Rapidash: Atomic Swaps Secure under User-Miner Collusion

Hao Chung[1], Elisaweta Masserova[1], Elaine Shi[1], and Sri AravindaKrishnan Thyagarajan[2]

[1] Carnegie Mellon Unviersity
[2] University of Sydney

Abstract. Cross-chain trading is fundamental to blockchains and Decentralized Finance (DeFi). A way to achieve such trading in a truly decentralized manner, i.e., without trusted third parties, is by using *atomic swaps*. However, recent works revealed that Hashed Time-Lock Contract, a key building block of the existing atomic swaps, is entirely insecure in the presence of user-miner collusion. Specifically, a user can bribe the miners of the blockchain to help it cheat.
In this work, we give the first and rigorous formal treatment of fair trading on blockchains, where users and miners may enter arbitrary binding contracts on the side. We propose Rapidash, a new atomic swap protocol, and prove its incentive-compatibility in the presence of user-miner collusion. Specifically, we show that Rapidash satisfies a coalition-resistant Nash equilibrium absent external incentives. We give instantiations of Rapidash that are compatible with Bitcoin and Ethereum, and incur only minimal overheads in terms of costs for the users.

1 Introduction

A major challenge in blockchain technology is ensuring interoperability across multiple blockchains. Cross-chain trading, which allows users to exchange different cryptocurrencies, is a crucial step in obtaining such interoperability. While there are multiple ways to achieve such cross-chain trading, an ideal solution would allow users to trade their coins without relying on a centralized platform and without using intermediate currencies. Atomic swaps [8] achieve exactly that – they allow users to exchange assets across two blockchains without a trusted third party. The atomicity guarantee ensures that, in the end, either both users successfully exchange their assets or they retain their original assets. Atomic swaps are fundamental to many applications, driving significant efforts in the blockchain community to develop secure and efficient solutions [8, 10, 12, 13].

Such protocols typically rely on Hashed Time-Lock Contracts (HTLCs). These allow Alice to sell her secret to Bob, i.e., perform a *knowledge-coin exchange*. HTLC typically assumes that both Alice and Bob are aware of the hash derived from Alice's secret. To assure Bob that the hash truly corresponds to the correct secret, Alice can give a zero-knowledge proof, such as [14][3]. Then, Bob

[3] A similar strategy is used in, e.g., zero knowledge contingent payments [3].

deposits v coins into a smart contract. If Alice reveals the preimage of the hash, i.e., the secret, before a specified timeout T, Alice obtains v coins. Otherwise, after timeout T, Bob can request his deposit back. In practice, to ensure that only Bob learns the secret, Alice can encrypt the secret using Bob's public key and use the *encryption* of the secret as the preimage, instead of the secret itself. Current atomic swap implementations [1,18] work by composing two HTLCs in a way that lets Alice reveal her secret to get Bob's coin from the first HTLC. Bob later uses the revealed secret to get Alice's coin from the second HTLC.

Unfortunately, MAD-HTLC [20] recently showed that a single HTLC instance is already incentive-incompatible and vulnerable to very cheap bribery attacks, where a malicious Bob can bribe the miners to ignore Alice's transaction until the timeout T and get both the secret and his money back. This attack renders the atomic swap solution above insecure as well. MAD-HTLC identified bribe opportunities on the Bitcoin and Ethereum main networks where a few dollars bribe yielded tens of thousands of dollars in reward. MAD-HTLC proposed a solution that addresses the bribing attack. Unfortunately, this solution itself opens up new attacks (cf. Appendix B). Indeed, as the authors acknowledge, *MAD-HTLC does not provide any provable guarantees in the presence of general user-miner collusion.* Given MAD-HTLC's deficiency, there seems to be little hope of achieving secure atomic swaps. However, in this work, we overcome the challenges and build an atomic swap that is secure under arbitrary user-miner collusion. In particular, **our scheme is secure even if colluding users and miners enter into legally binding side-contracts (even in the physical world)**, a much more generic attack vector than the bribery attacks proposed in MAD-HTLC. Note that general forms of miner-user collusion are not merely a hypothetical problem – such collusion is prevalent in the real world, especially in the context of miner extractable value, which has become one of the most important problems in the blockchain community. Middleman platforms such as Flashbots facilitate such collusion, resulting in a billion-dollar eco-system.

1.1 Our Contributions

We formalize the problem of blockchain-based fair exchange given user-miner collusion (Sec. 2). To the best of our knowledge, we are the first to give a formal treatment in this area. Towards this, we adopt the notion of *cooperative strategy proofness* (CSP fairness) [5,16,23]. It guarantees that, absent external incentives, any coalition of players is incentivized to play honestly as long as the coalition does not control 100% of the mining power. In other words, honest behavior is a *coalition-resistant Nash equilibrium.*

To build a CSP-fair atomic swap, we first build a new *knowledge-coin exchange* protocol, RapidashKC. It achieves the same functionality as an HTLC, but can be formally proven to satisfy CSP fairness (Appendix D, also see the proof intuition in Section 3). While RapidashKC is a key building block in our atomic swap, we show that surprisingly, the naive composition of two RapidashKC instances does *not* result in a secure atomic swap scheme (Section 4). Instead,

to obtain a secure atomic swap, we carefully combine central ideas from our RAPIDASHKC in a non-black-box way with additional techniques.

We show that our solution is practical and compatible not only with the Turing complete languages such as Ethereum's Solidity [6], but also with the limited scripting language of Bitcoin (Appendix F). For the latter, we rely only on the most commonly used Bitcoin scripts and exploit Bitcoin's transaction model. Assuming generic smart contracts, our schemes are very simple to implement. In Solidity, our atomic swap requires only 252 lines of code, and we deploy the corresponding smart contracts on the Goerli testnet. We further implement and evaluate our knowledge-coin exchange RAPIDASHKC, and compare it to HTLC, MAD-HTLC, and He-HTLC [21], which aim to achieve similar functionality.

In summary, we make the following contributions:

- We formalize the knowledge-coin exchange and atomic swap problems, and propose definitions that account for user-miner collusion.
- We give an atomic swap construction that satisfies CSP-fairness. Along the way, we design a CSP secure knowledge-coin exchange protocol.
- We implement and evaluate our schemes. We give instantiations both for Bitcoin and Ethereum.

Concurrent work. The concurrent He-HTLC [21] has results that are closely related to ours. Both works were initially completed in May 2022, and have undergone several revisions since. While He-HTLC considers only knowledge-coin exchange, main technical challenges arise in the atomic swap. In particular, as we show, directly composing two knowledge-coin instances does not yield a secure atomic swap. Rapidash provides a tailored solution for this problem.

2 Formalizing Atomic Swap

2.1 Our Model

Blockchain. We assume that a blockchain is an append-only ledger consisting of a number of ordered blocks, each of which contains *transactions* possibly involving *money*. We call a subset of the players in the system who are allowed to create blockchain blocks *miners*. We assume that the network delay is 0; i.e., posted transactions are seen by everyone immediately. Thus, when miners choose the transactions to include in a block for time step t, they can see transactions posted at time t. See "On network delay" in Section 3 for a discussion on network delay. While in a practical instantiation, each party may also need to pay a small *transaction fee* for their transaction to be confirmed, for simplicity, we ignore these fees in our theoretical model since we need not rely on them to achieve our security guarantees. Adding an ϵ-small transaction fee in a practical instantiation will only introduce $O(\epsilon)$-slack to our game theoretic guarantees.

We assume that a blockchain provides a way to set up *smart contracts*, which are modeled as ideal functionalities that are 1) aware of money; and 2) whose states are publicly observable. A smart contract can have one or more *activation points*. Each transaction is associated with a unique identifier, and consists of

the following information: 1) an activation point of a smart contract, 2) a non-negative amount of money, and 3) an arbitrary message. When the transaction is executed, the corresponding activation point of the smart contract is invoked and the computation specified by this contract takes place, accompanied by the possible transfer of money. Money can be transferred from and to the following entities: smart contracts and players' pseudonyms. Without loss of generality, we may assume that players cannot directly send and receive money among themselves; however, they can send money to or receive money from smart contracts. The balance of a smart contract is the difference between the amount of money it has received and sent, and must always be non-negative.

For simplicity, we assume an idealized mining process; i.e., in each time step t, an ideal functionality picks a winning miner with probability proportional to each miner's mining power (or amount of stake for Proof-of-Stake blockchains). Whenever a miner is selected to mine a block, it can include an arbitrary subset of the outstanding transactions into the block, and order them arbitrarily. The miner can also create new transactions and include them in the mined block.

Convention for Writing Smart Contracts. We use the following style of pseudo-code to express smart contracts. ping denotes an empty message.

A toy contract

– **Parameters:** time T. Alice deposits $\$d_a$, Bob deposits $\$d_b$.

A_{fast}: On receiving ping from Alice: send $\$d_b$ to Alice.
A_{wait}: After T, on receiving ping from Alice: send $\$d_a + \$d_b$ to Alice.
B_{other}: On receiving ping from Bob: send $\$d_a$ to Bob.

The leading letter defines the *type* of the activation point. All activation points of the same type are *mutually exclusive*, i.e., if A_{wait} has been invoked, neither A_{fast} nor A_{wait} can be invoked anymore. If an activation point constrained some time interval (e.g., after block height T), then any attempted invocation outside this interval is deemed invalid and not counted. An activation point cannot be invoked if the balance is lower than the amount it is supposed to send out. For example, if A_{wait} has been invoked, B_{other} cannot be invoked anymore.

Above, Alice and Bob each deposit some coins into the contract. Once *all* deposits are in place, the contract is *active* and its activation points can be used. In practice, the contract should allow each player to withdraw its deposit if the other player has not made its deposit yet. However, once the contract is active, the distribution of money is only possible through the activation points.

System participants. In addition to the miners, we consider *users*, who can post transactions, but do not necessarily participate in block creation. All users and miners are *interactive Turing machines* who can send and receive money.

(Adversarial) strategy space. The behavior of a deviating player can be any probabilistic polynomial time (PPT) algorithm (which takes into account the existence of money). For example, at any time deviating players can post new transactions or smart contracts, deposit money into smart contracts, attempt to find hash function preimages, abort from the protocol, or send arbitrary, even

ill-formed messages to other players or smart contracts. Colluding miners about to mine a block can further, e.g., choose to censor certain transactions.

We explicitly exclude consensus- or network-level attacks — there is an orthogonal and complementary line of work that focuses on this topic [7, 15, 17].

Coalition. We consider users Alice and Bob, who wish to trade between themselves using blockchains. Either can form a coalition with some of the miners. We assume that coalition members share all information they know, e.g., when the secret seller colludes with a miner, the miner is assumed to know the secret. Signing keys are also shared inside the coalition.[4] The coalition's strategy space is the union of its members' strategy spaces. As in standard cryptographic literature, we do not consider coalitions including *both* Alice and Bob.

2.2 Game Theoretic Definitions of Blockchain-based Fair Exchange

We now formalize the properties essential for blockchain-based trading. Our notions use an application-dependent *utility* function, which we later specify explicitly for each primitive. In the following, λ is the security parameter.

CSP fairness. We first review the notion of *cooperative strategy proofness* (*CSP fairness*), formulated in [4, 5, 16, 19, 23]. Intuitively, CSP fairness is achieved if a profit-driven coalition that wants to maximize its own utility has no incentive to deviate from the honest protocol, as long as all other players play by the rules. In this sense, the honest protocol achieves a *coalition-resistant Nash Equilibrium*.

Definition 2.1 (CSP fairness). *A protocol satisfies γ-CSP-fairness, iff the following holds. Let $\mathcal{C}$ be any coalition that controls at most a $\gamma \in [0, 1)$ fraction of the mining power, and possibly includes either Alice or Bob. Then, for any probabilistic polynomial-time (PPT) strategy $S_{\mathcal{C}}$ of $\mathcal{C}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that except with $\mathsf{negl}(\lambda)$ probability, we have*

$$\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, HS_{-\mathcal{C}}) \leq \mathsf{util}^{\mathcal{C}}(HS_{\mathcal{C}}, HS_{-\mathcal{C}}), \tag{1}$$

where $HS_{\mathcal{C}}$ denotes the honest strategy of $\mathcal{C}$, $HS_{-\mathcal{C}}$ denotes the honest strategy of anyone other than $\mathcal{C}$, and $\mathsf{util}^{\mathcal{C}}(X_{\mathcal{C}}, Y_{-\mathcal{C}})$ is the expected utility of the coalition $\mathcal{C}$ when $\mathcal{C}$ is executing strategy X and the remaining players (denoted by $-\mathcal{C}$) execute strategy Y.[5]

For simplicity, we ignore the transaction fee in our model. When accounting the transaction fee $\$f$, our results can be generalized if Equation (1) is modified as $\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, HS_{-\mathcal{C}}) \leq \mathsf{util}^{\mathcal{C}}(HS_{\mathcal{C}}, HS_{-\mathcal{C}}) + O(f)$.

Dropout resilience. In blockchain-based trading, it is crucial to provide *dropout resilience*; i.e., to protect an honest player if the counterparty drops out. In practice, such a drop out can happen due to mistakes, misconfiguration, or unforeseen circumstances; e.g., Alice may lose her hardware wallet. We define it as follows:

[4] This model is standard in both in game theory (when modeling cooperative strategies), and in cryptography literature. Allowing coalition members to share information and coordinate increases the coalition's power, thus making our notions stronger.

[5] The formal definition of the utility function util is given in Section 2.3 and Section 2.4 in the context of knowledge-coin exchange and atomic swap, respectively.

Definition 2.2 (Dropout resilience). *A protocol is dropout resilient, iff as long as at least* $1/\mathsf{poly}(\lambda)$ *fraction of the mining power is honest, then with* $1 - \mathsf{negl}(\lambda)$ *probability, an honest Alice (or Bob) is guaranteed to have non-negative utility even when Bob (or Alice) is honest but drops out during the protocol's execution.*

2.3 Defining Knowledge-Coin Exchange

Imagine that Alice has a secret pre_s and Bob offers to pay Alice $\$v$ amount of coins in exchange for pre_s. We assume that the secret pre_s is worth $\$v_a$ and $\$v_b$ to Alice and Bob, respectively. That is, Alice loses utility $\$v_a$ if pre_s is released to someone else, and Bob gains $\$v_b$ if he learns pre_s. We assume that $\$v_b > \$v > \$v_a$, i.e., Alice has the incentive to sell the secret pre_s for $\$v$ coins.

For $X \in \{$CSP fairness, dropout resilience$\}$, we say that a knowledge-coin exchange satisfies X, if it satisfies X with respect to the utility function below.

Utility. Let $\beta \in \{0, 1\}$ be such that $\beta = 1$ if and only if Bob outputs pre_s at the end of the protocol. Let $\$d_a \geq 0$ and $\$d_b \geq 0$ be the amount of money Alice and Bob deposit into the smart contract, respectively. Let $\$r_a \geq 0$ and $\$r_b \geq 0$ be the payments that Alice and Bob obtain from all smart contracts during the protocol. Then, Alice's and Bob's utilities, $\$u_a$ and $\$u_b$, are defined as

$$\$u_a = -\$d_a + \$r_a - \beta \cdot \$v_a, \qquad \$u_b = -\$d_b + \$r_b + \beta \cdot \$v_b.$$

We further define the utility for the miners. Fix a miner. Let $\$d_m$ be the money that the miner deposits into the smart contracts belonging to this protocol, and let $\$r_m$ be the payment received by the miner in the current protocol instance. A miner's utility, denoted $\$u_m$, is defined as $\$u_m = -\$d_m + \$r_m$.

Finally, the joint utility of the coalition is simply the sum of every coalition member's utility. Let $\mathcal{C}$ be any subset of players, and $-\mathcal{C}$ to denote all parties of the protocol that are not in $\mathcal{C}$. Let $S_{\mathcal{C}}$ and $S'_{-\mathcal{C}}$ be the strategies of $\mathcal{C}$ and $-\mathcal{C}$. We use $\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}})$ to denote the expected joint utility of $\mathcal{C}$ when $\mathcal{C}$ adopts the strategy $S_{\mathcal{C}}$ and the remaining parties adopt the strategy $S'_{-\mathcal{C}}$.

2.4 Defining Atomic Swap

Suppose Bob has x_b coins on $\mathsf{BobChain}$ (denoted ₿x_b), and Alice has x_a coins on $\mathsf{AliceChain}$ (denoted Ⱥx_a). Bob wants to exchange his ₿x_b for Alice's Ⱥx_a.

We may assume that Alice and Bob are not in the same coalition. Therefore, we have three types of coalitions: 1) Alice-miner coalition (or Alice alone); 2) Bob-miner coalition (or Bob alone); and 3) miner-only coalition.

Given a player or coalition, we assume that it has some specific valuation of each unit of coins on $\mathsf{AliceChain}$ and $\mathsf{BobChain}$. We use the notation $\$\mathsf{AV}(\cdot)$ to denote the valuation function of Alice (or an Alice-miner coalition); specifically, $\$\mathsf{AV}(₿x_b + Ⱥx_a) = \$v_a^{\mathsf{B}} \cdot x_b + \$v_a^{\mathsf{A}} \cdot x_a$ where $\$v_a^{\mathsf{B}} \geq 0$ and $\$v_a^{\mathsf{A}} \geq 0$ denote how much Alice or the Alice-miner coalition values each coin on $\mathsf{BobChain}$ and $\mathsf{AliceChain}$, respectively. Similarly, we use $\$\mathsf{BV}(\cdot)$ to denote the valuation function of Bob (or

a Bob-miner coalition), and we use $\$\mathsf{MV}(\cdot)$ to denote the valuation function of a miner-only coalition. In the following, we make the following assumption which justifies why Alice wants to exchange her $\text{₳}x_a$ with Bob's $\text{₿}x_b$, and vice versa.

$$\textbf{Assumption: } \$\mathsf{AV}(\text{₿}x_b - \text{₳}x_a) > 0,\ \$\mathsf{BV}(\text{₳}x_a - \text{₿}x_b) > 0.$$

The assumption is necessary to prove CSP fairness as it ensures that no PPT strategy outperforms the honest strategy. However, our protocol additionally guarantees that when the honest case yields negative utility, the best utility a strategic party can achieve is zero — equivalent to not participating in the protocol. See Remark E.6 for a detailed discussion.

Finally, we define atomic swap's utility function.

Utility. Let $\mathcal{C}$ be any subset of players, and let $S_{\mathcal{C}}$ and $S'_{-\mathcal{C}}$ be the strategies of $\mathcal{C}$ and $-\mathcal{C}$. Let $\text{₳}d_a^{\mathsf{A}}, \text{₿}d_a^{\mathsf{B}} \geq 0$ be the cryptocurrencies that Alice or an Alice-miner coalition deposit into the smart contracts; let $\text{₳}r_a^{\mathsf{A}}, \text{₿}r_a^{\mathsf{B}} \geq 0$ be the payment Alice or an Alice-miner coalition receive from all smart contracts during the protocol. Now, we can define the utility $\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}})$ when $\mathcal{C}$ consists of Alice or the Alice-miner coalition as follows:

$$\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}}) = \$\mathsf{AV}(\text{₳}r_a^{\mathsf{A}} - \text{₳}d_a^{\mathsf{A}} + \text{₿}r_a^{\mathsf{B}} - \text{₿}d_a^{\mathsf{B}}).$$

We define $\text{₳}d_b^{\mathsf{A}}, \text{₿}d_b^{\mathsf{B}}, \text{₳}r_b^{\mathsf{A}}, \text{₿}r_b^{\mathsf{B}}$ analogously for Bob (or the Bob-miner coalition), and $\text{₳}r_m^{\mathsf{A}}, \text{₿}r_m^{\mathsf{B}}, \text{₳}d_m^{\mathsf{A}}, \text{₿}d_m^{\mathsf{B}}$ for the miner-only coalition. We define the utility $\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}})$ when $\mathcal{C}$ consists of Bob or a Bob-miner coalition as

$$\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}}) = \$\mathsf{BV}(\text{₳}r_b^{\mathsf{A}} - \text{₳}d_b^{\mathsf{A}} + \text{₿}r_b^{\mathsf{B}} - \text{₿}d_b^{\mathsf{B}}),$$

and the utility $\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}})$ when $\mathcal{C}$ is a miner-only coalition as

$$\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}}) = \$\mathsf{MV}(\text{₳}r_m^{\mathsf{A}} - \text{₳}d_m^{\mathsf{A}} + \text{₿}r_m^{\mathsf{B}} - \text{₿}d_m^{\mathsf{B}}).$$

3 Knowledge-Coin Exchange

As a first step toward our atomic swap, we design a knowledge-coin exchange allowing Alice to sell Bob the secret preimage pre_s of a publicly known hash h_s.

3.1 Construction

To achieve this, Bob creates a smart contract, and deposits payment $\$v$ along with a collateral $\$c_b$ into it. The contract will facilitate the exchange of pre_s for Bob's money. It is parametrized by the hash h_s and an extra hash h_b generated by Bob. To obtain h_b, Bob generates $pre_b \leftarrow \{0,1\}^\lambda$ uniformly at random and computes $h_b = H(pre_b)$. Bob holds on to the preimage, but keeps it secret. We distinguish between: **(1)** an efficient **default** path, **(2)** a **refund** path to allow Bob obtain its money back if Alice drops out, and **(3)** a ***burn*** path, which is a novel technique we introduce to punish misbehavior. We now discuss each case.

Default path. In the default case Alice simply waits until Bob deposited his money and sends pre_s to the activation point P_{default} of the smart contract in Figure 1. P_{default} then sends Bob's payment to Alice and returns the collateral to Bob. If both players are honest and there are no network delays, the protocol completes at this point. In the remaining two paths, we ensure that in case of either misbhavior or unstable network, the honest party is still protected.

Refund path. This path ensures that if Alice did not send pre_s to P_{default} on time, Bob can recover his money. To achieve this, a standard HTLC simply has an activation point which returns Bob's money upon obtaining a request from him after a deadline T. However, as MAD-HTLC showed, this is insecure [20]. Briefly, Bob can bribe the miners to ignore Alice's transaction to P_{default} (which contains pre_s), and instead include Bob's refund transaction. This way, Bob obtains both the secret and his coins (minus a small bribe) back. To prevent such attacks, we need to disincentivize Bob from attempting to get a refund once Alice's secret pre_s is publicly known. Towards this, we let Bob generate a hash h_b at the beginning of the protocol, and split the refund process into two steps: First, Bob must announce his intent to obtain a refund by sending a preimage of h_b to an activation point P_{refund} which can only be triggered after time T_1. Then, after T_2 time has passed since the activation of P_{refund}, Bob can obtain his refund by sending a message to the activation point C_{refund}. As we show below, using the helper hash h_b in combination with the timelock on Bob obtaining his refund is key to ensuring that Bob is disincentivized from misbehaving.

Burn path. The goal of the burn path is to disincentivize parties from misbehaving. Note that currently, Alice has no incentive to misbehave: She only has the choice of either revealing her secret and obtaining Bob's payment, or not revealing the secret and thus forgoing the money. Bob, however, could attempt to bribe the miners to not include Alice's transaction for $T_1 + T_2$ time, and including his own refund transactions instead. Thus, we must ensure that miners have a "better choice". For this, we introduce the *bomb* – an activation point C_{burn}, which, given preimages of both Alice's h_s and Bob's h_b, sends a small amount of Bob's coins to the party who submitted these preimages, and burns the rest. Note that if Bob attempts to misbehave after Alice's secret is publicly known, as we split the refund path into two parts, both Alice's and Bob's preimages are known after Bob invoked P_{refund}. Thus, miners have at least T_2 time to submit both pre_s and pre_b to C_{burn} and obtain the reward. Thus, Bob would need to corrupt every miner who mines a block during this window to ensure that that miner chooses to *not* activate C_{burn}. In the following, we will describe how to set the parameters T_2, c_b, and the amount of the reward obtained in C_{burn} to ensure that it is irrational for Bob to attempt the attack. Similarly, by setting the parameters in this way we can provably ensure that a malicious Alice is disincentivised from attempting to activate C_{burn} instead of the default P_{default}.

RapidashKC contract. We give our formal knowledge-coin exchange smart contract below. Activation points of the same type are mutually exclusive.

RapidashKC protocol. Informally, we have Bob deposit $\$v + \$c_b$ into RapidashKC (let $t = 0$ denote the corresponding time), and have Alice post pre_s as

Fig. 1: RAPIDASHKC contract.

/* *Params:* $(h_s, h_b, T_1, T_2, \$v, \$c_b, \$\epsilon)$*, Bob deposits* $\$v + \$c_b$. */

P_{default}: On receiving z from Alice s.t. $H(z) = h_s$, send $\$v$ to Alice and $\$c_b$ to Bob.

P_{refund}: Time T_1 or greater: on receiving z from Bob s.t. $H(z) = h_b$, do nothing.

C_{refund}: At least T_2 after P_{refund} is activated: on receiving ping from anyone, send $\$v + \$c_b$ to Bob.

C_{burn}: On receiving (z_1, z_2) from any P s.t. $H(z_1) = h_s$ and $H(z_2) = h_b$, send $\$\epsilon$ to player P. All remaining coins are burnt.

soon as Bob has done so. If Alice has not posted a valid preimage by deadline T_1, Bob submits the refund request to P_{refund} (and revealing his secret pre_b). T_2 time after submitting his request, Bob can obtain his refund by sending ping to C_{refund}. Further, if anyone knows both pre_s and pre_b, they can send those to C_{burn} to obtain a small reward ϵ, and *burn all remaining coins.*

We now give the formal RAPIDASHKC protocol, i.e., the formal description of the sequence of actions that an honest Alice, Bob, and miner must follow. Note that when we give the description for Alice, we do *not* assume that Bob and the miners follow the protocol. The same holds for Bob.

RAPIDASHKC protocol

Alice: Alice sends pre_s to P_{default} at $t = 0$.

Bob: If Alice failed to send pre_s to P_{default} before T_1, Bob sends pre_b to P_{refund} at time $t = T_1$. Then, T_2 time after P_{refund} is activated, he sends ping to C_{refund}. If either P_{default} or C_{burn} is successfully activated, Bob outputs the corresponding pre_s value included in the corresponding transaction. Otherwise, Bob outputs $\perp$.

Miner: Every miner M watches all transactions posted to P_{default}, P_{refund}, and C_{burn}. If M observes the correct values of both pre_s and pre_b in these transactions, it sends (pre_s, pre_b) to C_{burn}. Further, M always includes all outstanding transactions in every block it mines. If multiple transactions are posted to C_{burn}, M places its own ahead of others (thus invalidating the others).

Theorem 3.1 (CSP fairness and dropout resilience). *Suppose that the hash function $H(\cdot)$ is a one-way function and that all players are PPT machines. Moreover, suppose that $\$c_b < \$\epsilon$, $\$\epsilon < \$v$, and $\gamma^{T_2} \leq \frac{\$c_b}{\$c_b + \$v}$. Then, the* RAPIDASHKC *protocol satisfies γ-CSP-fairness and dropout resilience.*

The formal proof of Theorem 3.1 is given in Appendix D. Here, we outline the intuition behind the parameter constraints. Briefly, burning a large part of Bob's collateral in C_{burn} disincentivizes Bob from attempting to get both the secret and the refund. To formally achieve security against general user-miner collusion, we set the parameters with respect to the following constraints.

- $\$c_b < \$\epsilon$, and $\$\epsilon < \$v$: the former ensures that a sufficient amount is burnt should the bomb C_{burn} be triggered, and thus activating $P_{\mathsf{refund}} + C_{\mathsf{burn}}$ does not make sense for Bob; the latter ensures that Alice prefers to activate P_{default} rather than C_{burn}.

– $\$\gamma^{T_2} \leq \frac{\$c_b}{\$c_b+\$v}$ where γ is an upper bound on the fraction of mining power controlled by the coalition: If the honest Alice posts pre_s to P_{default}, this condition ensures that it is not worth it for the Bob-miner coalition to gamble and attempt to invoke both P_{refund} and C_{refund} to get Bob's deposit back. As in this case after invoking P_{refund} both pre_s and pre_b are publicly known, the coalition must mine *all* blocks within the next T_2 window to guarantee that C_{refund} is invoked. Otherwise, any non-colluding miner who mines a block during this window will trigger the bomb C_{burn}.

On network delay. For simplicity, we assume that the network delay δ is zero, and honest miners always include honest players' transactions in the next block. In RapidashKC, T_1 is to ensure that Bob does not try to activate the refund path too early given Alice's transaction is delayed; and T_2 is to ensure that at least one non-colluding miner proposes a block among T_2 blocks with high probability. In practice, if the delay $\delta > 0$, we can choose the parameters such that T_1 is larger than δ plus the time required for Alice's transaction to be included in a block, and $\$\gamma^{T_2-\delta} \leq \frac{\$c_b}{\$c_b+\$v}$ to account for the delay.

On burning coins. Burning money is adopted by major cryptocurrencies to incentivize honest behavior. For example, Ethereum's EIP1559 transaction fee mechanism burns all the base fees. While we use burning as a crucial component in our construction, we emphasize that the burning logic is only triggered if either Alice or Bob misbehaves. Our construction incentivizes players to behave honestly, so the burning logic should not be invoked in the equilibrium state.

Concrete parameter examples. Suppose we choose $\$c_b = \$v$. Then, we need to make sure $\gamma^{T_2} \leq \frac{1}{2}$. This means that if $\gamma = 90\%$, we can set $T_2 = 7$; if $\gamma = 49.9\%$, we can set $T_2 = 1$. Asymptotically, for any $\gamma = O(1)$, T_2 is a constant. Increasing $\$c_b$ helps to make T_2 smaller. For CSP fairness to hold, $\$\epsilon$ can be arbitrarily small. However, as we discuss later when analyzing the coalition-forming meta-game (see Appendix B), we may want $\$\epsilon$ to be not too small, such that 100% coalition is not an equilibrium in the coalition-forming meta-game. In practice, we can set $\$\epsilon$ to be slightly smaller than $\$v$.

Comparison to He-HTLC and MAD-HTLC. The knowledge-coin exchange of the concurrent work He-HTLC is conceptually similar to ours. The difference is that in He-HTLC's path which is equivalent to our C_{burn}, player P obtains $\$c_b$ (instead of our $\$\epsilon$). Same as ours, their solution allows to fine-tune the collateral, i.e., there is a trade-off between the collateral size and the time that this collateral is locked for. For the example above, with $\$c_b = \$v$ and $\gamma = 90\%$, assuming the transaction fees are zero, we estimate the He-HTLC's equivalent of T_2 to be 11 (for us it was 7). For the example with $\gamma = 49.9\%$, we estimate their T_2 to be 2 (vs. 1 for us). Finally, we note that there is a bug in the Bitcoin evaluation of He-HTLC. For completeness, we give a short description in Appendix G.

For MAD-HTLC, the collateral can be any non-zero amount (again assuming that transaction fees are zero). However, MAD-HTLC's security guarantees do not match those of He-HTLC and ours. MAD-HTLC defends only against a very specific bribery attack, and as admitted by the MAD-HTLC authors (Sec.

8 of [20]), it does not defend against general user-miner collusion where users and miners can enter into arbitrary binding contracts.

4 Our Atomic Swap Construction

Naive Composition. Say Alice holds $\text{Å}x_a$ coins on AliceChain, and wishes to trade them for Bob's $\text{Ƀ}x_b$ from BobChain. Consider naively composing two knowledge-coin exchange instances: First, Alice *generates* a preimage pre_s (in contrast to knowledge-coin exchange, there is no secret knowledge to be sold) uniformly at random, and publishes its hash h_s. Then, Alice deposits the prescribed amount of money into RAPIDASHKC's contract on AliceChain. Essentially, on this chain Alice acts as the secret buyer in our knowledge-coin exchange protocol. On BobChain, Bob is one who makes the deposit. On both chains, the default path P_{default} is locked via h_s. The idea is that in order to obtain Bob's money, Alice has to publish her preimage pre_s on BobChain. In doing so, Alice inadvertedly reveals pre_s to Bob too, who can use it to get Alice's coins from AliceChain.

In more detail, we run one instance of RAPIDASHKC on AliceChain, and refer to its activation points as $P^{\mathsf{A}}_{\mathsf{default}}$, $P^{\mathsf{A}}_{\mathsf{refund}}$, $C'^{\mathsf{A}}_{\mathsf{refund}}$, $C'^{\mathsf{A}}_{\mathsf{burn}}$. We run another instance on BobChain, and refer to its activation points as $P^{\mathsf{B}}_{\mathsf{default}}$, $P^{\mathsf{B}}_{\mathsf{refund}}$, $C'^{\mathsf{B}}_{\mathsf{refund}}$, $C'^{\mathsf{B}}_{\mathsf{burn}}$. Alice deposits the payment $\text{Å}x_a$ and the collateral $\text{Å}c^{\mathsf{A}}_a$ into RAPIDASHKC on AliceChain. Similarly, Bob deposits $\text{Ƀ}x_b + \text{Ƀ}c^{\mathsf{B}}_b$ into the contract on BobChain.

Then, Alice generates $pre_s, pre_a \leftarrow \{0,1\}^\lambda$ uniformly at random, and Bob generates $pre_b \leftarrow \{0,1\}^\lambda$ uniformly at random. Here, pre_s is to facilitate the default path of the coin swap, and pre_a, pre_b are for the refund. As before, both parties reveal the corresponding hashes h_s, h_a, h_b. We use h_s to lock *both* $P^{\mathsf{A}}_{\mathsf{default}}$ and $P^{\mathsf{B}}_{\mathsf{default}}$, with the difference that $P^{\mathsf{B}}_{\mathsf{default}}$ can be unlocked by *Alice* sending a correct preimage, and $P^{\mathsf{A}}_{\mathsf{default}}$ can be unlocked by *Bob* sending a correct preimage. Intuitively, as Alice needs to send pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$ to obtain her payment from Bob, once she has done so, everyone (in particular, Bob) will know pre_s too. Bob can then send it to $P^{\mathsf{A}}_{\mathsf{default}}$ on AliceChain to obtain his payment from Alice.

If Alice drops out, Bob posts pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$ for a refund. If Bob drops out, Alice asks for a refund by posting pre_a to $P^{\mathsf{A}}_{\mathsf{refund}}$. One can hope that the intuition from the knowledge-coin exchange works here as well: Once Alice has posted pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$, a Bob-miner coalition is disincentivized from posting pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$ due to the fear of triggering the bomb (similar for Bob and Alice-miner coalition).

Vulnerability in the naive composition. Unfortunately, this intuition does not hold. The issue is that an Alice-miner coalition can wait for Bob to make the deposit, and instead of posting pre_s to BobChain, *first* get refunded on AliceChain. Of course, in response Bob will try to get his refund on BobChain. However, Alice-miner coalition can attempt to defer Bob's refund transaction, and *now* attempt to invoke $P^{\mathsf{B}}_{\mathsf{default}}$ by revealing pre_s. At this point, pre_s by itself is worth nothing to Bob, as pre_s in this construction is simply the means to obtain the money on each chain, and the contract on AliceChain has been emptied out already. Thus, if successful, the Alice-miner coalition gets Bob's $\text{Ƀ}x_b$ for free!

Alice can launch such attack by posting the following contract at the beginning: Alice will pay $\$r > \$\epsilon$ to the miner who invokes $P^{\mathsf{B}}_{\mathsf{default}}$ by using pre_s. For any miner with γ fraction of the mining power, the probability of being chosen as the block producer to invoke $P^{\mathsf{B}}_{\mathsf{default}}$ is γ. Thus, the expected utility of joining Alice's coalition, deferring Bob's refund transaction, and trying to invoke $P^{\mathsf{B}}_{\mathsf{default}}$ is $\gamma \cdot \$r$. Let $\$f$ be the maximum transaction fee a miner can get in expectation if it selects Bob's transaction. As long as $\gamma \cdot \$r > \$f$, the miner with at least γ fraction of the mining power is incentivized to join Alice's coalition.[6]

Second Attempt. To fix this, we must disincentivize Alice from refusing to post pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$ at the right time, but attempting to later invoke $P^{\mathsf{A}}_{\mathsf{refund}}$. To achieve this, we utilize the fact that if Alice fails to post pre_s, an honest Bob posts pre_b, and we allow the bomb $C^{\mathsf{A}}_{\mathsf{burn}}$ to be triggered with the pair (pre_a, pre_b).

Unfortunately, now we cannot guarantee dropout resilience for Alice: If Alice's deposit transaction takes too long to confirm, Bob will attempt to get refunded by posting pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$. Suppose Bob drops out at this point. In this case, whenever Alice's deposit transaction is finalized, Alice cannot get her own deposit back since if she posts pre_a to $P^{\mathsf{A}}_{\mathsf{refund}}$, it will trigger the bomb $C^{\mathsf{A}}_{\mathsf{burn}}$.

Intuitively, the key challenge is finding the right balance for how easy it is for a user to withdraw its deposit. If it is too easy, then it becomes risk-free to attack the other user. If it is too difficult, the protocol may not satisfy dropout resilience anymore. Next, we explain how we resolve the tension by introducing another hash to lock the deposits for both users.

4.1 Achieving CSP-Fairness

To address the issues above, we introduce a "two-phase preparation" stage. Initially, $P^{\mathsf{B}}_{\mathsf{default}}$ and $C^{\mathsf{B}}_{\mathsf{burn}}$ are locked with a hash h_c of a value $pre_c \leftarrow \{0,1\}^\lambda$ generated by Bob. Bob publishes pre_c if the deposits into both contracts take effect in a timely manner. Once pre_c is published, Alice must post pre_s immediately. Now, we can distinguish between the case where the deposit transactions take too long and the case where Alice is malicious, and let Bob act accordingly:

- If the deposit transactions take too long to confirm, before posting pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$, Bob will post ping to $P^{\mathsf{A}}_{\mathsf{refund}}$ (see contract below). The ping from Bob acts as an alternative way to invoke $P^{\mathsf{A}}_{\mathsf{refund}}$ on the path of Alice getting her deposit back. This resolves our prior dropout resilience issue where Alice could not get her deposit back once Bob has posted pre_b, as now Alice does not need to send pre_a to $P^{\mathsf{A}}_{\mathsf{refund}}$ anymore. Note that it is safe for Bob to help Alice get refunded before getting refunded himself *because he has not released* pre_c *yet*, and thus no one else can cash out his coins in Rapidash.
- If Bob has already opened the lock with pre_c, then, should the honest Bob ever post pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$, it must be due to Alice's failure to post pre_a to $P^{\mathsf{A}}_{\mathsf{default}}$, meaning that Alice is acting dishonestly. In this case, Bob does not help Alice get her deposit back.

[6] This attack is just an example. How to censor a user's transaction in the context of HTLC is described in [20, 22].

We now present the formal smart contracts and protocol for our atomic swap. All times are expressed in the time of the respective chain. As before, activation points of the same type are **mutually exclusive**. Moreover, the activation points can be triggered only if the contract is active, i.e. both parties have deposited.

RAPIDASH$^{\mathbf{B}}$

/* *Params:* $(h_s,\ h_b,\ h_c,\ T_1^{\mathsf{B}},\ \tau^{\mathsf{B}},\ \ddot{\mathrm{B}}x_b,\ \ddot{\mathrm{B}}c_b^{\mathsf{B}}, \ddot{\mathrm{B}}\epsilon^{\mathsf{B}})$, *Bob deposits* $\ddot{\mathrm{B}}x_b + \ddot{\mathrm{B}}c_b^{\mathsf{B}}$. */

$P_{\mathsf{default}}^{\mathsf{B}}$: On receiving z_1 from Alice and z_2 from Bob such that $H(z_1) = h_s$ and $H(z_2) = h_c$, send $\ddot{\mathrm{B}}x_b$ to Alice and $\ddot{\mathrm{B}}c_b^{\mathsf{B}}$ to Bob.

$P_{\mathsf{refund}}^{\mathsf{B}}$: Time T_1^{B} or greater: On receiving z from Bob such that $H(z) = h_b$ or on receiving ping from Alice, do nothing.

$C_{\mathsf{refund}}^{\mathsf{B}}$: At least τ^{B} after $P_{\mathsf{refund}}^{\mathsf{B}}$ is activated: on receiving ping from anyone, send $\ddot{\mathrm{B}}x_b + \ddot{\mathrm{B}}c_b^{\mathsf{B}}$ to Bob.

$C_{\mathsf{burn}}^{\mathsf{B}}$: On receiving (z_1, z_2, z_3) from anyone P such that $H(z_1) = h_s$, $H(z_2) = h_b$, and $H(z_3) = h_c$ send $\ddot{\mathrm{B}}\epsilon^{\mathsf{B}}$ to player P. All remaining coins are burnt.

RAPIDASH$^{\mathbf{A}}$

/* *Params:* $(h_s, h_a, T_1^{\mathsf{A}}, \tau^{\mathsf{A}}, \ddot{\mathrm{A}}x_a, \ddot{\mathrm{A}}c_a^{\mathsf{A}}, \ddot{\mathrm{A}}\epsilon^{\mathsf{A}})$, *Alice deposits* $\ddot{\mathrm{A}}x_a + \ddot{\mathrm{A}}c_a^{\mathsf{A}}$, *Bob deposits* $\ddot{\mathrm{A}}c_b^{\mathsf{A}}$ */

$P_{\mathsf{default}}^{\mathsf{A}}$: On receiving z from Bob such that $H(z) = h_s$ or on receiving ping from Alice, send $\ddot{\mathrm{A}}x_a + \ddot{\mathrm{A}}c_b^{\mathsf{A}}$ to Bob and $\ddot{\mathrm{A}}c_a^{\mathsf{A}}$ to Alice.

$P_{\mathsf{refund}}^{\mathsf{A}}$: Time T_1^{A} or greater: on receiving z from Alice such that $H(z) = h_a$ or on receiving ping from Bob, do nothing.

$C_{\mathsf{refund}}^{\mathsf{A}}$: At least τ^{A} after $P_{\mathsf{refund}}^{\mathsf{A}}$ is activated: on receiving ping from anyone, send $\ddot{\mathrm{A}}x_a + \ddot{\mathrm{A}}c_a^{\mathsf{A}}$ to Alice and $\ddot{\mathrm{A}}c_b^{\mathsf{A}}$ to Bob.

$C_{\mathsf{burn}}^{\mathsf{A}}$: On receiving either (z_1, z_2) where $H(z_1) = h_s$ and $H(z_2) = h_a$, or (z_2, z_3) such that $H(z_2) = h_a$ and $H(z_3) = h_b$ from any P, send $\ddot{\mathrm{A}}\epsilon^{\mathsf{A}}$ to P. All remaining coins are burnt.

The parameters above must respect the following parameter constraints.

- **Constraints for RAPIDASH$^{\mathbf{B}}$ (on BobChain):**
  - $h_s = H(pre_s)$, $h_b = H(pre_b)$ and $h_c = H(pre_c)$.
  - $T_1^{\mathsf{B}} > T_0^{\mathsf{B}} > T^{\mathsf{B}} > 0$, where T_0^{B} and T^{B} will be introduced later.
  - $\ddot{\mathrm{B}}x_b > \ddot{\mathrm{B}}\epsilon^{\mathsf{B}} > \ddot{\mathrm{B}}0$, and $\ddot{\mathrm{B}}c_b^{\mathsf{B}} > \ddot{\mathrm{B}}\epsilon^{\mathsf{B}}$
- **Constraints for RAPIDASH$^{\mathbf{A}}$ (on AliceChain):**
  - $h_s = H(pre_s)$ and $h_a = H(pre_a)$.
  - AliceChain time T_1^{A} > BobChain time T_1^{B}, i.e., AliceChain block of length T_1^{A} is mined after the BobChain block of length T_1^{B}.[7]
  - $\ddot{\mathrm{A}}\epsilon^{\mathsf{A}} > \ddot{\mathrm{A}}0$, $\ddot{\mathrm{A}}c_a^{\mathsf{A}} > \ddot{\mathrm{A}}\epsilon^{\mathsf{A}}$ and $\ddot{\mathrm{A}}c_b^{\mathsf{A}} > \ddot{\mathrm{A}}\epsilon^{\mathsf{A}}$.
- **Choice of timeouts:**
  - $\tau^{\mathsf{B}} \geq 1$, $\tau^{\mathsf{A}} \geq 1$.
  - $\gamma^{\tau^{\mathsf{A}}} \leq \frac{\ddot{\mathrm{A}}c_a^{\mathsf{A}}}{\ddot{\mathrm{A}}c_a^{\mathsf{A}} + \ddot{\mathrm{A}}x_a}$, $\gamma^{\tau^{\mathsf{B}}} \leq \frac{\ddot{\mathrm{B}}c_b^{\mathsf{B}}}{\ddot{\mathrm{B}}c_b^{\mathsf{B}} + \ddot{\mathrm{B}}x_b}$

[7] In practice, this constraint should be respected except with negligible probability despite the variance in inter-block times.

We provide the protocol i.e., description of the behavior for the honest parties. The moment that both contracts have been posted and take effect is defined to be the start of the execution (i.e. $t = 0$). Let BobChain time 0 and AliceChain time 0 be the length of BobChain and AliceChain when the execution starts, respectively. Note that whenever parties are required to "Wait", they wait until the specified event happens, and then execute the corresponding action. When they start waiting, they also verify whether (one of) the specified events took place *already*, and execute the corresponding action if this is the case.

Atomic Swap Protocol — Alice

Preparation Phase:

1. At $t = 0$, Alice sends the deposit transaction of $\ddot{A}x_a + \ddot{A}c_a^{\mathsf{A}}$ to $\textsc{Rapidash}^{\mathsf{A}}$;
2. Wait until one of the following happens:
   - Either $\textsc{Rapidash}^{\mathsf{B}}$ or $\textsc{Rapidash}^{\mathsf{A}}$ has not been active, and it is at least BobChain time T^{B}: Alice enters the abort phase.
   - Bob has not sent pre_c to $P^{\mathsf{B}}_{\mathsf{default}}$, and it is at least BobChain time T_0^{B}: Alice enters the abort phase.
   - Bob sent pre_c to $P^{\mathsf{B}}_{\mathsf{default}}$ and it is before BobChain time T_0^{B}: Alice enters the execution phase.

Execution Phase:

1. Alice sends pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$. As soon as $P^{\mathsf{B}}_{\mathsf{default}}$ has been activated, Alice sends ping to $P^{\mathsf{A}}_{\mathsf{default}}$.
2. If τ^{B} BobChain *time* has passed since $P^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Alice sends ping to $C^{\mathsf{B}}_{\mathsf{refund}}$. (Note that as soon as $C^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Bob sends ping to $P^{\mathsf{A}}_{\mathsf{refund}}$.)
3. If τ^{A} AliceChain *time* has passed since activating $P^{\mathsf{A}}_{\mathsf{refund}}$, Alice sends ping to $C^{\mathsf{A}}_{\mathsf{refund}}$.

Abort Phase:

1. At BobChain time T_0^{B}, Alice sends ping to $P^{\mathsf{B}}_{\mathsf{refund}}$.
2. Wait until BobChain time T_1^{B}. If Bob has not sent ping to $P^{\mathsf{A}}_{\mathsf{refund}}$, Alice sends pre_a to $P^{\mathsf{A}}_{\mathsf{refund}}$.
3. If τ^{A} AliceChain *time* has passed since $P^{\mathsf{A}}_{\mathsf{refund}}$ is activated, Alice sends ping to $C^{\mathsf{A}}_{\mathsf{refund}}$; similarly, if τ^{B} BobChain *time* has passed since $P^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Alice sends ping to $C^{\mathsf{B}}_{\mathsf{refund}}$.

Ignore all other events.

The protocol for Bob is defined similarly, and is given in Appendix C. In the abort phase, we require that the honest Alice and honest Bob to send ping to $P^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{refund}}$, respectively, at BobChain time T_0^{B} even though they would not be triggered until BobChain time T_1^{B} and AliceChain time T_1^{A}, respectively. This gap allows the honest Alice to decide whether she should send pre_a to $P^{\mathsf{A}}_{\mathsf{refund}}$ or not depending on Bob's behavior.

Observe that when Alice and Bob are both honest, Alice will post pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$ immediately, thus enabling Bob to learn pre_s and post it to $P^{\mathsf{A}}_{\mathsf{default}}$ immediately after. Therefore, both players get their desired cryptocurrency and all their collateral back as soon as new block is confirmed on both chains.

Finally, we show that CSP-fairness and dropout resilience are satisfied.

Theorem 4.1 (CSP fairness and dropout resilience). *Suppose that the hash function $H(\cdot)$ is a one-way function and that all players are PPT machines.*

For any $\gamma \in [0,1]$, if the parameters satisfy the constraints, then, the atomic swap protocol satisfies γ-CSP-fairness. Furthermore, the atomic swap protocol is also dropout resilient.

Intuition for achieving CSP-fairness. Intuitively, the constraint $\ddot{\mathrm{B}}\epsilon^{\mathsf{B}} < \ddot{\mathrm{B}}x_b$ ensures that Alice, who does not have collateral in RAPIDASH$^{\mathrm{A}}$, always prefers $P^{\mathsf{B}}_{\mathsf{default}}$ to the bomb $C^{\mathsf{B}}_{\mathsf{burn}}$. The constraint $\ddot{\mathrm{B}}c_b^{\mathsf{B}} > \ddot{\mathrm{B}}\epsilon^{\mathsf{B}}$ ensures that if Bob gets Alice's $\ddot{\mathrm{A}}x_a$ and triggers the bomb $C^{\mathsf{B}}_{\mathsf{burn}}$, he still loses to the honest case, and the constraint $\ddot{\mathrm{A}}c_a^{\mathsf{A}} > \ddot{\mathrm{A}}\epsilon^{\mathsf{A}}$ serves a similar purpose. The condition $\ddot{\mathrm{A}}c_b^{\mathsf{A}} > \ddot{\mathrm{A}}\epsilon^{\mathsf{A}}$ makes sure that Bob does not want to trigger the bomb $C^{\mathsf{A}}_{\mathsf{burn}}$ even when he can get all of his deposit into RAPIDASH$^{\mathrm{A}}$ refunded. Finally, the constraint $\gamma^{\tau^{\mathsf{B}}} < \frac{\ddot{\mathrm{B}}c_b^{\mathsf{B}}}{\ddot{\mathrm{B}}c_b^{\mathsf{B}}+\ddot{\mathrm{B}}x_b}$ ensures that the window between $P^{\mathsf{B}}_{\mathsf{refund}}$ and $C^{\mathsf{B}}_{\mathsf{refund}}$ is sufficiently long such that once the honest Alice has posted pre_s, it is not worth it for Bob to take a gamble to trigger $P^{\mathsf{B}}_{\mathsf{refund}}$ and $C^{\mathsf{B}}_{\mathsf{refund}}$. In particular, if during the τ^{B} window, any honest miner mines a block, then the bomb $C^{\mathsf{B}}_{\mathsf{burn}}$ will be triggered and Bob will lose his collateral. The condition $\gamma^{\tau^{\mathsf{A}}} < \frac{\ddot{\mathrm{A}}c_a^{\mathsf{A}}}{\ddot{\mathrm{A}}c_a^{\mathsf{A}}+\ddot{\mathrm{A}}x_a}$ serves a similar purpose, but now for Alice and RAPIDASH$^{\mathrm{A}}$. The formal proofs are given in Appendix E.

Concrete parameter examples. Suppose we choose $\ddot{\mathrm{B}}c_b^{\mathsf{B}} = \ddot{\mathrm{B}}x_b$. Then, we should ensure $\gamma^{\tau^{\mathsf{B}}} \leq 1/2$. This means that if $\gamma = 90\%$, we can set $\tau^{\mathsf{B}} = 7$; if $\gamma = 49.9\%$, we can set $\tau^{\mathsf{B}} = 1$. Asymptotically, for $\gamma = O(1)$, τ^{B} is a constant. Increasing $\ddot{\mathrm{B}}c_b^{\mathsf{B}}$ makes τ^{B} smaller. A similar calculation works for τ^{A} and $\ddot{\mathrm{A}}c_a^{\mathsf{A}}$.

4.2 Instantiation and Evaluation

Bitcoin. We now summarize the implementation of RAPIDASH$^{\mathrm{B}}$ in Bitcoin (see Appendix F for full version). First, Bob prepares a setup transaction $tx^{\mathrm{B}}_{\mathsf{stp}}$ which deposits his coins into the address $Adr^{\mathrm{B}}_{\mathsf{stp}}$. The script associated with $Adr^{\mathrm{B}}_{\mathsf{stp}}$ specifies activation points $P^{\mathsf{B}}_{\mathsf{default}}$, $P^{\mathsf{B}}_{\mathsf{refund}}$, and $C^{\mathsf{B}}_{\mathsf{burn}}$ for redeeming the coins. Alice and Bob pre-sign all redeeming transactions upfront, ensuring they can later withdraw from $Adr^{\mathrm{B}}_{\mathsf{stp}}$. To achieve the burning of coins in $C^{\mathsf{B}}_{\mathsf{burn}}$, transaction $tx_{C^{\mathsf{B}}_{\mathsf{burn}}}$ is set to redeem exactly $x_b + c_b^{\mathsf{B}} - \epsilon^{\mathsf{B}}$ coins to an irredeemable address, leaving behind ϵ^{B} as a reward. This transaction is broadcast into the network, and can later be published on the blockchain by any miner provided they reveal pre_s, pre_a, and pre_c. The implementation of RAPIDASH$^{\mathrm{A}}$ is similar to RAPIDASH$^{\mathrm{B}}$, except Alice deposits $x_a + c_a^{\mathsf{A}}$ and Bob deposits c_b^{A} using $tx^{\mathrm{A}}_{\mathsf{stp}}$. See Table 1 for the estimates of the transaction sizes.

Ethereum. We implemented our contracts in Solidity, Ethereum's smart contract language and deployed these on Goerli testnet. In Ethereum, the price of a transaction depends on its *gas* usage, which describes the cost of each operation performed by the smart contract.

Comparison of Knowledge-Coin Exchange. We compare gas cost of of RAPIDASHKC with those of MAD-HTLC and He-HTLC in Table 3. The cost of RAPIDASHKC is very similar to the concurrent He-HTLC. The total redeem

Table 1: Estimates of Bitcoin transaction sizes for CSP-fair atomic swap.

Contract	Activation branch	Size (vBytes)	Fees (BTC)
RapidashB	$P^{\mathsf{B}}_{\mathsf{default}}$	455	0.0025
	$P^{\mathsf{B}}_{\mathsf{refund}}$ (ping from Alice)	440	0.0022
	$P^{\mathsf{B}}_{\mathsf{refund}}$ (Call by Bob)	448	0.0025
RapidashA	$P^{\mathsf{A}}_{\mathsf{default}}$ (Call by Bob)	479	0.0027
	$P^{\mathsf{A}}_{\mathsf{default}}$ (ping from Alice)	471	0.0026
	$P^{\mathsf{A}}_{\mathsf{refund}}$ (Call by Alice)	437	0.0024
	$P^{\mathsf{A}}_{\mathsf{refund}}$ (ping from Bob)	429	0.0024

cost in the optimistic case in RapidashKC is lower than MAD-HTLC's, as the latter has Alice obtain the deposit and Bob retrieve the collateral separately.

Evaluation of Atomic Swap. Our Ethereum atomic swap implementation consists of two contracts, one for RapidashB, and one for RapidashA. Table 2 details gas costs of all redeem paths. The deployment gas costs of RapidashB and RapidashA are 1,097,177 and 1,514,861 units, respectively.

Table 2: CSP-fair atomic swap, gas cost. (O) denotes an optimistic case.

Contract	Redeem path	Gas
RapidashB	Normal path ($P^{\mathsf{B}}_{\mathsf{default}}$), Alice	52,279
	Normal path ($P^{\mathsf{B}}_{\mathsf{default}}$), Bob	56,681
	Refund path ($P^{\mathsf{B}}_{\mathsf{refund}} + C^{\mathsf{B}}_{\mathsf{refund}}$), Bob	123,631
	Burn path ($C^{\mathsf{B}}_{\mathsf{burn}}$), Miner	42,266
RapidashA	Input, Alice	50,465
	Input, Bob	55,817
	Withdraw, Alice	38,228
	Withdraw, Bob	35,911
	(O) ($P^{\mathsf{A}}_{\mathsf{default}}$), Alice	54,904
	(O) ($P^{\mathsf{A}}_{\mathsf{default}}$), Bob	58,656
	Refund ($P^{\mathsf{A}}_{\mathsf{refund}} + C^{\mathsf{A}}_{\mathsf{refund}}$), Alice	118,379
	Refund ($P^{\mathsf{A}}_{\mathsf{refund}} + C^{\mathsf{A}}_{\mathsf{refund}}$), Bob	114,647
	Burn ($C^{\mathsf{A}}_{\mathsf{burn}}$), Miner	53,431

5 Conclusion and Future Work

In this work, we formalized key notions for blockchain-based fair trading and presented protocols that satisfy these notions. We leave several interesting questions for future work: Is it possible to have an atomic swap secure against user-miner collusion which requires each user to deposit collateral on at most one chain? Can we have fair exchange among more than two parties? And many more.

Acknowledgements This work is in part supported by NSF awards 2212746, 2044679, 1704788, a Packard Fellowship, a generous gift from the late Nikolai Mushegian, a gift from Google, and an ACE center grant from Algorand Foundation.

References

1. Lightning network (2023), https://lightning.network/
2. Bonneau, J.: Why buy when you can rent? In: FC (2016)
3. Campanelli, M., Gennaro, R., Goldfeder, S., Nizzardo, L.: Zero-knowledge contingent payments revisited: Attacks and payments for services. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 229–243 (2017)
4. Chung, H., Shi, E.: Foundations of transaction fee mechanism design. In: SODA (2023)
5. Chung, K., Guo, Y., Lin, W., Pass, R., Shi, E.: Game theoretic notions of fairness in multi-party coin toss. In: TCC. vol. 11239, pp. 563–596 (2018)
6. Ethereum: The Solidity contract-oriented programming language (2022), URL: https://github.com/ethereum/solidity
7. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Eurocrypt (2015)
8. Herlihy, M.: Atomic cross-chain swaps. In: PODC (2018)
9. Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gazi, P., Meiklejohn, S., Weippl, E.: Pay to win: Cheap, crowdfundable, cross-chain algorithmic incentive manipulation attacks on pow cryptocurrencies. In: FC WTSC (2021)
10. Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: NDSS 2019
11. McCorry, P., Hicks, A., Meiklejohn, S.: Smart contracts for bribing miners. In: FC Workshops (2018)
12. van der Meyden, R.: On the specification and verification of atomic swap smart contracts. In: IEEE ICBC (2019)
13. Miraz, M.H., Donald, D.C.: Atomic cross-chain swaps: Development, trajectory and potential of non-monetary digital token swap facilities. In: AETiC (2019)
14. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy. pp. 238–252. IEEE Computer Society (2013)
15. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Eurocrypt (2017)
16. Pass, R., Shi, E.: Fruitchains: A fair blockchain. In: PODC (2017)
17. Pass, R., Shi, E.: Rethinking large-scale consensus. In: CSF (2017)
18. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
19. Shi, E., Chung, H., Wu, K.: What can cryptography do for decentralized mechanism design? In: ITCS 2023
20. Tsabary, I., Yechieli, M., Manuskin, A., Eyal, I.: MAD-HTLC: because HTLC is crazy-cheap to attack. In: S&P (2021)
21. Wadhwa, S., Stoeter, J., Zhang, F., Nayak, K.: He-htlc: Revisiting incentives in htlc. Cryptology ePrint Archive, Paper 2022/546 (2022). https://doi.org/10.14722/ndss.2023.24775, https://eprint.iacr.org/2022/546, https://eprint.iacr.org/2022/546

22. Winzer, F., Herd, B., Faust, S.: Temporary censorship attacks in the presence of rational miners. In: IEEE European Symposium on Security and Privacy Workshops (2019)
23. Wu, K., Asharov, G., Shi, E.: A complete characterization of game-theoretically fair, multi-party coin toss. In: Eurocrypt (2022)

A Comparison of Knowledge-Coin Exchange Protocols

Table 3: Solidity gas cost comparison. (O) denotes the optimistic case.

Contract	**Redeem path**	**Gas**
HTLC	Alice redeem	35,851
	Bob redeem	34,932
MAD-HTLC	(O) Alice and Bob	102,505
	Refund, Bob	104,611
	Deposit bomb, Miner	61,008
	Collateral bomb, Miner	46,063
He-HTLC	(O) Alice and Bob	72,723
	Refund, Bob	123,337
	Collateral bomb, Miner	70,327
Rapidash	(O) (P_{default}), Alice and Bob	73,246
	Refund ($P_{\mathsf{refund}} + C_{\mathsf{refund}}$), Bob	123,543
	Bomb (C_{burn}), Miner	70,327

B Rapidash Disincentivizes a 100% Coalition

So far, to prove our coalition-resistant fairness notions, we assumed that the coalition wields strictly less than 100% of the mining power. Take the knowledge-coin protocol for example: if Bob can solicit a coalition of 100% of the mining power, then its best strategy is to wait for Alice to post pre_s, and then activate P_{refund} and C_{refund}. In this way, Bob and the coalition effectively learns the secret pre_s for free.

In this section, we provide some justification about this assumption, and some evidence why 100% coalition is difficult to form in permissionless environment for Rapidash. We also compare Rapidash with existing approaches like HTLC and explain why existing approaches are susceptible to a 100% coalition.

B.1 The Meta-Game of Coalition Formation

We argue that in a permissionless proof-of-work setting and under some mild assumptions, Rapidash disincentivizes a 100% coalition to form, even in a world

where one can post bribery contracts [2,9,11,22] or other smart contracts in an attempt to openly solicit everyone.

More specifically, suppose that 100% of the miners are colluding with Bob through some joint strategy S, which invokes P_{refund} and C_{refund} with some non-negligible probability (since invoking P_{refund} and C_{refund} is the only way for a Bob-coalition to gain). One should think of the strategy as a general Turing Machine that can adaptively decide how to act based on the view in the protocol so far.

We make a few mild assumptions for our analysis. We assume that there exists some small miner i^* with a relatively small fraction of mining power such that its influence to the block generation process is small, and moreover, the small miner receives no more than its fair share of profit if it joined the coalition (where fair means proportional to mining power). We also assume a permissionless setting where the strategy S cannot tell if all miners have joined and make use of this information. Now, if i^* joins the coalition and cooperates, its expected reward is at most $p\gamma \cdot \$v$, where p is the probability P_{refund} is invoked and γ denotes its mining power. Note that $\$v$ is the coalition's maximum total gain possible. Now, suppose i^* chooses to not join the coalition, since its influence to the block generation process is small, we may assume that P_{refund} is invoked with probability p or more. Now, the moment P_{refund} is activated, i^* has a T_2 lead in time to mine a block in which i^* can redeem $\$\epsilon$ from the C_{burn} branch. In particular, without loss of generality, we may assume that every miner in the coalition commits to starving C_{burn} in every block they mine, e.g., by placing a collateral that it will honor its commitment — if not, then the coalition will not be stable since a coalition member will be incentivized to defect from the coalition and claim C_{burn} itself. This means that if i^* mines a block during the T_2 window after the activation of P_{refund}, i^* can claim $\$\epsilon$ from C_{burn} for itself. Suppose that $T_2 > 1$. The probability that i^* mines a block in a window of T_2 length is $1-(1-\gamma)^{T_2}$. Therefore, if i^* do not join the coalition, its expected gain would be at least $p \cdot \$\epsilon \cdot (1-(1-\gamma)^{T_2})$. If i^* joins the coalition, its expected gain is $p\gamma \cdot \$v$. Thus, as long as $p \cdot \$\epsilon \cdot (1-(1-\gamma)^{T_2}) > p\gamma \cdot \$v$, i^*'s best strategy is to not join the coalition. This means that if everyone else joins the coalition, some small user i^* wants to defect. In other words, a 100% coalition is not an equilibrium of the coalition-forming meta-game. For example, if we choose $T_2 = 2$, then it suffices to choose $\$\epsilon > \$v \cdot \frac{1}{2-\gamma}$.

As a special case and sanity check, the parameter constraints above implies that $\$\epsilon > \$v/T_2$. If $\$\epsilon < \$v/T_2$, Bob would be able bribe every miner that starves Alice's transaction $\$v/T_2$ such that every miner would want to cooperate — as explained shortly afterwards, the standard HTLC contract is subject to such a bribery attack.

The above argument is for the knowledge-coin exchange protocol. For our atomic swap protocol, essentially the same meta-game analysis would apply.

B.2 Comparison with Prior Approaches

Using this coalition formation meta-game perspective, we give a re-exposition of some incentive attacks for standard HTLC and MAD-HTLC [20].

Meta-games for HTLC. Recall that in a HTLC, Alice can obtain $\$v$ by revealing the preimage. On the other hand, after timeout T, Bob can request his deposit back. Consider the following attack. Bob can post a bribery contract soliciting participation of miners: if Alice's redeeming transaction is censored until Bob claims the $\$v$ back through pre_b, then, Bob will equally re-distribute $\$(v-\epsilon)$ to every miner that helped to mine a block that starved Alice's transaction where $\$\epsilon$ is a small amount Bob keeps for himself. Suppose the transaction fees are 0, then every miner's best strategy is to join the coalition, and thus a 100% coalition is an equilibrium of the meta-game.

MAD-HTLC. MAD-HTLC attempts to mitigate the attack we described for HTLC. It has Bob draw a random secret value pre_b, reveal its hash $h_b = H(pre_b)$, and deposit $\$v$ coins in the following smart contract [8]:

MAD-HTLC

- **On receiving** z_1 from Alice such that $H(z_1) = h_s$: send $\$v$ to Alice.
- **After** T, **on receiving** z_2 from Bob such that $H(z_2) = h_b$: send $\$v$ to Bob.
- **On receiving** (z_1, z_2) from anyone P such that $H(z_1) = h_s$ and $H(z_2) = h_b$: send $\$v$ to P.

Here, the attack outlined above is not possible, as any miner who sees both Alice's and Bob's transactions (and hence learns both pre_s and pre_b) would simply use these to grab the reward $\$v$ for themselves. However, *as admitted by the MAD-HTLC authors (Sec. 8 of [20]), MAD-HTLC does not defend against general user-miner collusion where users and miners can enter into arbitrary binding contracts.* Bob could propose a contract (e.g., on another chain, or even a physical legally-binding one) to some miners, and as soon as Alice posts pre_s, if the colluding miners happen to mine the next block, they can exclude Alice's transaction and redeem the $\$v$ coins for themselves by posting both (pre_s, pre_b). Then, using the binding side contract, the coalition can split off the $\$v$ coins among its members. It could also be that Bob is a miner himself. In this case, if Bob happens to mine the next block after Alice posts pre_s, Bob can get the secret for free.

The result of MAD-HTLC can be viewed as follows: by allowing the miner to claim $\$v$ itself through (pre_s, pre_b), it removes the undesirable 100%-coalition equilibrium in the coalition formation meta-game — the design of Rapidash is inspired by this elegant idea. Unfortunately, the design of MAD-HTLC incentivizes coalitions (with binding side contracts) to deviate in the protocol game

[8] MAD-HTLC has extra logic to defend against a spiteful Bob which we omit for simplicity. This logic does not mitigate the coalition attacks MAD-HTLC is susceptible to.

itself. As we discussed earlier, Bob colluding with a miner should always deviate: if it happens to be the miner when Alice posts pre_s, the coalition should always starve Alice's transaction and claim the $\$v$ itself by posting (pre_s, pre_b).

C Bob's Protocol for Atomic Swap

Atomic Swap Protocol — Bob (Section 4.1)

Preparation Phase:

1. At $t = 0$, Bob sends the deposit transaction of $\ddot{\mathrm{B}}x_b + \ddot{\mathrm{B}}c_b^{\mathsf{B}}$ to RAPIDASH$^{\mathrm{B}}$ and sends the collateral transaction of $\ddot{\mathrm{A}}c_b^{\mathsf{A}}$ to RAPIDASH$^{\mathrm{A}}$. [a]
2. Wait until one of the following happens:
   - Both RAPIDASH$^{\mathrm{B}}$ and RAPIDASH$^{\mathrm{A}}$ are active: Bob sends pre_c to $P^{\mathsf{B}}_{\mathsf{default}}$ and enters the execution phase.
   - Either RAPIDASH$^{\mathrm{B}}$ or RAPIDASH$^{\mathrm{A}}$ has not been active, and it is at least BobChain time T: Bob enters the abort phase.

Execution Phase:

1. Wait until one of the following happens:
   - Alice already sent pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$, and it is before BobChain time T_1^{B}: Bob sends pre_s to $P^{\mathsf{A}}_{\mathsf{default}}$.
   - Alice has not sent pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$, and it is at least BobChain time T_1^{B}: Bob sends pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$ at BobChain time T_1^{B}.
2. If τ^{B} BobChain *time* has passed since $P^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C^{\mathsf{B}}_{\mathsf{refund}}$. As soon as $C^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Bob sends ping to $P^{\mathsf{A}}_{\mathsf{refund}}$.
3. If τ^{A} AliceChain *time* has passed since $P^{\mathsf{A}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C^{\mathsf{A}}_{\mathsf{refund}}$.

Abort Phase:

1. At BobChain time T_0^{B}, Bob sends ping to $P^{\mathsf{A}}_{\mathsf{refund}}$ and pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$.
2. If τ^{A} AliceChain *time* has passed since $P^{\mathsf{A}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C^{\mathsf{A}}_{\mathsf{refund}}$; similarly, if τ^{B} BobChain *time* has passed since $P^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C^{\mathsf{B}}_{\mathsf{refund}}$.

Ignore all other events.

[a] Notice that only Bob needs to put collateral on both chains.

D Knowledge-Coin Exchange: Proof of Achieving CSP-Fairness and Dropout Resilience

Lemma D.1 (Alice-miner coalition). *Let $\mathcal{C}$ be any coalition that consists of Alice and an arbitrary subset of miners*[9] *(possibly no miner). Then, if $\$\epsilon < \$v$, for any (even unbounded) coalition strategy $S_{\mathcal{C}}$,*

$$\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, HS_{-\mathcal{C}}) \leq \mathsf{util}^{\mathcal{C}}(HS_{\mathcal{C}}, HS_{-\mathcal{C}})$$

where $HS_{-\mathcal{C}}$ denotes the honest strategy for everyone not in $\mathcal{C}$.

Proof. When the coalition $\mathcal{C}$ follows the protocol, they will send pre_s at $t = 0$, and P_{default} will be activated in the next block. In this case, the utility of $\mathcal{C}$ is $\$v - \$v_a$.

Now, consider the case that the coalition $\mathcal{C}$ deviates from the honest strategy. We may assume that the coalition does not post any new smart contract on the fly and deposit money into it[10] — if it did so, it cannot recover more than its deposit since any player not in $\mathcal{C}$ will not invoke the smart contract. There are two possibilities:

- First, P_{default} is activated at some point. In this case, nothing else can be activated. Thus, the utility of $\mathcal{C}$ is $\$v - \$v_a$, which is the same as the honest case.
- Second, P_{default} is never activated. The Alice-miner coalition cannot cash out from P_{refund} or C_{refund}, it can only cash out ϵ from C_{burn}. However, when C_{burn} is activated, pre_s is publicly known, so the utility of $\mathcal{C}$ is $\$\epsilon - \$v_a$, which is less than the honest case since $\$\epsilon < \$v$.

Lemma D.2 (Bob-miner coalition). *Let $\mathcal{C}$ be any coalition that consists of Bob and a subset of miners controlling at most γ fraction of mining power. Then, as long as $\$c_b < \$\epsilon$ and $\gamma^{T_2} \leq \frac{\$c_b}{\$c_b + \$v}$, for any (even unbounded) coalition strategy $S_{\mathcal{C}}$, it must be that*

$$\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, HS_{-\mathcal{C}}) \leq \mathsf{util}^{\mathcal{C}}(HS_{\mathcal{C}}, HS_{-\mathcal{C}}).$$

Proof. The honest Alice will always send pre_s to P_{default}. Thus, when $\mathcal{C}$ follows the protocol, P_{default} will be activated in the next block, and the utility of $\mathcal{C}$ is $\$v_b - \$v$.

Now, suppose $\mathcal{C}$ may deviate from the protocol. As in Lemma D.1, we may assume that the coalition does not post any new smart contract on the fly and deposit money into it. There are three cases.

[9] We assume that the coalition cannot break the underlying consensus layer. If the underlying consensus actually secures against, say, honest majority, then essentially the lemma holds for any coalition that wields minority of the mining power.

[10] However, the coalition $\mathcal{C}$ itself could be facilitated by smart contracts, our modeling of coalition already captures any arbitrary side contract within the coalition.

- First, neither P_{default} nor P_{refund} is activated. Because P_{refund} is not activated, C_{refund} cannot be activated. The Bob-miner coalition can only get $\$\epsilon$ from C_{burn}. Thus, the coalition's utility is at most $\$v_b - \$v - \$c_b + \$\epsilon < \$v_b - \$v$ where the inequality is due to the constraint $\$c_b > \$\epsilon$.
- Second, P_{default} is activated. In this case, nothing else can be activated, and the utility of $\mathcal{C}$ is $\$v_b - \$v$, which is the same as the honest case.
- Third, P_{refund} is activated. Let $t^* \geq T_1$ be the time at which P_{refund} is activated. There are two subcases. In the first subcase, the coalition also gets $\$\epsilon$ from C_{burn} during $[t^*, t^* + T_2]$. In this case, the coalition's utility is at most $\$v_b - \$c_b - \$v + \$\epsilon$, and since $\$c_b > \$\epsilon$, this is less than the honest case. Henceforth, we may assume that the coalition does not invoke C_{burn} after time t^* as after time $t^* + T_2$ it is always better to invoke C_{refund}. Since the honest Alice posts pre_s at $t = 0$ and $t^* \geq T_1$, both pre_s and pre_b are publicly known at t^*. Since all non-colluding miners are honest, after t^*, they will activate C_{burn} themselves when they mine a new block if C_{burn} has not already been activated before. If a non-colluding miner mines a new block during $(t^*, t^* + T_2]$, we say that the coalition loses the race. Otherwise, we say that the coalition wins the race. If the coalition loses the race, then it gets nothing from C_{refund} or C_{burn}, and thus its utility is at most $\$v_b - \$c_b - \$v$. Else if it wins the race, then the coalition's utility is at most $\$v_b$. The probability p that the coalition wins the race is upper bounded by $p \leq \gamma^{T_2}$. Therefore, the coalition's expected utility is at most

$$(\$v_b - \$c_b - \$v) \cdot (1 - p) + \$v_b \cdot p.$$

  For $(\$v_b - \$c_b - \$v) \cdot (1 - p) + \$v_b \cdot p$ to exceed the honest utility $\$v_b - \$v$, it must be that $p > \frac{\$c_b}{\$c_b + \$v}$ which contradicts our assumption.

We thus conclude that $\mathcal{C}$ cannot increase its utility through any deviation.

Theorem D.3 (CSP fairness). *Suppose that the hash function $H(\cdot)$ is a one-way function, $\$c_b < \$\epsilon$, $\$\epsilon < \$v$, and $\gamma^{T_2} \leq \frac{\$c_b}{\$c_b + \$v}$. Then, the* RapidashKC *protocol satisfies γ-CSP-fairness.*

Proof. Lemmas D.1 and D.2 proved γ-CSP-fairness for the cases when the coalition consists of either Alice or Bob, and possibly some miners. Since by our assumption, Alice and Bob are not in the same coalition, it remains to show γ-CSP-fairness for the case when the coalition consists only of some miners whose mining power does not exceed γ. Since both Alice and Bob are honest, the coalition's utility is 0 unless C_{burn} is activated. However, C_{burn} requires that $\mathcal{C}$ to find pre_b on its own — the probability of this happening is negligibly small due to the one-wayness of the hash function $H(\cdot)$.

We now prove that RapidashKC is dropout resilient.

Theorem D.4 (Dropout resilience). *Suppose that $H(\cdot)$ is a one-way function and that all players are PPT machines.* RapidashKC *is dropout resilient.*

Proof. Throughout the proof, for any $X \in \{pre_s, pre_b\}$, we ignore the negligible probability that the miners can find the preimage X by itself if Alice and Bob have never sent X before.

We first analyze the case where Alice drops out. There are two possible case: 1) Alice drops out before posting a transaction containing pre_s; 2) Alice drops out after she already posted a transaction containing pre_s at $t = 0$. In the first case, as long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, Bob would activate P_{refund} and C_{refund} in polynomial time except with negligible probability, and his utility is 0 since he simply gets all his deposit back. In the second case, the honest Bob will not post pre_b to P_{refund}. An honest miner would include Alice's transaction and activate P_{default}. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P_{default} will be activated in polynomial time except with negligible probability. As a result, Bob's utility is $\$v_b - \$v > 0$.

Next, we analyze the case where Bob drops out. In this case, Alice always posts a transaction containing pre_s, and except with negligible probability, P_{default} will always be activated. Thus, Alice's utility is always $\$v - \$v_a > 0$.

To sum up, in all cases, the utility of the remaining party is always non-negative except with negligible probability.

E Atomic Swap: Proof of Achieving CSP-Fairness and Dropout Resilience

Before proving CSP fairness of the protocol, we give some useful lemmas. CSP fairness is formally proven by Theorem E.5.

We define the *net profit* of $\mathcal{C}$ from $\textsc{Rapidash}^{\mathrm{B}}$ to be the coins that $\mathcal{C}$ gets from $\textsc{Rapidash}^{\mathrm{B}}$ *minus* the coins that $\mathcal{C}$ deposits into $\textsc{Rapidash}^{\mathrm{B}}$. The net profit of $\mathcal{C}$ from $\textsc{Rapidash}^{\mathrm{A}}$ is defined similarly. Notice that the net profit might be negative, which means $\mathcal{C}$ deposits more coins than what it gets.

Lemma E.1. *Suppose both* $\textsc{Rapidash}^{\mathrm{B}}$ *and* $\textsc{Rapidash}^{\mathrm{A}}$ *are active. Suppose the coalition* $\mathcal{A}$ *consists of Alice and an arbitrary* $\gamma \in [0, 1]$ *fraction of the mining power. If* $₳c_a^{\mathsf{A}} > ₳\epsilon^{\mathsf{A}}$*, the utility of* $\mathcal{A}$ *can be more than the honest case, that is,* $\$\mathsf{AV}(₿x_b - ₳x_a)$*, only if one of the following holds*

- $P_{\mathsf{default}}^{\mathsf{B}}$, $P_{\mathsf{refund}}^{\mathsf{A}}$ *and* $C'^{\mathsf{A}}_{\mathsf{refund}}$ *are activated;*
- $C'^{\mathsf{B}}_{\mathsf{burn}}$, $P_{\mathsf{refund}}^{\mathsf{A}}$ *and* $C_{\mathsf{refund}}^{\mathsf{A}}$ *are activated.*

Proof. First, we prove that either $P_{\mathsf{default}}^{\mathsf{B}}$ or $C_{\mathsf{burn}}^{\mathsf{B}}$ is the necessary condition for the utility of $\mathcal{A}$ to be more than the honest case. For the sake of reaching a contradiction, suppose neither of $P_{\mathsf{default}}^{\mathsf{B}}$ or $C_{\mathsf{burn}}^{\mathsf{B}}$ is activated. Because $\mathcal{A}$ cannot get any coin from $P_{\mathsf{refund}}^{\mathsf{B}}$ or $C'^{\mathsf{B}}_{\mathsf{refund}}$, the net profit from $\textsc{Rapidash}^{\mathrm{B}}$ is at most 0. However, because $₳c_a^{\mathsf{A}} > ₳\epsilon^{\mathsf{A}}$, we have $₳\epsilon^{\mathsf{A}} - ₳x_a - ₳c_a^{\mathsf{A}} < -₳x_a < 0$. Thus, the net profit from $\textsc{Rapidash}^{\mathrm{A}}$ is also at most 0. Consequently, the utility of $\mathcal{A}$ is at most zero, which is less than $\$\mathsf{AV}(₿x_b - ₳x_a)$. Thus, one of $P_{\mathsf{default}}^{\mathsf{B}}$ and $C_{\mathsf{burn}}^{\mathsf{B}}$ must be activated.

Next, we prove that $P^{\mathsf{A}}_{\mathsf{refund}}$ and $C'^{\mathsf{A}}_{\mathsf{refund}}$ must be activated for the utility of $\mathcal{A}$ to be more than the honest case. For the sake of reaching a contradiction, suppose one of them is not activated. Because $\ddot{\mathbb{A}}c^{\mathsf{A}}_a > \ddot{\mathbb{A}}\epsilon^{\mathsf{A}} > 0$, the net profit from RAPIDASH$^{\mathrm{A}}$ is at most $-\ddot{\mathbb{A}}x_a$ since $P^{\mathsf{A}}_{\mathsf{refund}}$ or $C'^{\mathsf{A}}_{\mathsf{refund}}$ is not activated. However, the net profit from RAPIDASH$^{\mathrm{B}}$ is at most $\ddot{\mathbb{B}}x_b$. Thus, the utility of $\mathcal{A}$ is at most $\$\mathsf{AV}(\ddot{\mathbb{B}}x_b - \ddot{\mathbb{A}}x_a)$, which is the same as the honest case. Thus, both of $P^{\mathsf{A}}_{\mathsf{refund}}$ and $C'^{\mathsf{A}}_{\mathsf{refund}}$ must be activated.

Lemma E.2 (Alice-miner coalition). *Suppose that the hash function $H(\cdot)$ is a one-way function. Let $\mathcal{A}$ be any coalition that consists of Alice and $\gamma \in [0,1]$ fraction of mining power. Then, as long as $\gamma^{\tau^{\mathsf{A}}} \leq \frac{\ddot{\mathbb{A}}c^{\mathsf{A}}_a}{\ddot{\mathbb{A}}c^{\mathsf{A}}_a + \ddot{\mathbb{A}}x_a}$, for any PPT coalition strategy $S_{\mathcal{A}}$, except with negligible probability, it must be*

$$\mathsf{util}^{\mathcal{A}}(S_{\mathcal{A}}, HS_{-\mathcal{A}}) \leq \mathsf{util}^{\mathcal{A}}(HS_{\mathcal{A}}, HS_{-\mathcal{A}}),$$

where $HS_{\mathcal{A}}$ and $HS_{-\mathcal{A}}$ denotes the honest strategy for coalition $\mathcal{A}$ and everyone not in $\mathcal{A}$, respectively.

Proof. Recall that the utility of $\mathcal{A}$ is $\$\mathsf{AV}(\ddot{\mathbb{B}}x_b - \ddot{\mathbb{A}}x_a) > 0$ under an honest execution. Now, suppose $\mathcal{A}$ may deviate from the protocol. We may assume that the coalition does not post any new smart contract on the fly and deposit money into it (see the definition of strategy space in Section 2.1) — if it did so, it cannot recover more than its deposit since any player not in $\mathcal{A}$ will not invoke the smart contract. We analyze the possible cases depending on which phase Bob enters.

Bob enters the abort phase. If RAPIDASH$^{\mathrm{B}}$ has never been active, the net profit of $\mathcal{A}$ from RAPIDASH$^{\mathrm{B}}$ is at most zero. Now, assume RAPIDASH$^{\mathrm{B}}$ is active. When Bob enters the abort phase, he never sends any transaction containing pre_c. Ignoring the negligible probability that $\mathcal{A}$ finds pre_c by itself, $P^{\mathsf{B}}_{\mathsf{default}}$ or $C^{\mathsf{B}}_{\mathsf{burn}}$ can never be activated. Because Alice does not get any coin from $P^{\mathsf{B}}_{\mathsf{refund}}$ or $C'^{\mathsf{B}}_{\mathsf{refund}}$, the net profit of $\mathcal{A}$ from RAPIDASH$^{\mathrm{B}}$ is at most zero. On the other hand, because $\ddot{\mathbb{A}}c^{\mathsf{A}}_a > \ddot{\mathbb{A}}\epsilon^{\mathsf{A}}$, the net profit of $\mathcal{A}$ from RAPIDASH$^{\mathrm{A}}$ is at most zero, no matter whether RAPIDASH$^{\mathrm{A}}$ is active or not.

To sum up, except with negligible probability, the utility of $\mathcal{A}$ is at most zero, which is less than the honest case.

Bob enters the execution phase. If Bob enters the execution phase, both RAPIDASH$^{\mathrm{B}}$ and RAPIDASH$^{\mathrm{A}}$ must be acitve. By Lemma E.1, the utility of $\mathcal{A}$ can exceed the honest case only when $(P^{\mathsf{B}}_{\mathsf{default}} + P^{\mathsf{A}}_{\mathsf{refund}} + C'^{\mathsf{A}}_{\mathsf{refund}})$ or $(C^{\mathsf{B}}_{\mathsf{burn}} + P^{\mathsf{A}}_{\mathsf{refund}} + C'^{\mathsf{A}}_{\mathsf{refund}})$ are activated. Henceforth, we assume either $(P^{\mathsf{B}}_{\mathsf{default}} + P^{\mathsf{A}}_{\mathsf{refund}} + C'^{\mathsf{A}}_{\mathsf{refund}})$ or $(C^{\mathsf{B}}_{\mathsf{burn}} + P^{\mathsf{A}}_{\mathsf{refund}} + C'^{\mathsf{A}}_{\mathsf{refund}})$ are activated. Notice that in either case, $P^{\mathsf{A}}_{\mathsf{refund}}$ must be activated. When Bob enters the execution phase, $P^{\mathsf{A}}_{\mathsf{refund}}$ can be activated only either 1) by Bob sending ping to $P^{\mathsf{A}}_{\mathsf{refund}}$ after $C'^{\mathsf{B}}_{\mathsf{refund}}$ has been activated, or 2) by Alice sending pre_a to $P^{\mathsf{A}}_{\mathsf{refund}}$. Consider the first scenario. In this case, since $C'^{\mathsf{B}}_{\mathsf{refund}}$ has been activated, Alice can't get any money from RAPIDASH$^{\mathrm{B}}$. However, from RAPIDASH$^{\mathrm{A}}$, Alice can get at most zero. Thus, the

utility of $\mathcal{A}$ is less than the honest case. Now consider the second case. Suppose that $P^{\mathsf{A}}_{\mathsf{refund}}$ is activated at AliceChain time $t^* \geq T^{\mathsf{A}}_1$, so pre_a is publicly known after AliceChain time t^*.

Now, notice that if $P^{\mathsf{B}}_{\mathsf{default}}$ or $C^{\mathsf{B}}_{\mathsf{burn}}$ is activated, $\mathcal{A}$ has to send a transaction containing pre_s.

- *Case 1: $\mathcal{A}$ sends a transaction containing pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$ or $C^{\mathsf{B}}_{\mathsf{burn}}$ before* BobChain *time T^{B}_1.* Since BobChain time T^{B}_1 is earlier than AliceChain time T^{A}_1, pre_s and pre_a are both publicly known at AliceChain time t^*. Thus, during AliceChain time $(t^*, t^* + \tau^{\mathsf{A}}]$, any miner in $-\mathcal{A}$ will activate $C^{\mathsf{A}}_{\mathsf{burn}}$ if it wins a block. We say $\mathcal{A}$ loses the race if a non-colluding miner mines a new block during AliceChain time $(t^*, t^* + \tau^{\mathsf{A}}]$. Otherwise, we say $\mathcal{A}$ wins the race. If $\mathcal{A}$ loses the race, it gets nothing from $C^{\mathsf{A}}_{\mathsf{refund}}$ or $C^{\mathsf{A}}_{\mathsf{burn}}$, and its utility is at most $\$\mathsf{AV}(\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a - \ddot{\mathrm{A}}c^{\mathsf{A}}_a)$. Else if $\mathcal{A}$ wins the race, then its utility is at most $\$\mathsf{AV}(\ddot{\mathrm{B}}x_b)$, which can be achieved by activating $P^{\mathsf{A}}_{\mathsf{refund}}$, $C^{\mathsf{A}}_{\mathsf{refund}}$ and $P^{\mathsf{B}}_{\mathsf{default}}$. The probability p that $\mathcal{A}$ wins the race is upper bounded by $p \leq \gamma^{\tau^{\mathsf{A}}}$. Therefore, the expected utility of $\mathcal{A}$ is upper bounded by

$$\$\mathsf{AV}((\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a - \ddot{\mathrm{A}}c^{\mathsf{A}}_a) \cdot (1-p) + \ddot{\mathrm{B}}x_b \cdot p).$$

  Since $p \leq \gamma^{\tau^{\mathsf{A}}} \leq \frac{\ddot{\mathrm{A}}c^{\mathsf{A}}_a}{\ddot{\mathrm{A}}c^{\mathsf{A}}_a + \ddot{\mathrm{A}}x_a}$, we have $p \cdot \$\mathsf{AV}(\ddot{\mathrm{A}}c^{\mathsf{A}}_a + \ddot{\mathrm{A}}x_a) \leq \$\mathsf{AV}(\ddot{\mathrm{A}}c^{\mathsf{A}}_a)$. Thus, we have

$$\$\mathsf{AV}((\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a - \ddot{\mathrm{A}}c^{\mathsf{A}}_a) + p \cdot \$\mathsf{AV}(\ddot{\mathrm{A}}c^{\mathsf{A}}_a + \ddot{\mathrm{A}}x_a)) \leq \$\mathsf{AV}(\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a).$$

  Finally, we obtain

$$\$\mathsf{AV}((\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a - \ddot{\mathrm{A}}c^{\mathsf{A}}_a) \cdot (1-p) + \ddot{\mathrm{B}}x_b \cdot p) \leq \$\mathsf{AV}(\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a),$$

  which implies the strategic utility is upper bounded by the utility of the honest case.
- *Case 2: $\mathcal{A}$ does not send any transaction containing pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$ or $C^{\mathsf{B}}_{\mathsf{burn}}$ before* BobChain *time T^{B}_1.* In this case, the honest Bob will send pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$ at BobChain time T^{B}_1. Because $P^{\mathsf{A}}_{\mathsf{refund}}$ is activated at AliceChain time $t^* \geq T^{\mathsf{A}}_1$, which is later than BobChain time T^{B}_1, pre_a and pre_b are both publicly known at AliceChain time t^*. Thus, during AliceChain time $(t^*, t^* + \tau^{\mathsf{A}}]$, any miner in $-\mathcal{A}$ will activate $C^{\mathsf{A}}_{\mathsf{burn}}$ if it wins a block. By the same calculation as the previous case, since $p \leq \gamma^{\tau^{\mathsf{A}}} \leq \frac{\ddot{\mathrm{A}}c^{\mathsf{A}}_a}{\ddot{\mathrm{A}}c^{\mathsf{A}}_a + \ddot{\mathrm{A}}x_a}$, we have $\$\mathsf{AV}((\ddot{\mathrm{A}}x_a - \ddot{\mathrm{A}}c^{\mathsf{A}}_a + \ddot{\mathrm{B}}x_b) \cdot (1-p) + \ddot{\mathrm{B}}x_b \cdot p) \leq \$\mathsf{AV}(\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a)$.

Lemma E.3. *Suppose* RapidashB *and* RapidashA *are both active. Suppose the coalition $\mathcal{B}$ consists of Bob and an arbitrary $\gamma \in [0,1]$ fraction of the mining power. If $\ddot{\mathrm{B}}c^{\mathsf{B}}_b > \ddot{\mathrm{B}}\epsilon^{\mathsf{B}}$ and $\ddot{\mathrm{A}}c^{\mathsf{A}}_b > \ddot{\mathrm{A}}\epsilon^{\mathsf{A}}$, the utility of $\mathcal{B}$ can be more than the honest case, that is, $\$\mathsf{BV}(\ddot{\mathrm{A}}x_a - \ddot{\mathrm{B}}x_b)$, only if $P^{\mathsf{B}}_{\mathsf{refund}}$, $C^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$ are activated.*

Proof. First, note that $P^{\mathsf{B}}_{\mathsf{default}}$ and $P^{\mathsf{B}}_{\mathsf{refund}}$ are mutually exclusive, and neither $C^{\mathsf{B}}_{\mathsf{refund}}$ nor $C^{\mathsf{B}}_{\mathsf{burn}}$ can be activated after $P^{\mathsf{B}}_{\mathsf{default}}$ because not enough money is

which is activated	net profit of Bob's coalition
none or only $P^{\mathsf{B}}_{\mathsf{refund}}$	$-\ddot{\mathbb{B}}x_b - \ddot{\mathbb{B}}c^{\mathsf{B}}_b$
$P^{\mathsf{B}}_{\mathsf{default}}$	$-\ddot{\mathbb{B}}x_b$
$P^{\mathsf{B}}_{\mathsf{refund}} + C^{\mathsf{B}}_{\mathsf{refund}}$	0
$C^{\mathsf{B}}_{\mathsf{burn}}$ or $P^{\mathsf{B}}_{\mathsf{refund}} + C^{\mathsf{B}}_{\mathsf{burn}}$	$\leq \ddot{\mathbb{B}}\epsilon^{\mathsf{B}} - \ddot{\mathbb{B}}x_b - \ddot{\mathbb{B}}c^{\mathsf{B}}_b$

Table 4: The net profit of Bob's coalition from RAPIDASH$^{\mathrm{B}}$, assuming that RAPIDASH$^{\mathrm{B}}$ is active.

available in the contract. Moreover, $C^{\mathsf{B}}_{\mathsf{refund}}$ and $C^{\mathsf{B}}_{\mathsf{burn}}$ are mutually exclusive. Thus, all the possible cases for the net profit of Bob's coalition from RAPIDASH$^{\mathrm{B}}$ can be summarized as shown in Table 4.

Similarly, if $P^{\mathsf{A}}_{\mathsf{default}}$ is activated, no other activation points of RAPIDASH$^{\mathrm{A}}$ can be activated. Moreover, $C^{\mathsf{A}}_{\mathsf{refund}}$ and $C^{\mathsf{A}}_{\mathsf{burn}}$ are mutually exclusive. Thus, all the possible cases for the net profit of Bob's coalition from RAPIDASH$^{\mathrm{B}}$ can be summarized as shown in Table 5.

which is activated	net profit of Bob's coalition
none or only $P^{\mathsf{A}}_{\mathsf{refund}}$	$-\ddot{\mathbb{A}}c^{\mathsf{A}}_b$
$P^{\mathsf{A}}_{\mathsf{default}}$	$\ddot{\mathbb{A}}x_a$
$P^{\mathsf{A}}_{\mathsf{refund}} + C^{\mathsf{A}}_{\mathsf{refund}}$	0
$C^{\mathsf{A}}_{\mathsf{burn}}$ or $P^{\mathsf{A}}_{\mathsf{refund}} + C^{\mathsf{A}}_{\mathsf{burn}}$	$\leq \ddot{\mathbb{A}}\epsilon^{\mathsf{A}} - \ddot{\mathbb{A}}c^{\mathsf{A}}_b$

Table 5: The net profit of Bob's coalition from RAPIDASH$^{\mathrm{A}}$, assuming that RAPIDASH$^{\mathrm{A}}$ is active.

Suppose the coalition $\mathcal{C}$ consists of the miners and Bob. If $\mathcal{C}$ follows the protocol, $P^{\mathsf{B}}_{\mathsf{default}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$ will be activated, and the utility of $\mathcal{C}$ is $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b) > 0$. When $P^{\mathsf{B}}_{\mathsf{refund}}$, $C^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$ are activated, $\mathcal{C}$'s utility is $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a)$. Now, we will show that it is the only scenario for $\mathcal{C}$'s utility to exceed the honest case. For the sake of reaching a contradiction, suppose $\mathcal{C}$'s utility is strictly greater than $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b)$, while one of $P^{\mathsf{B}}_{\mathsf{refund}}$, $C^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$ is not activated. There are two subcases.

- **Subcase 1: $P^{\mathsf{A}}_{\mathsf{default}}$ is not activated.** Because $\ddot{\mathbb{A}}c^{\mathsf{A}}_b > \ddot{\mathbb{A}}\epsilon^{\mathsf{A}}$, we have $\ddot{\mathbb{A}}\epsilon^{\mathsf{A}} - \ddot{\mathbb{A}}c^{\mathsf{A}}_b < 0$. Thus, if $P^{\mathsf{A}}_{\mathsf{default}}$ is not activated, the net profit from RAPIDASH$^{\mathrm{A}}$ is at most 0. Because $\ddot{\mathbb{B}}c^{\mathsf{B}}_b > \ddot{\mathbb{B}}\epsilon^{\mathsf{B}}$, we have $\ddot{\mathbb{B}}\epsilon^{\mathsf{B}} - \ddot{\mathbb{B}}x_b - \ddot{\mathbb{B}}c^{\mathsf{B}}_b < -\ddot{\mathbb{B}}x_b$. Thus, the net profit from RAPIDASH$^{\mathrm{B}}$ is also at most 0. Consequently, the utility of $\mathcal{C}$ is at most zero, which is less than $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b)$.
- **Subcase 2: $P^{\mathsf{B}}_{\mathsf{refund}}$ or $C^{\mathsf{B}}_{\mathsf{refund}}$ is not activated.** Because $\ddot{\mathbb{B}}c^{\mathsf{B}}_b > \ddot{\mathbb{B}}\epsilon^{\mathsf{B}} \geq 0$, the net profit from RAPIDASH$^{\mathrm{B}}$ is at most $-\ddot{\mathbb{B}}x_b$ since $P^{\mathsf{B}}_{\mathsf{refund}}$ or $C^{\mathsf{B}}_{\mathsf{refund}}$ is not activated. However, the net profit from RAPIDASH$^{\mathrm{A}}$ is at most $\ddot{\mathbb{A}}x_a$. Thus, the utility of $\mathcal{C}$ is at most $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b)$, which is the same as the honest case.

Therefore, we conclude that if $\mathcal{C}$'s utility is strictly greater than $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b)$, $P^{\mathsf{B}}_{\mathsf{refund}}$, $C^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$ must be activated.

Lemma E.4 (Bob-miner coalition). *Suppose that the hash function $H(\cdot)$ is a one-way function. Let $\mathcal{B}$ be any coalition that consists of Bob and a subset of miners controlling at most $\gamma \in [0,1]$ fraction of mining power. Then, as long as $\gamma^{\tau^{\mathsf{B}}} \leq \frac{\ddot{\mathbb{B}}c_b^{\mathsf{B}}}{\ddot{\mathbb{B}}c_b^{\mathsf{B}}+\ddot{\mathbb{B}}x_b}$, for any PPT coalition strategy $S_{\mathcal{B}}$, except with negligible probability, it must be*

$$\mathsf{util}^{\mathcal{B}}(S_{\mathcal{B}}, HS_{-\mathcal{B}}) \leq \mathsf{util}^{\mathcal{B}}(HS_{\mathcal{B}}, HS_{-\mathcal{B}}),$$

where $HS_{\mathcal{B}}$ and $HS_{-\mathcal{B}}$ denotes the honest strategy for coalition $\mathcal{B}$ and everyone not in $\mathcal{B}$, respectively.

Proof. Recall that the utility of $\mathcal{B}$ is $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b) > 0$ under an honest execution. Now, suppose $\mathcal{B}$ may deviate from the protocol. We may assume that the coalition does not post any new smart contract on the fly and deposit money into it — if it did so, it cannot recover more than its deposit since any player not in $\mathcal{B}$ will not invoke the smart contract. We analyze the two possible cases depending on which phase Alice enters.

Alice enters the abort phase. If $\textsc{Rapidash}^{\mathrm{A}}$ has never been active, the net profit of $\mathcal{B}$ from $\textsc{Rapidash}^{\mathrm{A}}$ is at most zero. Now, assume $\textsc{Rapidash}^{\mathrm{A}}$ is active. When Alice enters the abort phase, she never sends any transaction containing pre_s. Ignoring the negligible that $\mathcal{B}$ finds pre_s by itself, $P^{\mathsf{A}}_{\mathsf{default}}$ can never be activated. Because $\ddot{\mathbb{A}}c_b^{\mathsf{A}} > \ddot{\mathbb{A}}\epsilon^{\mathsf{A}}$, the net profit of $\mathcal{B}$ from $\textsc{Rapidash}^{\mathrm{A}}$ is at most zero. On the other hand, because $\ddot{\mathbb{B}}x_b > \ddot{\mathbb{B}}\epsilon^{\mathsf{B}}$, the net profit of $\mathcal{B}$ from $\textsc{Rapidash}^{\mathrm{B}}$ is at most zero, no matter $\textsc{Rapidash}^{\mathrm{B}}$ is active or not.

To sum up, except with negligible probability, the utility of $\mathcal{B}$ is at most zero, which is less than the honest case.

Alice enters the execution phase. By Lemma E.3, the utility of $\mathcal{B}$ can be more than the honest case only if $P^{\mathsf{B}}_{\mathsf{refund}}$, $C^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$ are activated, so we assume it is the case. Therefore, we may assume that $P^{\mathsf{B}}_{\mathsf{refund}}$ is activated at $\mathsf{BobChain}$ time $t^* \geq T_1^{\mathsf{B}}$, and pre_b is publicly known after $\mathsf{BobChain}$ time t^*. If Alice enters the execution, Bob must have sent pre_c before $\mathsf{BobChain}$ time T_0^{B}. Moreover, Alice sends pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$ at $\mathsf{BobChain}$ time T_0^{B} and $T_0^{\mathsf{B}} < T_1^{\mathsf{B}}$. Therefore, pre_s, pre_b and pre_c are all publicly known at $\mathsf{BobChain}$ time t^*. Thus, during $\mathsf{BobChain}$ time $(t^*, t^* + \tau^{\mathsf{B}}]$, any miner in $-\mathcal{B}$ will activate $C^{\mathsf{B}}_{\mathsf{burn}}$ if it wins a block. We say $\mathcal{B}$ loses the race if a non-colluding miner mines a new block during $\mathsf{BobChain}$ time $(t^*, t^* + \tau^{\mathsf{B}}]$. Otherwise, we say $\mathcal{B}$ wins the race. If $\mathcal{B}$ loses the race, it gets nothing from $C^{\mathsf{B}}_{\mathsf{refund}}$ or $C^{\mathsf{B}}_{\mathsf{burn}}$, and its utility is at most $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b - \ddot{\mathbb{B}}c_b^{\mathsf{B}})$ which can be achieved if $P^{\mathsf{A}}_{\mathsf{default}}$ is activated. Else if $\mathcal{B}$ wins the race, then its utility is at most $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a)$ which can be achieved by activating $P^{\mathsf{B}}_{\mathsf{refund}}$, $C^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$. Since $p \leq \gamma^{\tau^{\mathsf{B}}} \leq \frac{\ddot{\mathbb{B}}c_b^{\mathsf{B}}}{\ddot{\mathbb{B}}c_b^{\mathsf{B}}+\ddot{\mathbb{B}}x_b}$, we have

$$\$\mathsf{BV}((\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b - \ddot{\mathbb{B}}c_b^{\mathsf{B}}) \cdot (1-p) + \ddot{\mathbb{A}}x_a \cdot p) \leq \$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b).$$

Theorem E.5 (CSP fairness). *Suppose that the hash function $H(\cdot)$ is a one-way function. For any $\gamma \in [0,1]$, if the parameters satisfy the constraints specified in Section 4.1, then, the atomic swap protocol satisfies γ-CSP-fairness.*

Proof. In Lemma E.2 and Lemma E.4, we show that the atomic swap protocol satisfies γ-CSP-fairness when the coalition consists of Alice or Bob, and possibly with some miners. Because we assume that Alice and Bob are not in the same coalition, it remains to show γ-CSP-fairness when the coalition $\mathcal{C}$ consists only of miners controlling at most γ fraction of the mining power.

Henceforth, we assume Alice and Bob are both honest. It is clear from the protocol that the honest Alice and honest Bob always make the same decision whether to enter the execution phase or abort phase. We may assume that the coalition does not post any new smart contract on the fly and deposit money into it — if it did so, it cannot recover more than its deposit since any player not in $\mathcal{B}$ will not invoke the smart contract.

Next, when $\mathcal{C}$ follows the protocol, its utility is always zero. Suppose $\mathcal{C}$ may deviate from the protocol. Notice that the utility of $\mathcal{C}$ can be positive only when $C^{\mathsf{B}}_{\mathsf{burn}}$ or $C^{\mathsf{A}}_{\mathsf{burn}}$ is activated. There are two possible cases.

- *Case 1: both Alice and Bob enter the execution phase.* In this case, Alice always sends pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$, and she never sends any transaction containing pre_a. Ignoring the negligible probability that $\mathcal{C}$ finds pre_a by itself, $C^{\mathsf{A}}_{\mathsf{burn}}$ can never be activated. Moreover, Alice always sends pre_s to $P^{\mathsf{B}}_{\mathsf{default}}$ at latest at $\mathsf{BobChain}$ time T^{B}_0, and thus Bob will not post any transaction containing pre_b. Ignoring the negligible probability that $\mathcal{C}$ finds pre_b by itself, $C^{\mathsf{B}}_{\mathsf{burn}}$ can never be activated. To sum up, except the negligible probability, the utility of $\mathcal{C}$ is at most zero, which is the same as the honest case.
- *Case 2: both Alice and Bob enter the abort phase.* In this case, Alice never sends any transaction containing pre_s. Ignoring the negligible probability that $\mathcal{C}$ finds pre_s by itself, $C^{\mathsf{B}}_{\mathsf{burn}}$ can never be activated, and $C^{\mathsf{A}}_{\mathsf{burn}}$ can be activated only by (pre_a, pre_b). However, Bob always sends ping to $P^{\mathsf{A}}_{\mathsf{refund}}$ and pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$ at $\mathsf{BobChain}$ time T^{B}_0, so Alice never sends any transaction containing pre_a. Ignoring the negligible probability that $\mathcal{C}$ finds pre_a by itself, $C^{\mathsf{A}}_{\mathsf{burn}}$ cannot be activated by (pre_a, pre_b). To sum up, except with negligible probability, the utility of $\mathcal{C}$ is at most zero, which is the same as the honest case.

Remark E.6. The assumption that $\$\mathsf{AV}(\text{₿}x_b - \text{₳}x_a) > 0$ and $\$\mathsf{BV}(\text{₳}x_a - \text{₿}x_b) > 0$ (see Section 2.4) is crucial to prove CSP fairness, as it ensures that no PPT strategy outperforms the honest strategy. Otherwise, if the assumption does not hold, either Alice or Bob could prefer to drop out in order to get utility zero, since the utility of the honest case would be negative.

Nevertheless, our protocol still disincentivizes strategic parties from deviating from the protocol even if the assumption does not hold. Specifically, the protocol additionally guarantees that when the honest case yields negative utility, the best utility a strategic party can achieve is zero - equivalent to not participating in the protocol. To see this, notice that the proof of Lemma E.2 shows that

the strategic Alice's utility is either upper bounded by 0 or $\$\mathsf{AV}(\ddot{\mathbb{B}}x_b - \ddot{\mathbb{A}}x_a)$. Similarly, the proof of Lemma E.4 shows that the strategic Bob's utility is either upper bounded by 0 or $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b)$. Thus, if the assumption does not hold for the strategic parties, their utility is at most 0 for any PPT strategies, which is the same as not initiating the protocol.

Theorem E.7 (Dropout resilience of atomic swap). *Suppose that $H(\cdot)$ is a one-way function and that all players are PPT machines. Then, the atomic swap protocol is dropout resilient.*

Proof. Throughout the proof, for any $X \in \{pre_s, pre_b, pre_c, pre_a\}$, we ignore the negligible probability that the miners can find the preimage X by itself if Alice and Bob have never sent X before.

We first analyze the cases where Alice drops out with three possible cases.

- *Case 1: Bob enters the abort phase.* In this case, Bob will send pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$ and ping to $P^{\mathsf{A}}_{\mathsf{refund}}$ at $\mathsf{BobChain}$ time T^{B}_0. When τ^{B} $\mathsf{BobChain}$ time has passed since $P^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C'^{\mathsf{B}}_{\mathsf{refund}}$; when τ^{A} $\mathsf{AliceChain}$ time has passed since $P^{\mathsf{A}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C'^{\mathsf{A}}_{\mathsf{refund}}$. When Bob enters the abort phase, he never sends any transaction containing pre_c, and thus Alice never enters the execution phase and never sends any transaction containing pre_s no matter when she drops out. Because Bob sends ping to $P^{\mathsf{A}}_{\mathsf{refund}}$ at $\mathsf{BobChain}$ time T^{B}_0, Alice never sends any transaction containing pre_a. Without knowing pre_s, pre_c and pre_a, the miner cannot activate $P^{\mathsf{B}}_{\mathsf{default}}$, $C'^{\mathsf{B}}_{\mathsf{burn}}$, $P^{\mathsf{A}}_{\mathsf{default}}$ and $C'^{\mathsf{A}}_{\mathsf{burn}}$.
  As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, $P^{\mathsf{B}}_{\mathsf{refund}}$, $C'^{\mathsf{B}}_{\mathsf{refund}}$, $P^{\mathsf{A}}_{\mathsf{refund}}$ and $C'^{\mathsf{A}}_{\mathsf{refund}}$ must be activated in polynomial time except with negligible probability, and Bob's utility is 0 since he simply gets all his deposit back.
- *Case 2: Bob enters the execution phase, and Alice sent pre_s before* $\mathsf{BobChain}$ *time T^{B}_1.* In this case, Bob will send pre_s to $P^{\mathsf{A}}_{\mathsf{default}}$ at $\mathsf{BobChain}$ time T^{B}_1 at latest. Moreover, Alice and Bob never send any transaction containing pre_b and pre_a. Without knowing pre_b and pre_a, the miner cannot activate $P^{\mathsf{B}}_{\mathsf{refund}}$, $P^{\mathsf{A}}_{\mathsf{refund}}$, $C'^{\mathsf{B}}_{\mathsf{burn}}$ and $C'^{\mathsf{A}}_{\mathsf{burn}}$. If $P^{\mathsf{B}}_{\mathsf{refund}}$ and $P^{\mathsf{A}}_{\mathsf{refund}}$ are not activated, $C'^{\mathsf{B}}_{\mathsf{refund}}$ and $C'^{\mathsf{A}}_{\mathsf{refund}}$ cannot be activated either.
  As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, $P^{\mathsf{B}}_{\mathsf{default}}$ and $P^{\mathsf{A}}_{\mathsf{default}}$ must be activated in polynomial time except with negligible probability, and Bob's utility is $\$\mathsf{BV}(\ddot{\mathbb{A}}x_a - \ddot{\mathbb{B}}x_b) > 0$.
- *Case 3: Bob enters the execution phase, while Alice drops out before sending pre_s.* In this case, Bob will send pre_b to $P^{\mathsf{B}}_{\mathsf{refund}}$ at $\mathsf{BobChain}$ time T^{B}_1. When τ^{B} $\mathsf{BobChain}$ time has passed since $P^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C'^{\mathsf{B}}_{\mathsf{refund}}$. As soon as $C'^{\mathsf{B}}_{\mathsf{refund}}$ is activated, Bob sends ping to $P^{\mathsf{A}}_{\mathsf{refund}}$. When τ^{A} $\mathsf{AliceChain}$ time has passed since $P^{\mathsf{A}}_{\mathsf{refund}}$ is activated, Bob sends ping to $C'^{\mathsf{A}}_{\mathsf{refund}}$. Without knowing pre_s and pre_a, the miner cannot activate $P^{\mathsf{B}}_{\mathsf{default}}$, $C'^{\mathsf{B}}_{\mathsf{burn}}$, $P^{\mathsf{A}}_{\mathsf{default}}$ and $C'^{\mathsf{A}}_{\mathsf{burn}}$.
  As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, $P^{\mathsf{B}}_{\mathsf{refund}}$, $C'^{\mathsf{B}}_{\mathsf{refund}}$, $P^{\mathsf{A}}_{\mathsf{refund}}$ and $C'^{\mathsf{A}}_{\mathsf{refund}}$ must be activated in polynomial time except with negli-

gible probability, and Bob's utility is 0 since he simply gets all his deposit back.

Next, we analyze the case where Bob drops out. There are two cases.

- *Case 1: Alice enters the abort phase.* If Alice enters the abort phase, Bob must drop out before $\mathsf{BobChain}$ time T_0^{B}, so Bob has not sent pre_b to $P_{\mathsf{refund}}^{\mathsf{B}}$. Then, Alice will send ping to $P_{\mathsf{refund}}^{\mathsf{B}}$ at $\mathsf{BobChain}$ time T_0^{B}, and pre_a to $P_{\mathsf{refund}}^{\mathsf{A}}$ at $\mathsf{BobChain}$ time T_1^{B}. When τ^{B} $\mathsf{BobChain}$ time has passed since $P_{\mathsf{refund}}^{\mathsf{B}}$ is activated, Alice sends ping to $C_{\mathsf{refund}}^{\mathsf{B}}$; when τ^{A} $\mathsf{AliceChain}$ time has passed since $P_{\mathsf{refund}}^{\mathsf{A}}$ is activated, Alice sends ping to $C_{\mathsf{refund}}^{\mathsf{A}}$. If Alice enters the abort phase, she never sends any transaction containing pre_s. Without knowing pre_s and pre_b, the miner cannot activate $P_{\mathsf{default}}^{\mathsf{B}}$, $C_{\mathsf{burn}}^{\mathsf{B}}$, $P_{\mathsf{default}}^{\mathsf{A}}$ and $C_{\mathsf{burn}}^{\mathsf{A}}$. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, $P_{\mathsf{refund}}^{\mathsf{B}}$, $C_{\mathsf{refund}}^{\mathsf{B}}$, $P_{\mathsf{refund}}^{\mathsf{A}}$ and $C_{\mathsf{refund}}^{\mathsf{A}}$ must be activated in polynomial time except with negligible probability, and Alice's utility is 0 since she simply gets all her deposit back.
- *Case 2: Alice enters the execution phase.* In this case, Bob must have sent pre_c to $P_{\mathsf{default}}^{\mathsf{B}}$. Alice will send pre_s to $P_{\mathsf{default}}^{\mathsf{B}}$ before $\mathsf{BobChain}$ time T_1^{B}, and thus Bob never sends any transaction containing pre_b. As soon as $P_{\mathsf{default}}^{\mathsf{B}}$ is activated, she will send ping to $P_{\mathsf{default}}^{\mathsf{A}}$. In the execution phase, Alice never sends any transaction containing pre_a. Without knowing pre_b and pre_a, the miner cannot activate $P_{\mathsf{refund}}^{\mathsf{B}}$, $P_{\mathsf{refund}}^{\mathsf{A}}$, $C_{\mathsf{burn}}^{\mathsf{B}}$ and $C_{\mathsf{burn}}^{\mathsf{A}}$. If $P_{\mathsf{refund}}^{\mathsf{B}}$ and $P_{\mathsf{refund}}^{\mathsf{A}}$ are not activated, $C_{\mathsf{refund}}^{\mathsf{B}}$ and $C_{\mathsf{refund}}^{\mathsf{A}}$ cannot be activated either.
  As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, $P_{\mathsf{default}}^{\mathsf{B}}$ and $P_{\mathsf{default}}^{\mathsf{A}}$ must be activated in polynomial time except with negligible probability, and Alice's utility is $\$\mathsf{AV}(\ddot{\mathrm{B}}x_b - \ddot{\mathrm{A}}x_a) > 0$.

F Bitcoin Instantiation

As described earlier, with general smart contracts, users send coins to contracts, the contracts then hold the coins until some logic is triggered to pay part to all of the coins to one or more user(s). Instead, Bitcoin uses an *Unspent Transaction Output* (UTXO) model, where coins are stored in addresses denoted by $Adr \in \{0,1\}^\lambda$ and addresses are spendable (i.e., used as input to a transaction) *exactly once*. Transactions can be posted on the blockchain to transfer coins from a set of input addresses to a set of output addresses, and any remaining amount of coin is collected by the miner of the block as transaction fee.

More precisely, in Bitcoin transactions are generated by the transaction function tx. A transaction tx_A, denoted

$$tx_A := tx\left(\begin{matrix}[(Adr_1, \Phi_1, \$v_1), \ldots, (Adr_n, \Phi_n, \$v_n)],\\ [(Adr'_1, \Phi'_1, \$v'_1), \ldots, (Adr'_m, \Phi'_m, \$v'_m)]\end{matrix}\right),$$

charges v_i coins from each input address Adr_i for $i \in [n]$, and pays v'_i coins to each output address Adr'_j where $j \in [m]$. It must be guaranteed that $\sum_{i\in[n]} \$v_i \geq$

$\sum_{j\in[m]} \$v'_j$. The difference $\$f = \sum_{i\in[n]} \$v_i - \sum_{j\in[m]} \$v'_j$ is offered as the transaction fee to the miner who includes the transaction in his block.

An address in Bitcoin is typically associated with a *script* $\Phi : \{0,1\}^\lambda \to \{0,1\}$ which states what conditions need to be satisfied for the coins to be spent from the address. In contrast to smart contracts that can verify arbitrary conditions for coins to be transacted, the scripting language of Bitcoin has limited expressiveness. A transaction is considered authorized if it is attached with witnesses $[x_1, \ldots, x_n]$ such that $\Phi_i(x_i) = 1$ (publicly computable) for all $i \in [n]$. A transaction is confirmed if it is included in the blockchain.

Thus, for Bitcoin, the logic of our contracts must be encoded in scripts of addresses where the scripts are of limited expressiveness and the addresses are spendable exactly once. As we show, our instantiations only require some of the most standard scripts used currently in Bitcoin.

We largely rely on the following scripts: (1) computation of hash function[11] $H : \{0,1\}^* \to \{0,1\}^\kappa$, (2) transaction timestamp verification wrt. current height of the blockchain denoted by _NOW[12] and (3) digital signature verification. The signature scheme consists of the key generation algorithm $\mathsf{KGen}(1^\lambda)$ generating the signing key sk and the verification key pk, the signing algorithm $\mathsf{Sign}(\mathsf{sk}, m)$ generating a signature σ on the message m using sk, and the verification algorithm $\mathsf{Vf}(\mathsf{pk}, m, \sigma)$ validating the signature wrt. pk. [13] We say an address Adr (associated script Φ) is controlled by a user if the user knows a witness x s.t. $\Phi(x) = 1$.

F.1 Instantiating RapidashKC

We provide the list of all transactions in Table 6, the scripts associated with all addresses in Figure 2, and the relationship between the transactions, addresses, and scripts is depicted in Figure 3. Basically, Bob uses the transaction tx_{stp} to put his deposit $\$v + \$c_b$ into the address Adr_{stp}. The script on the address Adr_{stp} allows three ways to spend the deposit:

1. Use pre_s to pay $\$v$ amount to an address Adr_1^A owned by Alice, and $\$c_b$ to an address Adr_1^B owned by Bob.
2. After a timeout T_1 since the address Adr_{stp} comes into existence, use pre_b to pay the entire deposit amount $\$v + \$c_b$ to the address $Adr_{P_{\mathsf{refund}}}$, which is associated with the script $\Phi_{P_{\mathsf{refund}}}$. $\Phi_{P_{\mathsf{refund}}}$ says that either (1) T_2 time has passed after the address came into existence, in which case the $\$v + \$c_b$ coins

[11] $\kappa = 160$ in Bitcoin when using the opcode OP_HASH160.

[12] Instantiated using the opcode OP_CHECKSEQUENCEVERIFY in Bitcoin checking if the height of the blockchain has increased beyond some threshold after the script first appeared on the blockchain. It can also be instantiated with opcode OP_CHECKLOCKTIMEVERIFY in Bitcoin that checks if the current height of the blockchain is beyond a threshold.

[13] The signature scheme can be instantiated with either Schnorr or ECDSA in Bitcoin. ECDSA signatures are verified using the opcode OP_CHECKSIG and Schnorr signatures via the taproot fork.

Table 6: RAPIDASHKC's transactions in Bitcoin. Here Φ^B is the script that requires the signature under Bob's public key while Φ^A is the script that requires the signature under Alice's public key.

	Description
tx_{stp}	$tx\begin{pmatrix}[(Adr_0^B, \Phi^B, \$v + \$c_b)],\\ [(Adr_{\mathsf{stp}}, \Phi_{\mathsf{stp}}, \$v + \$c_b)]\end{pmatrix}$
$tx_{P_{\mathsf{default}}}$	$tx\begin{pmatrix}[(Adr_{\mathsf{stp}}, \Phi_{\mathsf{stp}}, \$v + \$c_b)],\\ [(Adr_1^A, \Phi^A, \$v), (Adr_1^B, \Phi^B, \$c_b)]\end{pmatrix}$
$tx_{P_{\mathsf{refund}}}$	$tx\begin{pmatrix}[(Adr_{\mathsf{stp}}, \Phi_{\mathsf{stp}}, \$v + \$c_b),\\ [(Adr_{P_{\mathsf{refund}}}, \Phi_{P_{\mathsf{refund}}}, \$v + \$c_b)]\end{pmatrix}$
$tx_{C_{\mathsf{refund}}}$	$tx\begin{pmatrix}[(Adr_{P_{\mathsf{refund}}}, \Phi_{P_{\mathsf{refund}}}, \$v + \$c_b)\\ [(Adr_2^B, \Phi^B, \$v + \$c_b)]\end{pmatrix}$
$tx_{C_{\mathsf{burn}}}$	$tx\begin{pmatrix}[(Adr_{\mathsf{stp}}, \Phi_{\mathsf{stp}}, \$v + \$c_b)],\\ [(Adr_{\mathsf{burn}}, \Phi_{\mathsf{burn}}, \$v + \$c_b - \$\epsilon)]\end{pmatrix}$
$tx_{C_{\mathsf{burn}}}^{P_{\mathsf{refund}}}$	$tx\begin{pmatrix}[(Adr_{P_{\mathsf{refund}}}, \Phi_{P_{\mathsf{refund}}}, \$v + \$c_b)\\ [(Adr_{\mathsf{burn}}, \Phi_{\mathsf{burn}}, \$v + \$c_b - \$\epsilon)]\end{pmatrix}$

in $Adr_{P_{\mathsf{refund}}}$ can be paid to Bob's address Adr_2^B, or (2) the pair (pre_s, pre_b) is revealed, in which case $\$v + \$c_b - \$\epsilon$ coins can be paid to some burn address Adr_{burn} whose private key is known to nobody, and the remaining $\$\epsilon$ is paid as fee to the miner who mines the block.

3. Use the pair (pre_s, pre_b) to pay $\$v + \$c_b - \$\epsilon$ amount to some burn address Adr_{burn} whose private key is known to nobody, the remaining $\$\epsilon$ is paid as fee to the miner who mines the block.

To make sure that Alice and Bob cannot unilaterally spend from the address Adr_{stp}, and $Adr_{P_{\mathsf{refund}}}$, their associated scripts require *signatures from both Alice and Bob* to spend from these addresses. Note also that the transactions $tx_{P_{\mathsf{default}}}$, $tx_{P_{\mathsf{refund}}}$, and $tx_{C_{\mathsf{burn}}}$ needed to spend from Adr_{stp} via activation points P_{default}, P_{refund}, or C_{burn} are signed with *different* public keys of Alice and Bob for each activation point, i.e., $(\mathsf{pk}_a, \mathsf{pk}_b)$, $(\mathsf{pk}'_a, \mathsf{pk}'_b)$, and $(\mathsf{pk}''_a, \mathsf{pk}''_b)$ respectively. This makes sure that each transaction can invoke only the intended activation point. Similarly for transactions $tx_{C_{\mathsf{burn}}}$ and $tx_{C_{\mathsf{burn}}}^{P_{\mathsf{refund}}}$ spending from $Adr_{P_{\mathsf{refund}}}$.

Protocol flow. Before setting up RAPIDASHKC on the blockchain, Alice and Bob agree on the setup transaction tx_{stp}. The transaction must be signed by Bob to take effect. However, before signing tx_{stp}, Alice and Bob agree on and sign all redeeming transactions, i.e., $tx_{P_{\mathsf{default}}}$, $tx_{P_{\mathsf{refund}}}$, $tx_{C_{\mathsf{refund}}}$,$tx_{C_{\mathsf{burn}}}^{P_{\mathsf{refund}}}$, and $tx_{C_{\mathsf{burn}}}$. Alice and Bob now broadcast all these transactions (not including tx_{stp}) and both of their signatures — notice that they cannot be published on the Bitcoin blockchain yet because the addresses they depend on, Adr_{stp} or $Adr_{P_{\mathsf{refund}}}$, are not ready yet.

$\Phi_{\mathsf{stp}}(tx, pre_s, pre_b, \sigma_a, \sigma_b)$

P_{default} : **if** $(H(pre_s) = h_s) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
then return 1

P_{refund} : **if** $(\_\mathrm{NOW} > T_1) \wedge (H(pre_b) = h_b) \wedge (\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge$
$(\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1)$ **then return** 1

C_{burn} : **if** $(\mathsf{Vf}(\mathsf{pk}''_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}''_b, tx, \sigma_b) = 1) \wedge (H(pre_s) = h_s) \wedge$
$(H(pre_b) = h_b)$ **then return** 1

// Values $h_s, h_b, \mathsf{pk}_a, \mathsf{pk}_b, T_1, \mathsf{pk}'_a, \mathsf{pk}'_b, \mathsf{pk}''_a, \mathsf{pk}''_b$ are hardwired

$\Phi_{P_{\mathsf{refund}}}(tx, pre_s, pre_b, \sigma_a, \sigma_b)$

C_{refund} : **if** $(\_\mathrm{NOW} > T_2) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
then return 1

C_{burn} : **if** $(\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1) \wedge (H(pre_s) = h_s)$
$\wedge (H(pre_b) = h_b)$ **then return** 1

// Values $T_2, h_s, h_b, \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired

Fig. 2: The description of scripts Φ_{stp} and $\Phi_{P_{\mathsf{refund}}}$. Here tx is the transaction spending from the script. Keys $\mathsf{pk}_a, \mathsf{pk}'_a$ and pk''_a belong to Alice, $\mathsf{pk}_b, \mathsf{pk}'_b$ and pk''_b belong to Bob.

At this moment, Bob reveals his signature on tx_{stp}. Once tx_{stp} is published on the Bitcoin blockchain, the *execution phase* starts. During the execution phase, either Alice reveals pre_s and publishes transaction $tx_{P_{\mathsf{default}}}$ (along with signatures on the transaction), or Bob reveals pre_b and publishes transaction $tx_{P_{\mathsf{refund}}}$ (along with signatures on the transaction) after T_1 time has passed since the confirmation of tx_{stp}. In the honest run of the protocol, if $tx_{P_{\mathsf{default}}}$ is confirmed, Bob gets back his collateral immediately. If not, Bob can redeem the collateral after waiting for time $T_1 + T_2$ using $tx_{P_{\mathsf{refund}}}$ and $tx_{C_{\mathsf{refund}}}$. In the event of misbehavior leading to both pre_s and pre_b being revealed, any miner in the system can immediately spend from the C_{burn} branch of either Adr_{stp}, or $Adr_{P_{\mathsf{refund}}}$, and burn all coins except $\$\epsilon$ coins as transaction fee for itself.

F.2 Instantiating Rapidash$^{\mathrm{B}}$ with CSP Fairness

We have minor differences compared to the single instance instantiation.

Transactions. We describe below the different transactions needed for our Rapidash$^{\mathrm{B}}$ instantiation.

– We now have an additional payment redeem transaction, $tx^{\mathsf{ping}}_{P^{\mathrm{B}}_{\mathsf{refund}}}$ (see Table 7) apart from $tx_{P^{\mathrm{B}}_{\mathsf{default}}}$ and $tx_{P^{\mathrm{B}}_{\mathsf{refund}}}$ that redeem from the payment address $Adr^{\mathrm{B}}_{\mathsf{stp}}$.

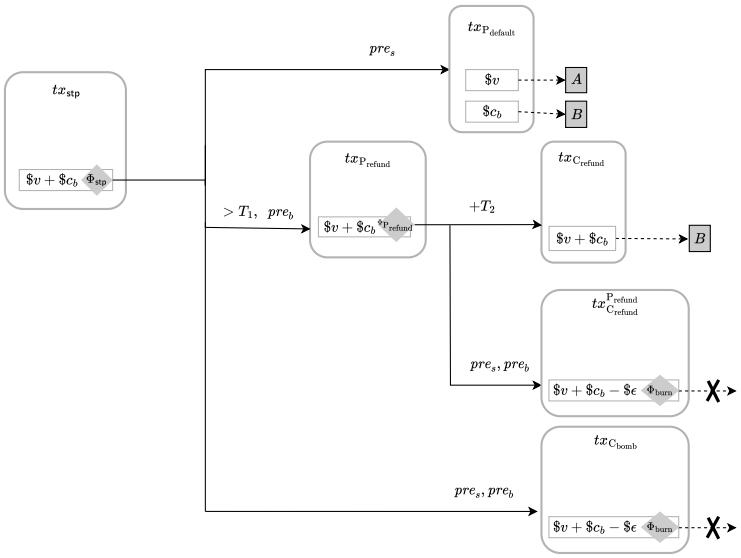

Fig. 3: The transaction flow of RAPIDASHKC in Bitcoin absent external incentives. Rounded boxes denote transactions, rectangles within are outputs of the transaction. Incoming arrows denote transaction inputs, outgoing arrows denote how an output can be spent by a transaction at the end of the arrow. Solid lines indicate the transaction output can be spent only if both users sign the spending transaction. Dashed arrows indicate that the output can be spent by one user (A for Alice, and B for Bob). The timelock (T_1 and T_2) associated with a transaction output is written over the corresponding outgoing arrow.

We have the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$ that redeems $\$x_b + \$c^{\mathsf{B}}_b$ coins to an auxiliary address $Adr_{P^{\mathsf{B}}_{\mathsf{refund}}}$. The description of $\Phi^{\mathrm{B}}_{\mathsf{stp}}$ is given below in Figure 4. We set the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$ to redeem the coins from the (E^{B}_2) branch. This transaction will correspond to the empty message call to the RAPIDASH$^{\mathrm{B}}$ contract in activation point $P^{\mathsf{B}}_{\mathsf{refund}}$. The script $\Phi^{\mathrm{B}}_{\mathsf{stp}}$ has a modification in the $C^{\mathsf{B}}_{\mathsf{burn}}$ branch, where we require either (pre_s, pre_b, pre_c) along with the signatures of Alice and Bob. Similarly, the script $\Phi_{P^{\mathsf{B}}_{\mathsf{refund}}}$ of the auxiliary addresses is modified in its $C^{\mathsf{B}}_{\mathsf{burn}}$ branch.

– In addition to the collateral redeem transaction $tx_{C^{\mathsf{B}}_{\mathsf{refund}}}$, we have the transaction $tx^{\mathsf{ping}}_{C^{\mathsf{B}}_{\mathsf{refund}}}$ which redeems the coins to Bob from the auxiliary address generated by $tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$. We have modified transactions $tx^{P^{\mathsf{B}}_{\mathsf{refund}}}_{C^{\mathsf{B}}_{\mathsf{burn}}}$ and $tx_{C^{\mathsf{B}}_{\mathsf{burn}}}$ which can be redeemed only if pre_s, pre_b and pre_c are revealed, such that $H(pre_s) = h_s, H(pre_b) = h_b$, and $H(pre_c) = h_c$. We have an additional transaction $tx^{P^{\mathsf{B}}_{\mathsf{refund}},\mathsf{ping}}_{C^{\mathsf{B}}_{\mathsf{burn}}}$ that redeems the coins from the auxiliary address of $tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$ if

pre_s, pre_b and pre_c are revealed. Unlike the single instance, here we replace T_2 with τ^{B}. A pictorial description of the transaction flow is described in Figure 5.

Table 7: Description of additional transactions in Bitcoin for RAPIDASH atomic swap with CSP fairness. Here Φ^B is a script that requires a signature from Bob's public key, respectively.

	Description
$tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$	$tx\left(\begin{matrix}[(Adr^{\mathrm{B}}_{\mathsf{stp}}, \Phi^{\mathrm{B}}_{\mathsf{stp}}, \$x_b + \$c^{\mathsf{B}}_b)],\\ [(Adr_{P^{\mathsf{B}}_{\mathsf{refund}}}, \Phi_{P^{\mathsf{B}}_{\mathsf{refund}}}, \$x_b + \$c^{\mathsf{B}}_b)]\end{matrix}\right)$
$tx^{\mathsf{ping}}_{C^{\mathsf{B}}_{\mathsf{refund}}}$	$tx\left(\begin{matrix}[(Adr_{P^{\mathsf{B}}_{\mathsf{refund}}}, \Phi_{P^{\mathsf{B}}_{\mathsf{refund}}}, \$x_b + \$c^{\mathsf{B}}_b)],\\ [(Adr^{B}_2, \Phi^{B}, \$x_b + \$c^{\mathsf{B}}_b)]\end{matrix}\right)$
$tx^{P^{\mathsf{B}}_{\mathsf{refund}},\mathsf{ping}}_{C^{\mathsf{B}}_{\mathsf{burn}}}$	$tx\left(\begin{matrix}[(Adr_{P^{\mathsf{B}}_{\mathsf{refund}}}, \Phi_{P^{\mathsf{B}}_{\mathsf{refund}}}, \$x_b + \$c^{\mathsf{B}}_b)],\\ [(Adr^{\mathrm{B}}_{\mathsf{burn}}, \Phi^{\mathrm{B}}_{\mathsf{burn}}, \$x_b + \$c^{\mathsf{B}}_b - \$\epsilon^{\mathsf{B}})]\end{matrix}\right)$

Protocol Flow. Alice and bob first agree on the setup transaction $tx^{\mathrm{B}}_{\mathsf{stp}}$ and sign the redeeming transactions $tx_{P^{\mathsf{B}}_{\mathsf{default}}}, tx_{P^{\mathsf{B}}_{\mathsf{refund}}}, tx_{C^{\mathsf{B}}_{\mathsf{refund}}}, tx^{P^{\mathsf{B}}_{\mathsf{refund}}}_{C^{\mathsf{B}}_{\mathsf{burn}}}, tx^{\mathsf{ping}}_{C^{\mathsf{B}}_{\mathsf{refund}}}, tx^{P^{\mathsf{B}}_{\mathsf{refund}},\mathsf{ping}}_{C^{\mathsf{B}}_{\mathsf{burn}}}$ and $tx_{C^{\mathsf{B}}_{\mathsf{burn}}}$ and broadcast all these transactions and the respective signatures, like before. They sign the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$ such that only Alice has both signatures, and she keeps them private for now. Finally, they sign the setup transaction $tx^{\mathrm{B}}_{\mathsf{stp}}$ and publish it on the blockchain, starting the execution phase.

Whenever Alice wishes to activate $P^{\mathsf{B}}_{\mathsf{refund}}$ branch with an empty ping message, she publishes the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$ along with the valid signatures she has in her possession. If $tx^{\mathsf{ping}}_{P^{\mathsf{B}}_{\mathsf{refund}}}$ is published on the blockchain, activation point $C^{\mathsf{B}}_{\mathsf{refund}}$ can be activated by $tx^{\mathsf{ping}}_{C^{\mathsf{B}}_{\mathsf{refund}}}$ after a timeout of τ^{B} time units. The rest of the protocol proceeds exactly as the description of the swap protocol.

F.3 Instantiating RAPIDASH$^{\mathrm{A}}$ with CSP Fairness

We describe all the transactions, addresses, and scripts needed in the RAPIDASH$^{\mathrm{A}}$ instantiation for the atomic swap. Notice that the roles of Alice and Bob are reversed compared to RAPIDASH$^{\mathrm{B}}$ above. Specifically, in RAPIDASH$^{\mathrm{A}}$, Bob can use pre_s to retrieve the coins from the payment address, while Alice can use pre_a after a timeout of T^{A}_1 to retrieve the coins. The main difference between this instantiation and the RAPIDASH$^{\mathrm{B}}$ instantiation above is that in the execution phase both the payment address activation points $P^{\mathsf{A}}_{\mathsf{default}}$ and $P^{\mathsf{A}}_{\mathsf{refund}}$ can be activated by empty message calls. We also have modified collateral redeeming transactions that redeem the coins from the $C^{\mathsf{A}}_{\mathsf{burn}}$ branch of the $\Phi^{\mathrm{A}}_{\mathsf{stp}}$.

$\Phi^{\mathsf{B}}_{\mathsf{stp}}(tx, pre_s, pre_b, \sigma_a, \sigma_b)$

$P^{\mathsf{B}}_{\mathsf{default}}$: **if** $(H(pre_s) = h_s) \wedge (H(pre_c) = h_c) \wedge$
$(\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \ \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
then return 1

$P^{\mathsf{B}}_{\mathsf{refund}}$: **if** $(\_\mathrm{NOW} > T^{\mathsf{B}}_1) \wedge (H(pre_b) = h_b)$
$\wedge (\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1)$
then return 1

$\mathrm{E}^{\mathsf{B}}_2$: **if** $(\_\mathrm{NOW} > T^{\mathsf{B}}_1) \wedge (\mathsf{Vf}(\mathsf{pk}^3_a, tx, \sigma_b) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}^3_b, tx, \sigma_b) = 1)$
then return 1

$C^{\mathsf{B}}_{\mathsf{burn}}$ **if** $(\mathsf{Vf}(\mathsf{pk}''_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}''_b, tx, \sigma_b) = 1) \wedge$
$\Big((H(pre_s) = h_s) \wedge (H(pre_b) = h_b) \wedge (H(pre_c) = h_c)\Big)$
then return 1

// Values $h_s, h_b, h_c, \mathsf{pk}_a, \mathsf{pk}_b, T^{\mathsf{B}}_1, \mathsf{pk}'_a, \mathsf{pk}'_b, \mathsf{pk}''_a, \mathsf{pk}''_b, \mathsf{pk}^3_a, \mathsf{pk}^3_b$ are hardwired

$\Phi_{P^{\mathsf{B}}_{\mathsf{refund}}}(tx, pre_s, pre_b, pre_c, \sigma_a, \sigma_b)$

$C^{\mathsf{B}}_{\mathsf{refund}}$: **if** $(\_\mathrm{NOW} > \tau^{\mathsf{B}}) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
then return 1

$C^{\mathsf{B}}_{\mathsf{burn}}$: **if** $(\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1) \wedge$
$\Big((H(pre_s) = h_s) \wedge (H(pre_b) = h_b) \wedge (H(pre_c) = h_c)\Big)$
then return 1

// Values $\tau^{\mathsf{B}}, h_s, h_b, h_c, \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired

Fig. 4: The description of script $\Phi^{\mathsf{B}}_{\mathsf{stp}}$ and $\Phi_{P^{\mathsf{B}}_{\mathsf{refund}}}$ for atomic swap with CSP fairness. Here tx is the transaction spending from the script. Keys $(\mathsf{pk}_a, \mathsf{pk}'_a, \mathsf{pk}''_a, \mathsf{pk}^3_a)$ and $(\mathsf{pk}_b, \mathsf{pk}'_b, \mathsf{pk}''_b, \mathsf{pk}^3_b)$ belong to Alice and Bob, respectively.

Table 8: Description of additional transaction in Bitcoin for RAPIDASH$^{\mathrm{A}}$ atomic swap with CSP fariness. Here Φ^A and Φ^B are scripts that require a signature from Alice's and Bob's public key, respectively.

	Description
$tx^{\mathsf{ping}}_{P^{\mathsf{A}}_{\mathsf{default}}}$	$tx\left(\begin{matrix}[(Adr^{\mathsf{A}}_{\mathsf{stp}}, \Phi^{\mathsf{A}}_{\mathsf{stp}}, \$x_a + \$c^{\mathsf{A}}_a + \$c^{\mathsf{A}}_b)], \\ [(Adr^A_1, \Phi^A, \$c^{\mathsf{A}}_a), (Adr^B_1, \Phi^B, \$x_a + \$c^{\mathsf{A}}_b)]\end{matrix}\right)$

Transactions. We describe below the different transactions needed for our RAPIDASH$^{\mathrm{A}}$ instantiation. We have the same set of transactions that are analog of the RAPIDASH$^{\mathrm{B}}$ instantiation, except for one additional transaction $tx^{\mathsf{ping}}_{P^{\mathsf{A}}_{\mathsf{default}}}$ (see Table 8). The transaction redeems the coins from the payment address

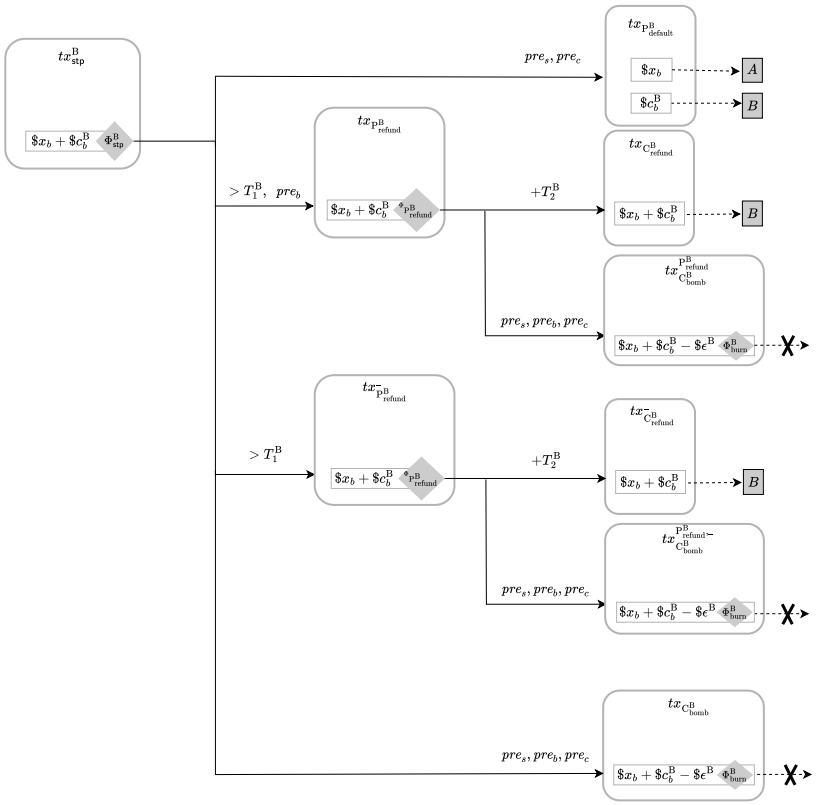

Fig. 5: The transaction flow of RAPIDASH^{B} in Bitcoin for atomic swap with CSP fairness. Rounded boxes denote transactions, rectangles within are outputs of the transaction. Incoming arrows denote transaction inputs, outgoing arrows denote how an output can be spent by a transaction at the end of the arrow. Solid lines indicate the transaction output can be spent only if both users sign the spending transaction. Dashed arrows indicate that the output can be spent by one user (A for Alice, and B for Bob).

Adr^{A}_{stp} using the (E^{A}_{1}) branch of Φ^{A}_{stp}. The description of Φ^{A}_{stp} is given below in Figure 6 with Alice and Bob's roles being reversed in RAPIDASH^{A}. This transaction will correspond to the empty message call to RAPIDASH^{A} activation point P^{A}_{default}. The script Φ^{A}_{stp} (and correspondingly $\Phi_{P^{A}_{\mathsf{refund}}}$) has a modification in the C^{A}_{burn} branch, where we require either (pre_s, pre_a) or (pre_a, pre_b) along with the signatures of Alice and Bob. We have the corresponding redeeming transactions as $tx^{P^{A}_{\mathsf{refund}}}_{C^{A}_{\mathsf{burn}}}$, $tx^{P^{A}_{\mathsf{refund}},\mathsf{ping}}_{C^{A}_{\mathsf{burn}}}$ and $tx_{C^{A}_{\mathsf{burn}}}$ similar to RAPIDASH^{B}. A pictorial description of the transaction flow for payment and collateral redeem is given in Figure 7.

Protocol Flow. Alice and Bob, first agree on the setup transaction tx^{A}_{stp} and sign the redeeming transactions. They broadcast all these transactions and the

$\Phi^{\mathsf{A}}_{\mathsf{stp}}(tx, pre_s, pre_a, pre_b, \sigma_a, \sigma_b)$

$P^{\mathsf{A}}_{\mathsf{default}}$: **if** $(H(pre_s) = h_s) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1)$
$\wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$ **then return** 1

$P^{\mathsf{A}}_{\mathsf{refund}}$: **if** $(\_\mathrm{NOW} > T_1^{\mathsf{A}}) \wedge (H(pre_a) = h_a)$
$\wedge (\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1)$
then return 1

$\mathrm{E}^{\mathrm{A}}_1$: **if** $(\mathsf{Vf}(\mathsf{pk}''_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}''_b, tx, \sigma_b) = 1)$
then return 1

$\mathrm{E}^{\mathrm{A}}_2$: **if** $(\_\mathrm{NOW} > T_1^{\mathsf{A}}) \wedge (\mathsf{Vf}(\mathsf{pk}^3_a, tx, \sigma_b) = 1) \wedge$
$(\mathsf{Vf}(\mathsf{pk}^3_b, tx, \sigma_b) = 1)$ **then return** 1

$C^{\mathsf{A}}_{\mathsf{burn}}$: **if** $(\mathsf{Vf}(\mathsf{pk}^4_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}^4_b, tx, \sigma_b) = 1) \wedge$

$$\Big(\big((H(pre_s) = h_s) \wedge (H(pre_a) = h_a)\big) \vee \big((H(pre_a) = h_a) \wedge (H(pre_b) = h_b)\big)\Big)$$

then return 1

// Values $h_s, h_a, h_b, \mathsf{pk}_a, \mathsf{pk}_b, T_1^{\mathsf{A}}, \mathsf{pk}'_a, \mathsf{pk}'_b, \mathsf{pk}''_a, \mathsf{pk}''_b, \mathsf{pk}^3_a, \mathsf{pk}^3_b, \mathsf{pk}^4_a, \mathsf{pk}^4_b$ are hardwired

$\Phi_{P^{\mathsf{A}}_{\mathsf{refund}}}(tx, pre_s, pre_a, pre_b, \sigma_a, \sigma_b)$

$C^{\mathsf{A}}_{\mathsf{refund}}$: **if** $(\_\mathrm{NOW} > \tau^{\mathsf{A}}) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
then return 1

$C^{\mathsf{A}}_{\mathsf{burn}}$: **if** $(\mathsf{Vf}(\mathsf{pk}^4_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}^4_b, tx, \sigma_b) = 1) \wedge$

$$\Big(\big((H(pre_s) = h_s) \wedge (H(pre_a) = h_a)\big) \vee \big((H(pre_a) = h_a) \wedge (H(pre_b) = h_b)\big)\Big)$$

then return 1

// Values $\tau^{\mathsf{A}}, h_s, h_a, h_b, \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired

Fig. 6: The description of script $\Phi^{\mathrm{A}}_{\mathsf{stp}}$ for RAPIDASH$^{\mathrm{A}}$ in atomic swap with CSP fairness.

respective signatures, like before. However, this time Alice and Bob sign the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{A}}_{\mathsf{default}}}$ such that only Alice has both signatures. She does not broadcast the signatures and keeps them private. Similarly, Alice and Bob sign the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{A}}_{\mathsf{refund}}}$ such that only Bob has both signatures. He keeps them private and does not broadcast them. Notice that none of the transactions can be published on the blockchain yet as the setup transaction is not yet published. Finally, they sign the setup transaction $tx^{\mathrm{A}}_{\mathsf{stp}}$ and publish it on the blockchain, thus starting the execution phase.

Whenever Alice wishes to activate $P^{\mathsf{A}}_{\mathsf{default}}$ in RAPIDASH$^{\mathrm{A}}$ with an empty message, she publishes the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{A}}_{\mathsf{default}}}$ along with the valid signatures she has in her possession. Similarly, whenever Bob wishes to activate $P^{\mathsf{A}}_{\mathsf{refund}}$ in

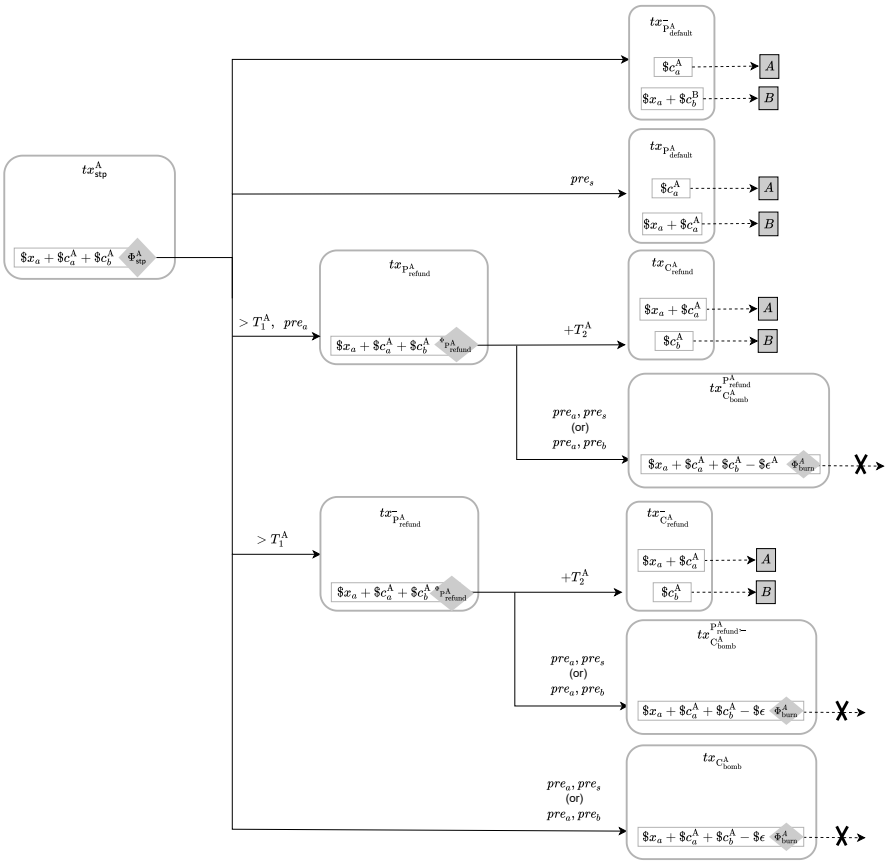

Fig. 7: The transaction flow of RAPIDASH$^{\mathrm{A}}$ in Bitcoin for atomic swap with CSP fairness. Rounded boxes denote transactions, and rectangles within are outputs of the transaction. Incoming arrows denote transaction inputs, outgoing arrows denote how an output can be spent by a transaction at the end of the arrow. Solid lines indicate the transaction output can be spent only if both users sign the spending transaction. Dashed arrows indicate that the output can be spent by one user (A for Alice, and B for Bob). The timelock (T^{A}_1 and τ^{A}) associated with a transaction output is written over the corresponding outgoing arrow.

RAPIDASH$^{\mathrm{A}}$ with an empty message, he publishes the transaction $tx^{\mathsf{ping}}_{P^{\mathsf{A}}_{\mathsf{refund}}}$ along with the valid signatures he has in his possession. If $tx^{\mathsf{ping}}_{P^{\mathsf{A}}_{\mathsf{refund}}}$ is published on the blockchain, activation point $C^{\mathsf{A}}_{\mathsf{refund}}$ can be activated by $tx^{\mathsf{ping}}_{C^{\mathsf{A}}_{\mathsf{refund}}}$ after a timeout of τ^{A} time units. Rest of the flow follows exactly the description of the atomic swap protocol.

G Bug in the Bitcoin Evaluation of He-HTLC [21].

We note that there is a bug in the Bitcoin evaluation of the concurrent He-HTLC, a brief overview of which we provide below.

Intuitively, the given implementation of He-HTLC uses Bitcoin scripts with if-else branches, where all branches specify the same public keys for the spending transaction. This results in a mix-and-match attack on the spending transactions as described below. Consider the script specifying two branches, both of which require the corresponding spending transaction to be signed by two public keys pk_A (of Alice) and pk_B (of Bob). Additionally, branch 1 requires a secret value x_1, and branch 2 requires a secret value x_2. Take RAPIDASHKC as an example, branch 1 is P_{default} and branch 2 is P_{refund}. Logically, whenever P_{default} is activated, the payment should go to Alice. However, because of an implementation-level bug below, Bob can trigger P_{default} but redirect the payment to himself. Let Alice and Bob (as they do in He-HTLC) pre-sign transaction tx_1 meant to invoke branch 1 of the script and redeem coins to Alice, and pre-sign transaction tx_2 meant to invoke branch 2 and redeem coins to Bob. Now, whenever Alice posts tx_1 with the signatures and the secret x_1 in the network, the malicious Bob can simultaneously post tx_2 with the signatures and the secret x_1. If Bob's transaction succeeds ahead of Alice's, Bob can redeem the coins to himself when Alice's secret x_1 is revealed. Since tx_1 and tx_2 can be verified by the same public key pair $(\mathsf{pk}_A, \mathsf{pk}_B)$, Bob can use Alice's secret x_1 and transaction tx_2 to trigger branch 1. We emphasize that this is an implementation-level issue, and their pseudocode does not suffer from the attack.

The issue is resolved if the branches require signatures on different pairs of public keys, namely, $(\mathsf{pk}_A^1, \mathsf{pk}_B^1)$ for branch 1, and $(\mathsf{pk}_A^2, \mathsf{pk}_B^2)$. We adopt this approach in our evaluation which is also the standard mechanism for branched scripts used in Bitcoin Lightning Network.