B}$, Sim draws these two preimages from the same distribution as in the real world; for a corrupted $\mathcal{B}$, Sim learns the chosen preimages throughout the execution. Additionally, Sim internally-simulates $\mathcal{H}$ and $\mathcal{G}_{\text{mbp}}$, and interacts with $\mathcal{F}_{\text{rmh}}$ through leakage and influence ports.

The existence of these simulators shows Lemma 1 applies to the hybrid world as well, meaning it details the possible valid redeeming transactions of a relaxed *MAD-HTLC*.

Recall the relaxed version considers only the vPreImg and vSig predicates while disregarding vTime and transactions conflicts, which we now move to consider.

B. MAD-HTLC Game

The *MAD-HTLC* construction within the blockchain system gives rise to a game: the participants are $\mathcal{A}$, $\mathcal{B}$ and the system miners; their utilities are their tokens; and the action space is detailed by Lemma 1 while considering the timeout constraints and transaction conflicts.

Note that Lemma 1 considers party $\mathcal{M}$ representing any system miner, while the upcoming analysis considers all the miners and their individual rewards.

The *MAD-HTLC* game begins when the *MH-Dep* and *MH-Col* contracts are initiated in some block b_j. The game, which we denote by Γ^{MH}, comprises T rounds, representing the creation of blocks $b_{j+1}, ..., b_{j+T}$. Each round begins with $\mathcal{A}$ and $\mathcal{B}$ publishing redeeming transactions, followed by a miner creating a block including a transaction of her choice.

$\mathcal{A}$ and $\mathcal{B}$'s strategies are their choices of published transactions – which transactions to publish, when, and with what fee. Miner strategies are the choices of which transaction to include in a block if they are chosen to create one.

To accommodate for the stochastic nature of the game [110] we consider entity utilities as the expected number of tokens they own at game conclusion, i.e., after the creation of T blocks. $\mathcal{A}$ and $\mathcal{B}$'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 (§V-B1) and the suitable solution concept (§V-B2).

1) Game Details: The game progresses in rounds, where each round comprises two steps. First, $\mathcal{A}$ and $\mathcal{B}$ alternately publish transactions, until neither wishes 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 $\mathcal{A}$ and $\mathcal{B}$ may offer, meaning the publication step is finite.

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

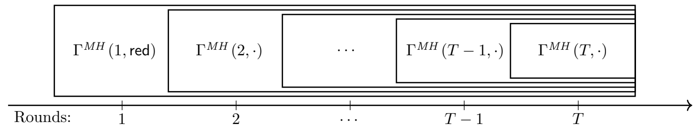

Figure 1: Γ^{MH} subgames.

a) Subgames: The dynamic and turn-altering nature of the game allows us to define *subgames*, representing suffixes of Γ^{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 the *MH-Dep* is already redeemed, future blocks do not allow inclusion of conflicting transactions that redeem the *MH-Dep* as well.

Hence, when considering *MAD-HTLC* states we 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 $s \in \{\text{red}, \text{irred}\}$. We denote such a subgame by $\Gamma^{MH}(k, s)$.

We use $\cdot$ to denote sets of subgames, e.g., $\Gamma^{MH}(\cdot, \text{red})$ denotes 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 Γ^{MH} is complete. We refer to all other subgames as *non-final*.

The game begins when there are T blocks to be created, $\mathcal{A}$ and $\mathcal{B}$ 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: $\mathcal{A}$ and $\mathcal{B}$'s actions are the publication of transactions in any $\Gamma^{MH}(\cdot, \cdot)$ subgame.

$\mathcal{A}$ can only redeem *MH-Dep* and only if she has pre_a ($shared = 1$), hence has a single transaction of interest tx_a^{dep}, offering fee of f_a^{dep} tokens. Note tx_a^{dep} has to outbid unrelated transactions and thus has to offer a fee $f_a^{\text{dep}} > f$, however, cannot offer more tokens than the redeemed ones, so $f_a^{\text{dep}} < v^{\text{dep}}$. This transaction utilizes the *dep-*$\mathcal{A}$ redeem path of *MH-Dep*, hence publishing it also publishes pre_a.

$\mathcal{B}$ can redeem *MH-Dep*, *MH-Col* or both. We thus consider three transactions of interest: tx_b^{dep}, redeeming *MH-Dep* while offering fee f_b^{dep}; tx_b^{col}, redeeming *MH-Col* while offering fee f_b^{col}; and $tx_b^{\text{dep+col}}$, redeeming both *MH-Dep* and *MH-Col* while offering fee $f_b^{\text{dep+col}}$. To redeem *MH-Dep* $\mathcal{B}$ uses the *dep-*$\mathcal{B}$ redeem path, hence publishing transactions tx_b^{dep} or $tx_b^{\text{dep+col}}$ also publishes pre_b. Redeeming *MH-Col* is by the *col-*$\mathcal{B}$ redeem path. Similarly to $\mathcal{A}$'s fee considerations, $\mathcal{B}$'s transactions have to outbid unrelated transactions, and cannot offer more tokens than they redeem, so $f < f_b^{\text{dep}} < v^{\text{dep}}$, $f < f_b^{\text{col}} < v^{\text{col}}$ and $f < f_b^{\text{dep+col}} < v^{\text{dep}} + v^{\text{col}}$.

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

She can also include any of the following transactions, assuming they were previously published by $\mathcal{A}$ or $\mathcal{B}$, and as a function of the contract state: tx_a^{dep} if *MH-Dep* is redeemable, that is, in any $\Gamma^{MH}(\cdot,\text{red})$; tx_b^{col} if the timeout has elapsed, that is, in any $\Gamma^{MH}(T,\cdot)$; and tx_b^{dep} or $tx_b^{\text{dep+col}}$ if the timeout has elapsed and *MH-Dep* is redeemable, that is, in $\Gamma^{MH}(T,\text{red})$.

Conditioned on knowing pre_a and pre_b through published transactions, a miner can also create and include the following transactions, redeeming the contracts herself:

- Transaction tx_m^{dep} redeeming *MH-Dep*, using the *dep-*$\mathcal{M}$ redeem path, and getting the v^{dep} tokens of *MH-Dep* as reward. This action is only available if the miner knows both pre_a and pre_b, and if *MH-Dep* is redeemable, that is, in any $\Gamma^{MH}(\cdot,\text{red})$ subgame where tx_a^{dep} and either of tx_b^{dep} or $tx_b^{\text{dep+col}}$ were published.
- Transaction tx_m^{col} redeeming *MH-Col*, using the *col-*$\mathcal{M}$ redeem path, and getting the v^{col} tokens of *MH-Col* as reward. This action is only available if the miner knows both pre_a and pre_b, and the timeout has elapsed, that is, in any $\Gamma^{MH}(T,\text{red})$ subgame where tx_a^{dep} and either of tx_b^{dep} or $tx_b^{\text{dep+col}}$ were published.
- Transaction $tx_m^{\text{dep+col}}$ redeeming both *MH-Dep* and *MH-Col*, using the *dep-*$\mathcal{M}$ and *col-*$\mathcal{M}$ redeem paths, and getting the $v^{\text{dep}}+v^{\text{col}}$ tokens of *MH-Dep* and *MH-Col* as reward. This action is only available if the miner knows both pre_a and pre_b, the *MH-Dep* is redeemable, and the timeout has elapsed, that is, in subgame $\Gamma^{MH}(T,\text{red})$ where tx_a^{dep} and either of tx_b^{dep} or $tx_b^{\text{dep+col}}$ were published.

We disregard actions that are trivially dominated [52], such as $\mathcal{A}$ and $\mathcal{B}$ sharing their secret keys or publishing the relevant preimages not via a transaction; a miner including a transaction of another entity that redeems either of the contracts using the two preimages instead of redeeming it herself; and a miner creating an empty block instead of including an unrelated transaction.

c) Strategy: A *strategy* σ 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 $\bar{\sigma}$.

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 $\bar{\sigma}$ in subgame $\Gamma^{MH}(k,s)$ by $u_i\left(\bar{\sigma},\Gamma^{MH}(k,s)\right)$.

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* [111], [112]. We are thus interested in strategy profiles that are *subgame perfect equilibria* [41], [48]–[54].

A strategy profile $\bar{\sigma}$ 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 $\bar{\sigma}$ are a Nash equilibrium.

We say that a prescribed strategy profile is *incentive compatible* [34] 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* [54], [113]–[115], suitable for perfect-information finite games. Intuitively, to determine her best action, a player analyzes the game outcome for each possible action, repeating the process recursively for each possible game suffix.

C. MAD-HTLC Incentive Compatibility

We now show the *MAD-HTLC* prescribed behavior (Protocol 1) is incentive compatible and implements *HTLC-Spec*.

We first analyze $\mathcal{A}$'s and $\mathcal{B}$'s utilities when both follow the prescribed strategy, starting with the scenario where $\mathcal{A}$ knows the preimage pre_a (i.e., when $shared = 1$).

Lemma 2. *In $\Gamma^{MH}(1,red)$, if $\mathcal{A}$ knows pre_a and $\mathcal{A}$ and $\mathcal{B}$ both follow the prescribed strategies, then miners' best-response strategy leads to $\mathcal{A}$ redeeming MH-Dep for $v^{dep}-f_a^{dep}$ tokens, and $\mathcal{B}$ redeeming MH-Col for $v^{col}-f_b^{col}$ tokens.*

Proof. The prescribed strategy states that $\mathcal{A}$ publishes tx_a^{dep} during the first $T-1$ rounds, and that $\mathcal{B}$ publishes tx_b^{col} in round T.

Note that $\mathcal{B}$ does not publish tx_b^{dep} and $tx_b^{\text{dep+col}}$, hence miners do not know pre_b. The transactions tx_a^{dep} and tx_b^{col} offer f_a^{dep} and f_b^{col} fees, respectively, both greater than the base fee f.

The induced subgames therefore enable miners to include tx_a^{dep} in one of the first $T-1$ blocks, and including tx_b^{col} in the last one. Using backward induction shows the subgame perfect equilibrium is to include tx_a^{dep} in its published round, and tx_b^{col} in the last.

So both tx_a^{dep} and tx_b^{col} are included in blocks, and $\mathcal{A}$ and $\mathcal{B}$ get $v^{\text{dep}}-f_a^{\text{dep}}$ and $v^{\text{col}}-f_b^{\text{col}}$ tokens, respectively. □

We now consider $\mathcal{A}$ that does not know pre_a (i.e., when $shared = 0$).

Lemma 3. *In $\Gamma^{MH}(1,red)$, if $\mathcal{A}$ does not know pre_a and $\mathcal{A}$ and $\mathcal{B}$ both follow the prescribed strategies, then miners' best-response strategy leads to $\mathcal{B}$ redeeming both MH-Dep and MH-Col for $v^{dep}+v^{col}-f_b^{dep+col}$ tokens, and $\mathcal{A}$ gets none.*

Proof. As $\mathcal{A}$ does not know pre_a she does not publish any transaction, hence redeems no contract and receives no tokens.

By the prescribed strategy $\mathcal{B}$ publishes $tx_b^{\text{dep+col}}$, offering fee $f_b^{\text{dep+col}} > f$ and revealing pre_b. However, pre_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 $\mathcal{A}$ and $\mathcal{B}$ get 0 and $v^{\text{dep}}+v^{\text{col}}-f_b^{\text{dep+col}}$ tokens, respectively. □

We now present three lemmas, considering potential deviations from the prescribed strategy, and showing that any such deviation is strictly dominated. We provide the gist of the proofs, with the details deferred to Appendix A.

We first show that if $\mathcal{A}$ and $\mathcal{B}$ contend then the miners do not take their transactions in the last round.

Lemma 4. *In the last round of the game, i.e. subgame* $\Gamma^{MH}(T, \cdot)$*, if* tx_a^{dep} *and either* tx_b^{dep} *or* $tx_b^{dep+col}$ *are published then miners' best-response strategy is not to include any of* $\mathcal{A}$*'s or* $\mathcal{B}$*'s transactions in this round.*

This holds because in the described scenario any miner can simply redeem all the tokens herself. Then we show $\mathcal{A}$'s cannot deviate to increase her utility.

Lemma 5. *In* $\Gamma^{MH}(1, \textsf{red})$*,* $\mathcal{A}$ *cannot increase her utility by deviating from the prescribed strategy.*

This holds as publishing at the last round or not publishing at all results with $\mathcal{A}$ not getting any tokens. Similarly, we claim $\mathcal{B}$ does not gain from deviating.

Lemma 6. *In* $\Gamma^{MH}(1, \textsf{red})$*,* $\mathcal{B}$ *cannot increase his utility by deviating from the prescribed strategy.*

Intuitively, if $\mathcal{B}$ publishes when $\mathcal{A}$ also does then $\mathcal{B}$ loses all the tokens, whilst refraining from doing so earns him the collateral.

Following directly from Lemma 5 and Lemma 6), 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 1. *MAD-HTLC satisfies HTLC-Spec with rational PPT participants.*

Proof. Lemma 1 shows the possible redeeming transactions for PPT participants, disregarding invalidity due to timeouts and transaction conflicts. Consequently, the game description considering the timeouts and conflicts (§V-B) captures the possible redeeming transactions of PPT participants.

The game analysis (Corollary 1) shows the prescribed strategy (Protocol 1) is incentive compatible, and Lemma 2 and Lemma 3 show the prescribed strategy matches *HTLC-Spec*. Note that matching *HTLC-Spec*, Protocol 1 states the redeeming transaction fee should be negligibly larger than f, and is independent of v^{dep}. □

Myopic Miners: *MAD-HTLC*'s design deters $\mathcal{B}$ from bribe attempts as he knows rational non-myopic miners will seize his funds if he acts dishonestly.

However, even in the presence of unsophisticated, myopic miners, *MAD-HTLC* still satisfies *HTLC-Spec*. The common transaction selection logic [116]–[119] as of today has miners myopically optimize for the next block. Since $\mathcal{B}$'s transaction can only be confirmed in the last round, these miners will simply include $\mathcal{A}$'s transaction, achieving the desired outcome.

Table III: Bitcoin contract and redeeming transaction sizes.

Contract	Size [bytes]	Redeem path	Redeeming tx [bytes]
HTLC	99	*htlc-*$\mathcal{A}$	291
		htlc-$\mathcal{B}$	259
MH-Dep	129	*dep-*$\mathcal{A}$	323
		dep-$\mathcal{B}$	322
		dep-$\mathcal{M}$	282
MH-Col	88	*col-*$\mathcal{B}$	248
		col-$\mathcal{M}$	241

Only miners that are sophisticated enough to be non-myopic but not sophisticated enough to take advantage of the *dep-*$\mathcal{M}$ 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 *dep-*$\mathcal{M}$ path during the T rounds in order to thwart the attack.

VI. *MAD-HTLC* Implementation

We demonstrate the efficacy of *MAD-HTLC* by evaluating it in Bitcoin and Ethereum. We discuss the deployment of *MAD-HTLC* and its overhead (§VI-A), and our implementation of a framework for implementing MEV infrastructure [37]–[39] on Bitcoin (§VI-B), used to facilitate *MAD-HTLC* guarantees.

A. Contract Implementation, Overhead and Deployment

We implement *MH-Dep* and *MH-Col* in Bitcoin's Script [35] and Ethereum's Solidity [36], [42] smart contract languages. We also implement a version of the standard *HTLC* for reference. We bring the code in Appendix B.

We briefly discuss these implementations, show their transaction-fee overhead is negligible compared to the secured amounts, and present main network deployments.

Bitcoin implementation: Bitcoin's transaction fees are determined by the transaction sizes. Our contracts use *P2SH* [120] (non SegWit [121]) addresses, so the initiating transactions contain only the hashes of the scripts, and each contract initiation within a transaction requires 28 bytes. The redeeming transactions provide the full predicate script along with its inputs. 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 [122] and exchange rate [123] implies an additional cost of a mere \$0.02. Including the auxiliary *MH-Col* implies an additional cost of about \$0.10.

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.

Ethereum implementation: Compared to Bitcoin's Script, Solidity [36], [42] is a richer smart contract language, allowing *MAD-HTLC* to be expressed as a single contract consolidating *MH-Dep* and *MH-Col*.

On the Ethereum platform transactions pay fees according to their so-called *gas* usage, an inner form of currency describing the cost of each operation the transaction performs. We compare the initiation and redeeming costs of *HTLC* and *MAD-HTLC*. Note that *MAD-HTLC* contains about twice the

Table IV: Ethereum gas for contract initiation and redeeming.

Contract	Initiation [gas]	Redeem path	Redeeming [gas]
HTLC	362,000	*htlc-*$\mathcal{A}$	34,734
		htlc-$\mathcal{B}$	32,798
MAD-HTLC	600,000	*dep-*$\mathcal{A}$	58,035
		dep-$\mathcal{B}$	58,885
		dep-$\mathcal{M}$	59,043
		col-$\mathcal{B}$	41,175
		col-$\mathcal{M}$	44,887

code of *HTLC*, and as expected, its operations are more gas-consuming. We bring the details in Table IV.

We stress these numbers regard the most basic, straightforward implementation, and that Ethereum and Solidity enable further optimizations – for example, deploying a more elaborate *library contract* once [124], and simpler *contract instances* that use the library, achieving significantly reduced amortized costs. More importantly, the additional fee costs are independent of (e.g., \$0.2 [125]), and can be negligible compared to, the secured amounts (e.g., \$6.2K [125]).

Recall that for off-chain channels this overhead is incurred only in the abnormal unilateral channel closure.

Main network deployment: We deployed *MAD-HTLC* on both blockchains (Appendix C details the transaction IDs).

For Bitcoin, we deployed three *MH-Dep* instances on the main network and redeemed them using its three redeem paths. We also deployed two *MH-Col* instances and redeemed using its two redeem paths.

For Ethereum, we deployed a consolidated *MAD-HTLC*, and posted transactions redeeming the v^{dep} through both *dep-*$\mathcal{A}$ and *dep-*$\mathcal{B}$. These transactions offered relatively low fees, so were not included in a block by any miner, and only revealed pre_a and pre_b. At this point there were no other transactions trying to redeem the *MAD-HTLC*, although users and miners monitoring the blockchain could have created a transaction redeeming the v^{dep} using the *dep-*$\mathcal{M}$ with the revealed pre_a and pre_b. We deduce this optimization currently does not take place on the Ethereum main network.

Then, we published a transaction of our own using *dep-*$\mathcal{M}$, revealing (again) pre_a and pre_b, offering a relatively-high fee. Nevertheless, our transaction was slightly out-bid by another transaction, which also used *dep-*$\mathcal{M}$, and took the deposit. It was likely published by a front-running bot [37], [56], [60], presenting yet another example of entities monitoring the blockchain looking for MEV opportunities [37], [56], [60], as required for *MAD-HTLC* security.

B. Bitcoin-MEV Infrastructure

By default, cryptocurrency clients [116]–[119] only perform myopic transaction-inclusion optimizations, trying to generate a single maximal-fee block each time. As recently shown [37], [56], [60] (including in our deployment above), miners and other entities perform more sophisticated optimizations on the Ethereum network.

In contrast, we are not aware of similar optimizations taking place on the Bitcoin network. Specifically, Bitcoin Core, which is used by roughly 97% of current Bitcoin nodes [40], maintains a local *mempool* data structure that only contains unconfirmed transactions whose timeouts (if any) have elapsed.

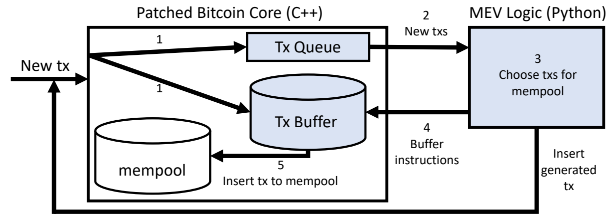

Figure 2: Bitcoin-MEV, new components shaded.

This implementation prevents miners from optimizing based on transaction pending on a timeout. However, this limitation is not a consensus rule, but an implementation choice. Taking more elaborate considerations into account when choosing transactions is not a violation of legitimate miner behavior.

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

Bitcoin-MEV's main design goal is to enable users to deploy their own optimization algorithms. It comprises a patched (140 LoC) C++ Bitcoin Core node with additional RPCs, and a Python script (Fig. 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.

We implemented and locally tested a Python script (350 LoC) for enforcing *MAD-HTLC* by taking advantage of the opportunities it provides miners to increase their revenue. We screen received transactions, tease out $\mathcal{A}$ and $\mathcal{B}$'s preimages, and create a transaction redeeming the *MAD-HTLC* contracts using the extracted preimages.

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 $\mathcal{A}$'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 [57], [58], [62]–[67], and include transactions that benefit their personal gains [37], [56], [59], [61]. Specifically, $\mathcal{B}$ can incentivize miners with a bribe [26]–[28] to exclude $\mathcal{A}$'s transaction until the timeout elapses, and then redeem the *HTLC* himself.

Predicate 3: *HTLC*

Parameters: pk_a, pk_b, T, dig_a

$HTLC\,(pre, sig) :=$

```
1   (vPreImg (pre; dig_a) ∧ vSig (sig; pk_a)) ∨      // htlc-A
2   (vSig (sig; pk_b) ∧ vTime (T))                   // htlc-B
```

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

A. Construction

$\mathcal{A}$ and $\mathcal{B}$ execute *HTLC-Spec* by having an *HTLC* contracted with v^{dep} tokens placed in some block b_j. The *HTLC*'s predicate is parameterized with $\mathcal{A}$'s and $\mathcal{B}$'s public keys, pk_a and pk_b, respectively; a hash digest of the predefined secret $dig_a = H\,(pre_a)$ such that any entity other than $\mathcal{A}$ and $\mathcal{B}$ does not know pre_a ($\mathcal{A}$ or $\mathcal{B}$ know pre_a based on the specific use case); and a timeout T.

HTLC has two redeem paths, denoted *htlc-*$\mathcal{A}$ and *htlc-*$\mathcal{B}$, and presented in Predicate 3. In *htlc-*$\mathcal{A}$ (line 1), $\mathcal{A}$ can redeem with a transaction including pre_a and sig_a, a signature with her secret key sk_a. In *htlc-*$\mathcal{B}$ (line 2), $\mathcal{B}$ can redeem with a transaction including sig_b, a signature with his secret key sk_b. This transaction can only be included in a block at least T blocks after *HTLC*'s initiation, that is, block b_{j+T}.

As only $\mathcal{A}$ and $\mathcal{B}$ know their respective secret keys, other entities cannot redeem the contract.

The intended way $\mathcal{A}$ and $\mathcal{B}$ should interact with *HTLC* is as follows. If $\mathcal{A}$ knows the predefined preimage pre_a, she publishes a transaction tx_a^{h} offering a fee $f_a^{\text{h}} > f$ 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 $\mathcal{A}$ does not know the predefined preimage pre_a she does not publish any transactions.

$\mathcal{B}$ observes the published transactions in the mempool, watching for tx_a^{h}. If by block b_{j+T-1} $\mathcal{A}$ did not publish tx_a^{h} then $\mathcal{B}$ publishes tx_b^{h} with a fee $f_b^{\text{h}} > f$, redeeming the *HTLC*. If $\mathcal{A}$ did publish tx_a^{h} by block b_{j+T-1} then $\mathcal{B}$ does not publish any transactions.

B. HTLC Game

HTLC operation gives rise to a game, denoted by Γ^H, played among $\mathcal{A}$, $\mathcal{B}$ and the miners. It is similar to that of the *MAD-HTLC* game (§V-B), so we present the differences.

a) Subgames: The game state is simply the number of blocks (k) created so far and state of the *HTLC*, which can be either redeemable (red) or irredeemable (irred), so denoted $\Gamma^H\,(k, \text{red}/\text{irred})$.

The game begins when one block (initiating the *HTLC*) was created, $\mathcal{A}$ and $\mathcal{B}$ did not publish any transactions, and the *HTLC* is redeemable. Thus, the initial, complete game is $\Gamma^H\,(1, \text{red})$.

b) Actions: $\mathcal{A}$ can redeem the *HTLC* with a transaction tx_a^{h}, offering f_a^{h} tokens as fee. Note tx_a^{h} has to outbid unrelated transactions and thus has to offer a fee $f_a^{\text{h}} > f$, however, cannot offer more tokens than the redeemed ones, so $f_a^{\text{h}} < v^{\text{dep}}$. $\mathcal{A}$ redeems *HTLC* using the *htlc-*$\mathcal{A}$ redeem path, so tx_a^{h} can be confirmed in any round.

$\mathcal{B}$ can redeem *HTLC* with a transaction tx_b^{h}, offering f_b^{h} tokens as fee. Similarly, f_b^{h} is bounded such that $f < f_b^{\text{h}} < v^{\text{dep}}$. $\mathcal{B}$ redeems *HTLC* using the *htlc-*$\mathcal{B}$ redeem path, so tx_b^{h} can only be confirmed in the last round.

Any miner can include the following transactions: an unrelated transaction in any $\Gamma^H\,(\cdot,\cdot)$ subgame; tx_a^{h} in any $\Gamma^H\,(\cdot, \text{red})$ subgame; and tx_b^{h} in the $\Gamma^H\,(T, \text{red})$ subgame.

C. Bribe Attack Analysis

We now show the *HTLC* prescribed strategy (§VII-A) is not incentive compatible. Specifically, we show that if $\mathcal{A}$ commits to the prescribed strategy, then $\mathcal{B}$ strictly gains by publishing a conflicting transaction, outbidding $\mathcal{A}$'s fee, thus incentivizing miners to exclude $\mathcal{A}$'s transaction and include his instead.

Let $\mathcal{A}$ publish tx_a^{h} with fee f_a^{h} in the first round, and $\mathcal{B}$ publish a transaction tx_b^{h} with fee $f_b^{\text{h}} > \frac{f_a^{\text{h}} - f}{\lambda_{\min}} + f$. Focusing on miner actions, we show through a series of lemmas they are incentivized to include tx_b^{h} and to exclude tx_a^{h}, resulting with lower utility for $\mathcal{A}$, higher utility for $\mathcal{B}$, and a violation of the *HTLC-Spec*.

First, we show miner utilities for subgames where the HTLC is irredeemable. Denote by $\bar{\sigma}$ the best response strategy of all miners in this setting.

Lemma 7. *For any $k \in [1, T]$, the utility of miner i in subgame $\Gamma^H\,(k, \text{irred})$ is $u_i\left(\bar{\sigma}, \Gamma^H\,(k, \text{irred})\right) = \lambda_i\,(T-k+1)\,f$.*

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 λ_i. This scenario can be viewed as a series of $T-k+1$ Bernoulli trials with success probability λ_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 $u_i\left(\bar{\sigma}, \Gamma^H\,(k, \text{irred})\right) = \lambda_i\,(T-k+1)\,f$. □

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 8. *Choosing to include tx_b^{h} is a unique subgame perfect equilibrium in $\Gamma^H\,(T, \text{red})$, and miner i's utility when doing so is $u_i\left(\bar{\sigma}, \Gamma^H\,(T, \text{red})\right) = \lambda_i f_b^{\text{h}}$.*

Proof. In the $\Gamma^H\,(T, \text{red})$ subgame, the miner that creates the block has three transactions to pick from: an unrelated transaction for the base fee f, tx_a^{h} for f_a^{h}, or tx_b^{h} for f_b^{h}.

As $f_b^{\text{h}} > \frac{f_a^{\text{h}} - f}{\lambda_{\text{min}}} + f$, $0 < \lambda_{\text{min}} < 1$ and $f_a^{\text{h}} > f$, it follows that $f_b^{\text{h}} > f_a^{\text{h}}$ and $f_b^{\text{h}} > f$. That means including tx_b^{h} 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 λ_i, and so her expected profit, i.e. utility, is $u_i\left(\bar{\sigma}, \Gamma^H\left(T, \text{red}\right)\right) = \lambda_i f_b^{\text{h}}$. □

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 9. *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 $u_i\left(\bar{\sigma}, \Gamma^H(k, \text{red})\right) = \lambda_i\left((T-k)f + f_b^h\right)$.*

To prove this lemma we show that for any $k \in [1, T-1]$, including $\mathcal{A}$'s transaction in subgame $\Gamma^H(k, \text{red})$ results with lower overall utility at game conclusion – intuitively, it redeems the contract, so in the last subgame miners cannot include $\mathcal{B}$'s transaction. The proof is by induction on k, and we bring it in full in Appendix D.

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

Theorem 2. *Alice's prescribed behavior of HTLC allows $\mathcal{B}$ to bribe miners at a cost of $\frac{f_a^h - f}{\lambda_{min}} + f$.*

Proof. The proof follows directly from Lemma 8 and Lemma 9, both showing that if $\mathcal{A}$ naively follows the prescribed strategy then subgame perfect equilibrium of the initial subgame is for all miners to place unrelated transactions until round T and then place $\mathcal{B}$'s transaction. □

Note that by Theorem 2, 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 $\mathcal{A}$ sees an attack is taking place she can respond by increasing her fee. In turn, this could lead to $\mathcal{B}$ increasing his fee as well, and so forth. Instead of focusing on these bribe and counter-bribe dynamics, we conclude by showing that $\mathcal{A}$ can preemptively prevent the attack, or assure winning with a counter-bribe, by paying a high fee dependent on v^{dep},. We note that such a high fee is in violation of the *HTLC-Spec*.

Corollary 2. *$\mathcal{B}$ cannot bribe the miners in this manner if $\mathcal{A}$'s tx_a^h offers at least $f_a^h > \lambda_{min}\left(v^{dep} - f\right) + f$.*

Proof. In order to achieve the attack, $\mathcal{B}$ ought to make placing unrelated transactions until T and placing his transaction at T a subgame perfect equilibrium. As shown (Theorem 2), the threshold to incentivize the smallest miner is $f_b^{\text{h}} > \frac{f_a^{\text{h}} - f}{\lambda_{\text{min}}} + f$. Recall the fee f_b^{h} of the bribing transaction tx_b^{h} is upper bounded by the *HTLC* tokens v^{dep}. Therefore, to achieve the attack it must hold that $v^{\text{dep}} > \frac{f_a^{\text{h}} - f}{\lambda_{\text{min}}} + f$. By choosing $f_a^{\text{h}} > \lambda_{\text{min}}\left(v^{\text{dep}} - f\right) + f$, $\mathcal{A}$ can prevent $\mathcal{B}$ from paying a fee adhering to the bounds. □

Myopic Miners: This bribery attack variant 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 $\mathcal{A}$'s transaction and $\mathcal{B}$'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, $\mathcal{B}$ has to increase his offered bribe exponentially in T.

The analysis relies on assumptions on the mining power distribution, and 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. Non-Myopic Bribery-Accepting Miner Implementation

Aside from the Bitcoin-MEV infrastructure (§VI-B), we also implemented a simpler Bitcoin Core patch supporting the mentioned bribe attack on *HTLC*.

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 2.

The simplicity of this patch (150 LoC, no external modules) 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^{dep}, the base fee f, and the ratio of required tokens for bribery resistance (Theorem 2) and the base fee $\frac{\lambda_{\text{min}}\left(v^{\text{dep}} - f\right) + f}{f}$. To estimate the base fee we conservatively take the actual paid fee, which is an upper bound. We conservatively estimate $\lambda_{\text{min}} = 0.01$ [87]; miners with lower mining power are less likely due to economy-of-scale [126].

The first example is of a Bitcoin Lightning channel [127], [128], where the required fee to secure the contract against a bribery is 1.34e4 times the actual fee. Plugging in \$10K as the average Bitcoin price at the time [123], we get that an attack requires about a \$2 bribe for a payoff of over \$25K. Note this is just an arbitrary example, and there are plenty of such low-fee, high-capacity channels, in all a few dollars bribe is sufficient to yield tens of thousands of dollars as reward [127].

Table V: *HTLC* bribe resistance cost examples.

Name	v^{dep}	f	$\frac{\lambda_{\min}\left(v^{\text{dep}}-f\right)+f}{f}$
Lightning channel (BTC) [127], [128]	2.684	2.22e-6	1.34e4
Litecoin atomic swap (LTC) [129]	1.337	3.14e-4	435.7
Liquality [130] atomic swap (ETH) [125]	12	0.0004	301
Liquality [130] atomic swap (BTC) [131]	0.278	5.76e-6	483.63

The second example is of a Litecoin atomic swap [129], requiring 436 times higher fee to be secured against bribes. The last two examples are the two sides of a BTC-ETH atomic swap conducted by Liquality [130], requiring more than 300X and 480X fees to be secure, respectively.

VIII. Future Directions

We briefly present two future research directions. First, we discuss attacks and mitigations in a weaker model, where either $\mathcal{A}$ or $\mathcal{B}$ have significant mining power (§VIII-1). Then, we discuss how using *MAD-HTLC* can reduce latency in systems utilizing *HTLC-Spec* (§VIII-2)

1) Mining $\mathcal{A}$ or $\mathcal{B}$: As in previous work [26], [28], the security analysis of *MAD-HTLC* assumes that $\mathcal{A}$ and $\mathcal{B}$ have no mining capabilities and do not collude with any miner. Indeed, acquiring mining capabilities (or forming collusion agreements) requires a significant investment, substantially higher than necessary for a simple bribe. Removing this assumption extends the game space considerably, and brings in timing and probability considerations that are outside the scope of this work. Nevertheless, we briefly present the issue and a potential low-overhead modification that disincentivizes such attacks.

$\mathcal{A}$ with mining capabilities that knows pre_a can stall until the timeout elapses and $\mathcal{B}$ publishes pre_b, and then redeem both *MH-Dep* (using either *dep-*$\mathcal{A}$ or *dep-*$\mathcal{M}$) and *MH-Col* (using *col-*$\mathcal{M}$). This requires $\mathcal{A}$ to create the block right after the timeout elapses, otherwise another miner would include $\mathcal{B}$'s transactions. The potential profit is the v^{col} tokens, whose number is in the order of a transaction fee.

$\mathcal{B}$ with mining capabilities can redeem *MH-Dep* (using *dep-*$\mathcal{M}$) if he knows pre_a. This requires $\mathcal{B}$ to create the first block after the *MAD-HTLC* initiation, otherwise another miner would include $\mathcal{A}$'s transaction. The potential damage for this case is similar to the *HTLC* bribery (Winzer et al. [28] and §VII-C), and note that any miner will be able to redeem *MH-Col* once the timeout elapses.

Both variants require the miner to reveal pre_a and pre_b by creating a block at a specific height, meaning they only succeed with some probability. As such, their profitability depends on the relative mining size of the miner, the deposit and collateral amounts, and the transaction fees.

Nevertheless, these are vulnerabilities of *MAD-HTLC*, and we propose the following countermeasure: Instead of having a single *MH-Dep* and a single *MH-Col*, have multiple of each, all with the same dig_a and dig_b, but each with a different timeout T, and split v^{dep} and v^{col} among them.

As one of the timeouts elapse, if the miner attacks then she loses her advantage, as once she exposes pre_a and pre_b, any miner can compete for the remaining contracts. Therefore, this mechanism diminishes the attack profitability.

This adjustment's overhead is only due to the fees for creating and redeeming more contracts. However, those can be small, independent of the secured amount.

2) Latency Reduction: Systems utilizing *HTLC-Spec* must set the timeout parameter T, facing a trade-off. Too short timeouts result in a security risk – $\mathcal{B}$ might get the tokens unjustly because $\mathcal{A}$'s transaction was not yet confirmed. Too long timeouts imply an opportunity cost due to the unavailability of the locked coins, and increase susceptibility to various attacks [12], [132], [133].

MAD-HTLC can allow for significantly reduced timeouts compared to *HTLC*, since instead of waiting for confirmation, it now suffices to consider transaction publication. The analysis depends on mempool and congestion properties that are outside the scope of this work.

IX. 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 using the UC framework and with game-theoretic analysis that *MAD-HTLC* is secure. We also show the prevalent *HTLC* is vulnerable to cheap bribery attacks in a wider variety of systems, and qualitatively tighten the known cost bound in presence of rational miners. We demonstrate the efficacy of our approach by implementing and executing *MAD-HTLC* on Bitcoin and Ethereum. 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.

X. Acknowledgments

We thank the anonymous reviewers, Sebastian Faust, and our shepherd Dominique Schröder for their valuable feedback and guidance. This research was supported by the Israel Science Foundation (grant No. 1641/18), an IC3 research grant, the US-Israel Binational Science Foundation (BSF), 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/coins/
[4] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[5] C. Decker and R. Wattenhofer, "A fast and scalable payment network with Bitcoin Duplex Micropayment Channels," in *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium*, 2015.
[6] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies," in *Proceedings of the 2017 ACM CCS*, 2017.
[7] P. McCorry, M. Möser, S. F. Shahandasti, and F. Hao, "Towards bitcoin payment networks," in *Australasian Conference on Information Security and Privacy*, 2016.
[8] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Financial Cryptography and Data Security*, 2019.
[9] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proceedings of the 2018 ACM CCS*, 2018.
[10] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual payment channels over cryptographic currencies." *IACR ePrint*, 2017.
[11] M. Herlihy, "Atomic cross-chain swaps," in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, 2018.
[12] G. Malavolta, P. Moreno-Sanchez, C. Schneidewind, A. Kate, and M. Maffei, "Anonymous multi-hop locks for blockchain scalability and interoperability." in *NDSS*, 2019.
[13] R. van der Meyden, "On the specification and verification of atomic swap smart contracts," in *IEEE ICBC*, 2019.
[14] M. H. Miraz and D. C. Donald, "Atomic cross-chain swaps: development, trajectory and potential of non-monetary digital token swap facilities," *Annals of Emerging Technologies in Computing (AETiC) Vol*, 2019.
[15] J.-Y. Zie, J.-C. Deneuville, J. Briffaut, and B. Nguyen, "Extending atomic cross-chain swaps," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, 2019.
[16] G. Maxwell, "The first successful zero-knowledge contingent payment." [Online]. Available: https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/
[17] M. Campanelli, R. Gennaro, S. Goldfeder, and L. Nizzardo, "Zero-knowledge contingent payments revisited: Attacks and payments for services," in *Proceedings of the 2017 ACM CCS*, 2017.
[18] W. Banasik, S. Dziembowski, and D. Malinowski, "Efficient zero-knowledge contingent payments in cryptocurrencies without scripts," in *European Symposium on Research in Computer Security*, 2016.
[19] G. Fuchsbauer, "Wi is not enough: Zero-knowledge contingent (service) payments revisited," in *Proceedings of the 2019 ACM CCS*, 2019.
[20] S. Bursuc and S. Kremer, "Contingent payments on a public ledger: models and reductions for automated verification," in *European Symposium on Research in Computer Security*, 2019.
[21] M. Möser, I. Eyal, and E. G. Sirer, "Bitcoin covenants," in *Financial Cryptography and Data Security*, 2016.
[22] P. McCorry, M. Möser, and S. T. Ali, "Why preventing a cryptocurrency exchange heist isn't good enough," in *Cambridge International Workshop on Security Protocols*, 2018.
[23] Bryan Bishop, "Bitcoin vaults with anti-theft recovery/clawback mechanisms." [Online]. Available: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2019-August/017231.html
[24] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. Knottenbelt, "Xclaim: Trustless, interoperable, cryptocurrency-backed assets," in *2019 IEEE S&P*, 2019.
[25] J. Bonneau, "Why buy when you can rent?" in *Financial Cryptography and Data Security*, 2016.
[26] P. McCorry, A. Hicks, and S. Meiklejohn, "Smart contracts for bribing miners," in *FC*, 2018.
[27] A. Judmayer, N. Stifter, A. Zamyatin, I. Tsabary, I. Eyal, P. Gaži, S. Meiklejohn, and E. Weippl, "Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies," 2019.
[28] F. Winzer, B. Herd, and S. Faust, "Temporary censorship attacks in the presence of rational miners," in *2019 IEEE EuroS&PW*, 2019.
[29] J. Harris and A. Zohar, "Flood & loot: A systemic attack on the lightning network," in *Proceedings ACM AFT*, 2020.
[30] D. Deudney, *Whole Earth Security: A Geopolitics of Peace. Worldwatch Paper 55.*, 1983.
[31] A. Asgaonkar and B. Krishnamachari, "Solving the buyer and seller's dilemma: A dual-deposit escrow smart contract for provably cheat-proof delivery and payment for a digital good without a trusted mediator," in *IEEE ICBC*, 2019.
[32] S. Tadelis, *Game theory: an introduction*, 2013.
[33] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC*, 2007.
[34] T. Roughgarden, "Algorithmic game theory," *Communications of the ACM*, 2010.
[35] Wikipedia, "Bitcoin script." [Online]. Available: https://en.bitcoin.it/wiki/Script
[36] Ethereum. (2020) Solidity language. [Online]. Available: https://github.com/ethereum/solidity
[37] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE S&P*.
[38] A. Shevchenko. (2020) Researcher suggests miners are manipulating ethereum blocks to exploit defi. [Online]. Available: https://cointelegraph.com/news/researcher-suggests-miners-are-manipulating-ethereum-blocks-to-exploit-defi
[39] E. Felten. (2020) Meva (what is it good for?). [Online]. Available: https://medium.com/offchainlabs/meva-what-is-it-good-for-de8a96c0e67c
[40] C. Dance. (2020) Bitcoin client software distribution. [Online]. Available: https://coin.dance/nodes
[41] R. Selten, "Spieltheoretische behandlung eines oligopolmodells mit nachfrageträgheit: Teil i: Bestimmung des dynamischen preisgleichgewichts," *Zeitschrift für die gesamte Staatswissenschaft/Journal of Institutional and Theoretical Economics*, 1965.
[42] C. Dannen, *Introducing Ethereum and Solidity*, 2017.
[43] S. Blackshear, E. Cheng, D. L. Dill, V. Gao, B. Maurer, T. Nowacki, A. Pott, S. Qadeer, D. R. Rain, S. Sezer *et al.*, "Move: A language with programmable resources," 2019.
[44] M. Baudet, A. Ching, A. Chursin, G. Danezis, F. Garillot, Z. Li, D. Malkhi, O. Naor, D. Perelman, and A. Sonnino, "State machine replication in the libra blockchain," 2018.
[45] Litecoin Project, "Litecoin, open source P2P digital currency," 2014. [Online]. Available: https://litecoin.org
[46] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, 2016.
[47] B. Cash. (2020) Bitcoin cash. [Online]. Available: https://www.bitcoincash.org/
[48] R. W. Rosenthal, "Games of perfect information, predatory pricing and the chain-store paradox," *Journal of Economic theory*, 1981.
[49] D. Fudenberg and J. Tirole, "Game theory, 1991," *Cambridge, Massachusetts*, 1991.
[50] R. Myerson, "Game theory: Analysis of conflict harvard univ," *Press, Cambridge*, 1991.
[51] E. Van Damme, "Strategic equilibrium," *Handbook of game theory with economic applications*, 2002.
[52] J. Watson, *Strategy: an introduction to game theory*, 2002.
[53] J. Cerny, "Playing general imperfect-information games using game-theoretic algorithms," Ph.D. dissertation, PhD thesis, Czech Technical University, 2014.
[54] B. D. Bernheim, "Rationalizable strategic behavior," *Econometrica: Journal of the Econometric Society*, 1984.
[55] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, "Timelocked bribing."
[56] D. Robinson and G. Konstantopoulos. (2020) Ethereum is a dark forest. [Online]. Available: https://medium.com/@danrobinson/ethereum-is-a-dark-forest-ecc5f0505dff
[57] K. Malinova and A. Park, "Market design with blockchain technology," *Available at SSRN 2785626*, 2017.
[58] Y. Doweck and I. Eyal, "Multi-party timed commitments," 2020.
[59] S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: front-running attacks on blockchain," 2019.
[60] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," *arXiv preprint arXiv:2009.14021*, 2020.
[61] A. Munro. (2018) Fomo3d ethereum ponzi game r1 ends as hot play outmaneuvers bots. [Online]. Available: https://www.finder.com.au/fomo3d-ethereum-ponzi-game-r1-ends-as-hot-play-outmaneuvers-bots
[62] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proceedings of the 2019 ACM CCS*, 2019.
[63] J. Prestwich, "Miners aren't your friends," 2018. [Online]. Available: https://blog.keep.network/miners-arent-your-friends-cde9b6e0e9ac
[64] I. Tsabary and I. Eyal, "The gap game," in *ACM CCS*, 2018.

[65] J. Sliwinski and R. Wattenhofer, "Blockchains cannot rely on honesty."
[66] 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 CCS*, 2016.
[67] I. Tsabary, A. Spiegelman, and I. Eyal, "Heb: Hybrid expenditure blockchain," *arXiv*, 2019.
[68] D. Easley, M. O'Hara, and S. Basu, "From mining to markets: The evolution of bitcoin transaction fees," *Journal of Financial Economics*, 2019.
[69] R. Khalil and A. Gervais, "Revive: Rebalancing off-blockchain payment networks," in *Proceedings of the 2017 ACM CCS*, 2017.
[70] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Off the chain transactions." *IACR ePrint*, 2019.
[71] L. Aumayr, O. Ersoy, A. Erwig, S. Faust, K. Hostáková, M. Maffei, P. Moreno-Sanchez, and S. Riahi, "Generalized bitcoin-compatible channels."
[72] Blockstream. (2020) Blockstream lightning implementation. [Online]. Available: https://blockstream.com/lightning/
[73] L. Labs. (2020) Lightning labs lightning implementation. [Online]. Available: https://lightning.engineering/
[74] ——. (2020) Acinq lightning implementation. [Online]. Available: https://acinq.co/
[75] Raiden. (2020) Raiden network. [Online]. Available: https://raiden.network/
[76] O. Network, "Omg network blockchain design." [Online]. Available: https://docs.omg.network/blockchain-design
[77] G. Avarikioti, O. S. T. Litos, and R. Wattenhofer, "Cerberus channels: Incentivizing watchtowers for bitcoin," *Financial Cryptography and Data Security (FC)*, 2020.
[78] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration outsourcing for state channels," in *Proceedings of ACM AFT*, 2019.
[79] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, "Outpost: A responsive lightweight watchtower," in *Proceedings of ACM AFT*, 2019.
[80] E. Wagner, A. Völker, F. Fuhrmann, R. Matzutt, and K. Wehrle, "Dispute resolution for smart contract-based two-party protocols," in *IEEE ICBC*, 2019.
[81] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," in *Annual International Cryptology Conference*, 2017.
[82] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on computing*, 1988.
[83] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *International workshop on fast software encryption*, 2004.
[84] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *ACM CCS*, 2018.
[85] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security*, 2014.
[86] A. Sapirshtein, Y. Sompolinsky, and A. Zohar, "Optimal selfish mining strategies in Bitcoin," in *FC*, 2016.
[87] blockchain.info, "Bitcoin mining pools." [Online]. Available: https://blockchain.info/pools
[88] andes, "Bitcoin's kryptonite: The 51% attack." June 2011. [Online]. Available: https://bitcointalk.org/index.php?topic=12435
[89] M. Mirkin, Y. Ji, J. Pang, A. Klages-Mundt, I. Eyal, and A. Juels, "Bdos: Blockchain denial-of-service," in *ACM CCS*, 2020.
[90] J. A. Garay, A. Kiayias, and N. Leonardos, "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.
[91] 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*, 2017.
[92] L. Kiffer, R. Rajaraman, and A. Shelat, "A better method to analyze blockchain consistency," in *Proceedings of the 2018 ACM CCS*, 2018.
[93] blockchain.info, "Mempool transaction count." [Online]. Available: https://blockchain.info/charts/mempool-count
[94] R. Lavi, O. Sattath, and A. Zohar, "Redesigning bitcoin's fee market," in *The World Wide Web Conference*, 2019.
[95] I. Tsabary, M. Yechieli, and I. Eyal, "Mad-htlc: because htlc is crazy-cheap to attack (extended version)," *arXiv:2006.12031*, 2020.
[96] A. Kiayias, H.-S. Zhou, and V. Zikas, "Fair and robust multi-party computation using a global transaction ledger," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2016.
[97] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE EuroS&P*, 2019.
[98] A. Kiayias and O. S. T. Litos, "A composable security treatment of the lightning network," in *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*, 2020.
[99] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM CCS*, 2016.
[100] M. Wohrer and U. Zdun, "Smart contracts: security patterns in the ethereum ecosystem and solidity," in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018.
[101] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart contract security: A software lifecycle perspective," *IEEE Access*, 2019.
[102] E. Zhou, S. Hua, B. Pi, J. Sun, Y. Nomura, K. Yamashita, and H. Kurihara, "Security assurance for smart contract," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018.
[103] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *International conference on financial cryptography and data security*, 2016.
[104] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *International conference on principles of security and trust*, 2017.
[105] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018.
[106] L. Breidenbach, P. Daian, A. Jules, and E. G. Sirer, "An in-depth look at the parity multisig bug," 2017. [Online]. Available: https://hackingdistributed.com/2017/07/22/deep-dive-parity-bug/
[107] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy *et al.*, "Formal verification of smart contracts: Short paper," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016.
[108] R. Canetti, A. Jain, and A. Scafuro, "Practical uc security with a global random oracle," in *Proceedings of the 2014 ACM CCS*, 2014.
[109] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven, "The wonderful world of global random oracles," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, 2018.
[110] J.-F. Mertens and A. Neyman, "Stochastic games," *International Journal of Game Theory*, 1981.
[111] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*, 2008.
[112] M. J. Osborne and A. Rubinstein, *A course in game theory*, 1994.
[113] E. Zermelo, "Über eine anwendung der mengenlehre auf die theorie des schachspiels," in *Proceedings of the fifth international congress of mathematicians*, 1913.
[114] R. J. Aumann, "Backward induction and common knowledge of rationality," *Games and Economic Behavior*, 1995.
[115] M. M. Kamiński, "Backward induction: Merits and flaws," *Studies in Logic, Grammar and Rhetoric*, 2017.
[116] B. Core, "Bitcoin core." [Online]. Available: https://bitcoin.org/en/bitcoin-core/
[117] G. Ethereum, "Geth." [Online]. Available: https://geth.ethereum.org/
[118] P. Wallet, "Parity." [Online]. Available: https://www.parity.io/ethereum/
[119] E. S. Exchange, "Ethereum client transaction ordering." [Online]. Available: https://ethereum.stackexchange.com/questions/6107/what-is-the-default-ordering-of-transactions-during-mining-in-e-g-geth/
[120] G. Andresen. (2013) Bip 16 : Pay to script hash. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki
[121] P. W. Eric Lombrozo, Johnson Lau. (2017) Bip 141 : Segregated witness (consensus layer). [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki
[122] blockchain.info, "Bitcoin transaction fees," 2020. [Online]. Available: https://www.blockchain.com/charts/mempool-state-by-fee-level
[123] CoinDesk. (2020) Bitcoin price. [Online]. Available: https://www.coindesk.com/price/bitcoin

[124] E. developer documents. (2020) Smart contract libraries. [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/libraries/
[125] Liquality. (2020) Liquality btc-eth atomic swap, ethereum transaction. [Online]. Available: shorturl.at/lpDP7
[126] N. Arnosti and S. M. Weinberg, "Bitcoin: A natural oligopoly," *arXiv preprint arXiv:1811.08572*, 2018.
[127] 1ml. (2020) Lightning top capacity channel. [Online]. Available: https://1ml.com/channel/684407505443356673
[128] Blockstream. (2020) Lightning top capacity channel closing transaction. [Online]. Available: shorturl.at/jwNO7
[129] insight.litecore. (2017) Litecoin atomic swap spend. [Online]. Available: https://insight.litecore.io/tx/6c27cffab8a86f1b3be1ebe7acfbbbdcb82542c5cfe7880fcca60eab36747037
[130] Liquality. (2020) Liquality atomic swaps. [Online]. Available: https://liquality.io/swap/
[131] ——. (2020) Liquality btc-eth atomic swap, bitcoin transaction. [Online]. Available: https://blockstream.info/address/bc1qhmjx0mk3p2trxe829lxgc9hwqjdzv0ueshmmgl7yfqdlnje45q4st7q9q8
[132] S. Tikhomirov, P. Moreno-Sanchez, and M. Maffei, "A quantitative analysis of security, anonymity and scalability for the lightning network." *IACR Cryptol. ePrint Arch.*, 2020.
[133] A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," *arXiv preprint arXiv:2002.06564*, 2020.

APPENDIX A
MAD-HTLC INCENTIVE COMPATIBILITY LEMMA PROOFS

This section presents the proofs for Lemma 4, Lemma 5 and Lemma 6. We recall the lemmas for readability.

Lemma (4). *In the last round of the game, i.e. subgame $\Gamma^{MH}(T,\cdot)$, if tx_a^{dep} and either tx_b^{dep} or $tx_b^{dep+col}$ are published then miners' best-response strategy is not to include any of $\mathcal{A}$'s or $\mathcal{B}$'s transactions in this round.*

Proof. Since $\mathcal{A}$ and $\mathcal{B}$ published their transactions, both pre_a and pre_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 $\left(\Gamma^{MH}(T,\mathsf{irred})\right)$, then miners can create tx_m^{col} and redeem *MH-Col* themselves in round T, getting v^{col} tokens as reward. Alternatively, if tx_b^{col} is published they can include it in a block, getting a fee of f_b^{col} tokens. As $f_b^{col} < v^{col}$, including tx_b^{col} is strictly dominated by including tx_m^{col}. In this case miners can also not include tx_a^{dep} as the *MH-Dep* is irredeemable.

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

Either way, including any of $\mathcal{A}$'s or $\mathcal{B}$'s transactions results with a strictly lower reward, hence miners avoid doing so. $\square$

Lemma (5). *In $\Gamma^{MH}(1,\mathsf{red})$, $\mathcal{A}$ cannot increase her utility by deviating from the prescribed strategy.*

Proof. First, if $\mathcal{A}$ does not know pre_a, she can take no action, hence trivially complies with the prescribed strategy.

If $\mathcal{A}$ does know pre_a, then her possible deviations are not publishing tx_a^{dep} at all, or publishing it only in the last round T.

Not publishing tx_a^{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 $\mathcal{B}$ also follows the prescribed strategy (Lemma 2).

The inclusion of tx_a^{dep} in the last block depends on what transactions $\mathcal{B}$ publishes throughout the game (Lemma 4). That is, if $\mathcal{B}$ published either tx_b^{dep} or $tx_b^{dep+col}$ then miners' best-response is not to include tx_a^{dep}, and $\mathcal{A}$ gets no tokens. Otherwise, miners' best response is to include the transaction that offers the highest fee, which can be either tx_a^{dep} or another, resulting with $\mathcal{A}$ receiving $v^{dep}-f_a^{dep}$ and 0 tokens, respectively.

So, $\mathcal{A}$ cannot gain, and in several scenarios strictly lose, by deviating from her prescribed strategy. $\square$

Lemma (6). *In $\Gamma^{MH}(1,\mathsf{red})$, $\mathcal{B}$ cannot increase his utility by deviating from the prescribed strategy.*

Proof. Consider all of $\mathcal{B}$'s possible actions. His potential maximal utility is from having $tx_b^{dep+col}$ included, which he obtains by following the prescribed strategy in the scenario where $\mathcal{A}$ does not know pre_a (Lemma 3). So, he has no incentive to deviate in this case.

Now, consider the case where $\mathcal{A}$ knows pre_a, hence according to Lemma 5 publishes tx_a^{dep} in the first $T-1$ rounds.

$\mathcal{B}$ can publish tx_b^{dep}, $tx_b^{dep+col}$ and tx_b^{col} throughout the game. If he publishes tx_b^{dep} or $tx_b^{dep+col}$ in any round then none of his transactions are included (Lemma 4) and he gets no reward. However, if he only publishes tx_b^{col} then by Lemma 2 he receives $v^{col}-f_b^{col}>0$ tokens.

Not publishing tx_b^{col} at all results with the minimal utility of 0, and an earlier publication still leads miners to include both tx_b^{col} and tx_a^{dep} (cf. 6), obtaining the same utility as of the prescribed behavior. $\square$

APPENDIX B
MAD-HTLC BITCOIN AND ETHEREUM IMPLEMENTATIONS

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 dig_a and dig_b.

If the first matches dig_a but the second does not match dig_b (*dep-$\mathcal{A}$*), then the script verifies the existence of a third data element, and that it is a signature created with $\mathcal{A}$'s secret key.

If the first does not match dig_a but the second matches dig_b (*dep-$\mathcal{B}$*), then the script verifies the existence of a third data element, and that it is a signature created with $\mathcal{B}$'s secret key. It also verifies the timeout has elapsed.

If both the first and the second data elements match dig_a and dig_b (*dep-$\mathcal{M}$*), respectively, then the script expects no third data element and evaluates successfully.

MH-Dep

```
OP_HASH160
dig_a
OP_EQUAL
OP_SWAP
OP_HASH160
dig_b
OP_EQUAL
OP_IF
        OP_IF
                OP_1
        OP_ELSE
                T
                OP_CHECKSEQUENCEVERIFY
                OP_DROP
                pk_b
                OP_CHECKSIG
        OP_ENDIF
OP_ELSE
        OP_VERIFY
        pk_a
        OP_CHECKSIG
OP_ENDIF
```

Redeem path	Input data
1	sig_a OP_0 pre_a
2	sig_b pre_b OP_0
3	pre_b pre_a

MH-Col

```
T
OP_CHECKSEQUENCEVERIFY
OP_DROP
OP_HASH160
dig_a
OP_EQUAL
OP_IF
        OP_HASH160
        dig_b
        OP_EQUAL
OP_ELSE
        pk_b
        OP_CHECKSIG
OP_ENDIF
```

Redeem path	Input data
1	sig_b OP_0
2	pre_b pre_a

HTLC

```
OP_HASH160
dig_a
OP_EQUAL
OP_IF
        pk_a
OP_ELSE
        T
        OP_CHECKSEQUENCEVERIFY
        OP_DROP
        pk_b
OP_ENDIF
OP_CHECKSIG
```

Redeem path	Input data
1	sig_a pre_a
2	sig_b OP_0

Figure 3: *MH-Dep*, *MH-Col* and *HTLC* Bitcoin Script implementations.

Table VI: Bitcoin main-net experiment transaction IDs.

Description	Transaction ID
Initiate *MH-Dep*	d032175260145055860296cbca8f7462 4f30334ddf948d5da12f0c7414d80cc0
MH-Dep path 1	33c957bb2f75e797d240a38504ce49a3 aeaaceb72f8577096b4f2ff23f5b3a1e
MH-Dep path 2	cd090c90afaacc0e2648834fe96f6177 ec2f967b7e50245537afdaf0d5a80263
MH-Dep path 3	505c7f1f3862b7f5c6b78f72cce5e37a 655b946fbdc7d03526055f7ea206781a
Initiate *MH-Col*	ea830dba56000b3486cf1c5122fedcf8 8169ab596536fd406b4f989e7761c1b4
MH-Col path 1	4c06ebff8de6bb56242c75849767a633 9e40a0442f815a2487fd9d6237c51b9f
MH-Col path 2	68270b94ca80281e31e193dac6779d3a 22d2799fe2afff8cef66c0ec6b420c88

Table VII: Ethereum main-net experiment transaction IDs.

Description	Transaction ID
Initiation	f10be5e53b9ad8a6f10d7e9b9bfbd63a b8737c50274885182a67e7adc3fa59c2
dep-$\mathcal{A}$	36e349b4fdc5385ef57a88d077837223 b3a26b0e6afc75f90bbaf2860d9295fd
dep-$\mathcal{B}$	84aa626d659b63e0554f8de1a3d6e204 41d8d778b7e1e79d0a36ded325afedb4
dep-$\mathcal{M}$ (ours)	ebdb267e8b612d59910bc2348a95eec8 388e62dbd6d64458c982f0cdacea67d9
dep-$\mathcal{M}$ (other)	74e87bba99ccd7a0bd794b793f108674 5b462390df01594ce057a430c122635a

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 dig_a.

If not (*col-$\mathcal{B}$*), the script then verifies the second data is a signature created with $\mathcal{B}$'s secret key. Otherwise (*col-$\mathcal{M}$*), the script hashes the second data element and verifies it matches dig_b.

We bring the Ethereum Solidity implementation of *MAD-HTLC* and *HTLC* in the extended report [95].

Appendix C
MAD-HTLC Bitcoin and Ethereum Deployment

Tables VI and VII show the transaction IDs in our Bitcoin and Ethereum deployments (§VI-A), respectively. Their details can be viewed with online block explorers.

Appendix D
HTLC Bribe Attack Analysis Proof

We recall Lemma 9 and prove it.

Lemma (9). *For any $k \in [1, T-1]$, the unique subgame perfect equilibrium is that every miner includes an unrelated transaction in $\Gamma^H(k, \mathsf{red})$, and miner i's utility when doing so is $u_i(\bar{\sigma}, \Gamma^H(k, \mathsf{red})) = \lambda_i((T-k)f + f_b^h)$.*

Proof. Note that in $\Gamma^H(k, \mathsf{red})$ there are two actions available, either include an unrelated transaction and receive f reward, or include tx_a^{h} and receive f_a^{h} reward.

Consider any miner i. Denote by λ_u^k the accumulated block-creation rates of miners, excluding miner i, that choose to include an unrelated transaction in $\Gamma^H(k, \mathsf{red})$. Therefore, the accumulated probabilities of miners that choose to include tx_a^{h}, excluding miner i, is $1 - \lambda_u^k - \lambda_i$.

If miner i chooses to include an unrelated transaction then either of the following occurs. First, with probability λ_i miner i gets to create a block, includes an unrelated transaction and receives a reward of f. The subsequent subgame is $\Gamma^H(k+1, \mathsf{red})$. Alternatively, with probability λ_u^k another miner that includes an unrelated transaction gets to create a block, miner i gets no reward and the subsequent subgame is $\Gamma^H(k+1, \mathsf{red})$. Finally, with probability $1-\lambda_u^k-\lambda_i$ another miner that includes tx_a^{h} gets to create a block, miner i gets no reward and the subsequent subgame is $\Gamma^H(k+1, \mathsf{irred})$.

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

$$\begin{aligned} &u_i\left(\bar{\sigma}, \Gamma^H(k, \mathsf{red})\right) = \\ &\lambda_i \cdot \left(f + u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{red})\right)\right) + \\ &\lambda_u^k \cdot u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{red})\right) + \\ &\left(1 - \lambda_i - \lambda_u^k\right) \cdot u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{irred})\right) . \end{aligned} \tag{2}$$

Similarly, if miner i chooses to include tx_a^{h} than either of the following occurs. First, with probability λ_i miner i gets to create a block, includes tx_a^{h} and receives a reward of tx_a^{h}. The subsequent subgame is $\Gamma^H(k+1, \mathsf{irred})$. Alternatively, with probability λ_u^k another miner that includes an unrelated transaction gets to create a block, miner i gets no reward and the subsequent subgame is $\Gamma^H(k+1, \mathsf{red})$. Finally, with probability $1-\lambda_u^k-\lambda_i$ another miner that includes tx_a^{h} gets to create a block, miner i gets no reward and the subsequent subgame is $\Gamma^H(k+1, \mathsf{irred})$.

Therefore, miner i's utility when including tx_a^{h} in these subgames is

$$\begin{aligned} & u_i\left(\bar{\sigma}, \Gamma^H(k, \mathsf{red})\right) = \\ & \lambda_i \cdot \left(f_a^{\text{h}} + u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{irred})\right)\right) + \\ & \lambda_u^k \cdot u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{red})\right) + \\ & \left(1-\lambda_i-\lambda_u^k\right) \cdot u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{irred})\right) \ . \end{aligned} \tag{3}$$

To prove the lemma we need to show that for any $k \in [1, T-1]$ the utility from including an unrelated transaction (Eq. 2) exceeds that of including tx_a^{h} (Eq. 3). This reduces to showing that

$$\begin{aligned} & f + u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{red})\right) > \\ & f_a^{\text{h}} + u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{irred})\right) \ , \end{aligned} \tag{4}$$

which we do inductively.

a) Base: First, consider $k = T-1$. Using Lemma 8 and Lemma 7 we get the condition presented in Eq. 4 is $f + \lambda_i f_b^{\text{h}} > f_a^{\text{h}} + \lambda_i f$, or alternatively,

$$f_b^{\text{h}} > \frac{f_a^{\text{h}} - f}{\lambda_i} + f \ . \tag{5}$$

Since $\lambda_{\text{min}} \leq \lambda_i$ and $f_b^{\text{h}} > \frac{f_a^{\text{h}} - f}{\lambda_{\text{min}}} + f$, the condition (Eq. 5) holds, meaning that in any subgame perfect equilibrium miner i is strictly better by including an unrelated transaction in subgame $\Gamma^H(T-1, \mathsf{red})$.

Therefore, all miners choose to include unrelated transactions in such subgames, meaning $\lambda_u^j = 1-\lambda_i$ and $1-\lambda_i-\lambda_u^j = 0$. Therefore, miner i's utility (Eq. 2) is $u_i\left(\bar{\sigma}, \Gamma^H(k, \mathsf{red})\right) = \lambda_i\left(f + f_b^{\text{h}}\right)$.

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 $\Gamma^H(k+1, \mathsf{red})$ is for all miners to include an unrelated transaction, and the utility of miner i when doing so is $u_i\left(\bar{\sigma}, \Gamma^H(k+1, \mathsf{red})\right) = \lambda_i\left((T-k) + f_b^{\text{h}}\right)$.

c) Step: Using the inductive assumption and Lemma 7 the condition of Eq. 4 translates to $f + \lambda_i\left((k+1)f + f_b^{\text{h}}\right) > f_a^{\text{h}} + \lambda_i(k+1)f$, or alternatively,

$$f_b^{\text{h}} > \frac{f_a^{\text{h}} - f}{\lambda_i} + f \ . \tag{6}$$

Again, since $\lambda_{\text{min}} \leq \lambda_i$ and $f_b^{\text{h}} > \frac{f_a^{\text{h}} - f}{\lambda_{\text{min}}} + f$, the condition (Eq. 6) holds, meaning that in the subgame perfect equilibrium miner i's strict best response is to include an unrelated transaction in subgame $\Gamma^H(k, \mathsf{red})$.

Since all miners include unrelated transactions, we get $\lambda_u^j = 1-\lambda_i$ and $1-\lambda_i-\lambda_u^j = 0$. Therefore, miner i's utility (Eq. 2) is $u_i\left(\bar{\sigma}, \Gamma^H(k, \mathsf{red})\right) = \lambda_i\left((T-k)f + f_b^{\text{h}}\right)$. $\square$