RAPIDASH: Improved Constructions for Side-Contract-Resilient Fair Exchange

Hao Chung Elisaweta Masserova Elaine Shi
Sri AravindaKrishnan Thyagarajan

Carnegie Mellon University

Abstract

The recent work of Chung et al. suggested new formal foundations for side-contract-resilient fair exchange protocols. In this paper, we suggest new constructions that achieve a coalition-resistant Nash equilibrium, and moreover, our constructions improve over Chung et al. in the following senses: 1) we achieve optimistic reponsiveness; and 2) we reduce the amount of collateral from the parties.

1 Introduction

Fair exchange protocols [BBSU12, Her18, MMS$^+$, vdM19, MD19, Max, CGGN, BDM, Fuc, BK, MES16, MMA, Bis, ZHL$^+$19, JMM14, TYME21, PD] have been widely adopted in cryptocurrency systems, in the form of atomic swaps [Her18, MMS$^+$, vdM19, MD19], contingent payment [Max, CGGN, BDM, Fuc, BK], payment channels [PD, DW15, GM, MMSH, MBB$^+$, DFH18, DEFM19], or vaults [MES16, MMA, Bis, ZHL$^+$19]. Recently, the community are increasingly concerned that potential user-miner collusion can completely break the fairness guarantees promised by fair exchange protocols [TYME21, WHF19, Bon, MMS$^+$, MHM18, JSZ$^+$21, Ham]. This phenomenon is also commonly referred to as Miner Extractable Value (MEV): since the miner has the power to decide which transactions to include in the block and their relative ordering, a user colluding with a miner may be able to jointly benefit and harm the counterparty.

Very recently, the works of Wadhwa et al. [WSZN22] and Chung et al. [CMST22] were the first to explore side-contract-resilient fair exchange. In particular, with the goal of building a formal foundation for studying side-contract-resilient fair exchange, Chung et al. [CMST22] suggested two game-theoretic fairness notions:

- *Cooperative strategy proofness (CSP-fairness)*: CSP fairness was initially adopted in a line of work at the intersection of game theory and cryptography [PS17a, CGL$^+$18, WAS22, CCWS21, KMSW22]. Informally speaking, a blockchain-based fair exchange protocol between two parties Alice and Bob is said to be CSP-fair, iff an Alice-miner coalition or a Bob-miner coalition cannot increase the coalition's joint gain through any deviation. In other words, the honest strategy is a *coalition-resistant Nash equilibrium*, and is the coalition's best response assuming no external incentives.

  In particular, CSP fairness considers the coalition's *joint* payoff. This is a good fit for a blockchain environment since the user and the colluding miner may employ arbitrary smart contract mechanisms to split off their joint gains off the table, and moreover, the payoff division mechanism is binding.

- *Safe participation in the presence of external incentives*: Chung et al. [CMST22] suggest a new notion called "safe participation in the presence of external incentives", aiming to protect the honest player even when the counterparty, possibly colluding with the miners, may have external incentives that incentivize them to behave maliciously in a way that may lead to losses in the present protocol. Essentially, the external incentives may cover for the loss in the present protocol. In particular, in Chung et al. [CMST22]'s protocol, an honest player's utility is guaranteed to be non-negative, even when the other player (possibly colluding with some miners) may have *arbitrary but bounded* external incentives.

The two game-theoretic notions above are both desirable but incomparable in nature. Chung et al. [CMST22] aimed to achieve both properties in their fair exchange protocols called Ponyta. However, their approach suffers from a few drawbacks. To understand their drawbacks, consider a cross-chain atomic swap scenario where Alice wants to exchange x' amount of Bitcoins (henceforth denoted ₿x') for Bob's x' amount of Ethers (henceforth denoted Ɇx).

1. *No optimistic responsiveness.* Ideally, we want that when both parties and the miners are honest, the exchange should complete *responsively*, i.e., as soon as a new block is mined on each chain. Unfortunately, in Chung et al.'s Ponyta protocol, even in the optimistic case, both players must wait for a preset timeout value for the exchange to complete.

2. *Collateral in both Bitcoin and Ether for both parties.* Using Chung et al. [CMST22]'s protocol, besides Alice and Bob's intended payment for each other, i.e., ₿x' and Ɇx, Alice and Bob must prepare *additional collateral in both Bitcoin and Ether.* At a very high level, should Alice or Bob misbehave in the protocol, part of their collateral may be taken away, and this helps the protocol achieve the aforementioned game-theoretic fairness notions.

1.1 Our Results and Contributions

In this paper, we show that if we opt for only CSP fairness and are not concerned about resilience against external incentives, then, we can construct improved protocols that avoid the aforementioned two drawbacks. Specifically, we have the following main results:

1. *Single-instance* RAPIDASH: We construct fair exchange protocol that allows Alice to exchange a secret for Bob's coins, and prove the protocol to satisfy CSP-fairness. Moreover, the protocol satisfies optimistic responsiveness, and does not require Alice to put in any collateral.

2. *Cross-chain atomic swap*: We construct a cross-chain atomic swap protocol that allows Alice to exchange her Bitcoins with Bob's Ether, and prove the protocol CSP-fair. Moreover, the protocol satisfies optimistic responsiveness, and requires Alice to have collateral only in Bitcoins and Bob to have collateral only in Ether. The currencies exchanged can be other currencies but we shall use Bitcoin and Ether as an example for convenience.

3. *Instantiation atop Bitcoin*: Our contracts are easy to implement if the cryptocurrency involved has a general programming language like Ethereum. Interestingly, since our contracts use simple logic, they can be instantiated even atop Bitcoin which has a very limited scripting language.

Our cross-chain atomic swap can be viewed as an application of the single-instance RAPIDASH, since at a very high level, it composes two instances of RAPIDASH, one on Bitcoin and one on Ethereum. What is technically intriguing is that direct composition fails as we explain in more detail in Section 4.2. This means that our single-instance Rapidash does not readily give rise to the atomic swap application as one might anticipate initially. In Section 4, we describe the new tricks that are necessary to overcome the compositional issues thus leading to our atomic swap construction.

1.2 Related Work

Various works showed that user-miner collusion is possible through bribery mechanisms [TYME21, WHF19, HZ20, MHM18, JSZ+21, Ham]. Such bribery attacks may be instantiated in various ways [TYME21, WHF19, HZ20, MHM18, JSZ+21, Ham], e.g., by exploiting decentralized smart contracts.

Tsabary et al. [TYME21] made a pioneering attempt to defend against such bribery attacks. They proposed a fair exchange protocol called a *Mutual-Assured Destruction Hash Timelock Contract (MAD-HTLC)*. Unfortunately, their work defends only against one specific type of bribery attack, but in turn opens up new attacks. They acknowledge in their paper that their scheme does not defend against user-miner collusion. In particular, their protocol is blatantly flawed when one of the users is a miner itself.

The elegant work of Wadhwa et al. [WSZN22] and Chung et al. [CMST22] the most closely related to our work[1]. As mentioned earlier, our work directly builds on top of the formal models and definitions introduced by Chung et al [CMST22]. The work of Wadhwa et al. [WSZN22] describes in greater details techniques to instantiate attacks against MAD-HTLC [TYME21]. The also suggest a new protocol that defends against these attacks. Although they did not explicitly define CSP-fairness, their construction also seems to satisfy CSP fairness. Their protocol (as of the version dated 5/10/2022, see footnote 1) does not satisfy optimistic responsiveness.

Both MAD-HTLC [TYME21] and Wadhwa et al. [WSZN22] consider only a single-instance exchange where one party exchanges a digital secret for the other party's coins. Chung et al. [CMST22] is the first to consider an end-to-end application (specifically, atomic swap) in the study of side-contract-resilient fair exchange. As shown in our work and the earlier work of Chung et al. [CMST22], due to composition issues that arise for game-theoretic fairness notions, a CSP-fair single-instance protocol does not readily lead to any actual application that uses the building block.

2 Preliminaries

We use the same protocol execution model and fairness definitions as Chung et al. [CMST22]. For completeness, we describe the formal definitions below, where some of the description is taken verbatim from Chung et al. [CMST22].

2.1 Blockchain Execution Model

Smart contracts and transactions. We assume that *smart contracts* are *ideal functionalities* 1) enriched with a special type of variable used to denote 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 arbitrary message, 2) some non-negative amount of money, and 3) which activation point of which smart contract it wants to be sent to. When the transaction is executed, the corresponding activation point of the smart

[1]The version dated 5/10/2022 of Wadhwa et al. [WSZN22] is prior work to this paper, and any substantial technical change in later versions is concurrent to our work.

contract will be invoked, and then, some arbitrary computation may take 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 amount of money it has received minus the amount of money it has sent out. *The balance of any smart contract must always be non-negative.*

We assume that each smart contract has a unique name, and each player may have multiple pseudonyms — in practice, a pseudonym is encoded as a public key. A miner is also a special player who is capable of mining blocks.

Mining. In this paper, we do not consider strategies that involve consensus- or network-level attacks — there is an orthogonal and complementary line of work that focuses on this topic [GKL15, PSS17,PS17b], For example, a 51% miner can possibly gain by performing a double spending attack.

For simplicity, we assume an idealized mining process, that is, 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). The winning miner may choose to include a set of transactions in the block, and order these transactions in an arbitrary order. At this moment, a new block is mined, and all (valid) transactions contained in the block are executed. Any transaction that has already been included in the blockchain before is considered invalid and will be ignored. The above idealized mining process can capture standard Proof-of-Work blockchains and Proof-of-Stake blockchains where the next proposer is selected on the fly with probability proportional to the stake held by the miner.

2.2 Players and Strategy Spaces

There are three kinds of players in the model: Alice, Bob, and the miners. We also call Alice and Bob the users to differentiate from miners. We consider the following strategy space for players.

Anyone, including Alice, Bob, or the miners, is allowed to do the following at any point of time:

1. Post a *transaction* to the network at the beginning of any time step. We assume that the network delay is 0, such that transactions posted are immediately seen by all other users and miners. When miners pick which transactions to include in some time step t, they can see transactions posted by users for time step t.

2. Create an arbitrary smart contract and put an arbitrary amount of money into the smart contract. For example, a smart contract can say, "if the state of the blockchain satisfies *some predicate* at *some time*, send *some pseudonym some amount* of money, where the recipient and the amount of money can also be dependent on the state of the blockchain.

Additionally, the *miners* are allowed the following actions: whenever it is chosen to mine a block, it can choose to include an arbitrary subset of the outstanding transactions into the block, and order them arbitrarily. The miner can also create new transactions on the fly and include them in the block it mines.

Coalition. Alice or Bob can form a coalition with some of the miners. When the coalition is formed, all members in the coalition share their private information. The coalition's strategy space is the union of the strategy space of each member in the coalition. Notice that once Alice and Bob are in the same coalition, they can exchange the secret s privately without using the blockchain. Thus, we do not consider the coalition consists of Alice and Bob.

2.3 Protocol Execution

In our paper, an honest protocol is always a *simple* protocol that does not create additional smart contracts in the middle of the execution. Strategic parties can deviate from the honest protocol and create new smart contracts on the fly during the execution.

A protocol execution involves Alice, Bob, and the miners who are modeled as interactive Turing machines who can send and receive a special type of variables called money. Additionally, the protocol may involve one or more smart contracts which can be viewed as ideal functionalities whose states are publicly visible to anyone. Ideal functionalities are also interactive Turing machines capable of sending and receiving money.

For the honest protocol, we always want the miners' honest behavior to be consistent with their honest behavior in typical consensus protocols, i.e., *the miner's honest behavior should include all outstanding transactions in the mined block.*

Finally, since we consider probabilistic polynomial time (PPT) players, we assume that the protocol execution is parametrized by a security parameter λ.

Throughout this paper, we assume that the total utility of all players from the protocol cannot exceed a polynomial function in the security parameter λ.

2.4 Smart Contract Notation

We use the same smart contract notation as Chung et al. [CMST22]. In particular, each activation point is given a type denoted by a single letter. All activation points of the same type are mutually exclusive, i.e., at most one of them can be invoked, and at most once.

Our smart contracts can be instantiated atop either Ethereum or Bitcoin. Note that some extra tricks are needed to instantiate the contracts using Bitcoin since it does support a general-purpose programming language.

2.5 Definition of Game-Theoretic Fairness

We often use $\mathcal{C}$ to denote a coalition, and use $-\mathcal{C}$ to denote all parties of the protocol that are not part of the coalition. We use $HS_{\mathcal{C}}$ or $HS_{-\mathcal{C}}$ to denote the honest strategy executed by either the coalition $\mathcal{C}$ or its complement. Let $S_{\mathcal{C}}$ and $S'_{-\mathcal{C}}$ be the strategies of the coalition $\mathcal{C}$ and its complement. We use $\mathsf{util}^{\mathcal{C}}(S_{\mathcal{C}}, S'_{-\mathcal{C}})$ to denote the expected utility of $\mathcal{C}$ when the coalition $\mathcal{C}$ adopts the strategy $S_{\mathcal{C}}$ and the remaining parties adopt the strategy $S'_{-\mathcal{C}}$.

CSP fairness. We define a game-theoretic fairness notion called cooperative strategy proofness (CSP fairness) — the same notion was formalize earlier in a recent line of works [PS17a, CGL+18, WAS22]. Intuitively, CSP fairness says that a coalition that is profit-driven and wants to maximize its own utility has no incentive to deviate from the honest protocol, as long as all other players play by the book. In this sense, the honest protocol achieves a *coalition-resistant Nash Equilibrium.*

Definition 2.1 (CSP fairness)**.** We say that a protocol satisfies γ-cooperative-strategy-proofness (or γ-CSP-fairness for short), iff the following holds. Let $\mathcal{C}$ be any coalition that controls at most $\gamma \in [0, 1)$ 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

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

where we use HS to mean the honest strategy.

The atomic swap application in Section 4 involves two separate blockchains. In this case, the γ parameter above is an upper bound on the coalition's mining power in both chains.

Dropout resilience. We now define a property called dropout resilience which guarantees that an honest party's utility is non-negative even when the other party is honest but may drop out in the middle.

Definition 2.2 (Dropout resilience)**.** A protocol is said to be dropout resilient, iff the following holds: as long as at least $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, except 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 may drop out in the middle of the protocol execution.

As Chung et al. [CMST22] pointed out, achieving CSP-fairness without dropout resilience is easy. However, requiring the combination of the two properties makes the problem technically challenging. For the atomic swap application in Section 4 which involves two separate blockchains, the $1/\mathsf{poly}(\lambda)$ honest mining power constraint in Definition 2.2 should hold for each chain.

3 Rapidash: Single Instance

3.1 Definitions

Problem definition. Imagine that Alice has some secret pre_a and Bob offers to pay Alice $\$v$ amount of coins in exchange for the secret. For example, pre_a may be a secret value that Bob can later use to unlock some other coins, e.g., through a smart contract.

We assume that the secret pre_a is worth $\$v_a$ and $\$v_b$ to Alice and Bob, respectively. That is, Alice will lose utility $\$v_a$ if pre_a is released to someone else, and Bob will gain $\$v_b$ if he learns pre_a. We assume that $\$v_b > \$v > \$v_a$, such that Alice wants to sell the secret pre_a to Bob at a price of $\$v$.

Players' utility. Let $\beta \in \{0, 1\}$ be an indicator such that $\beta = 1$ if and only if Bob outputs the secret pre_a 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 utility, $\$u_a$, is defined as

$$\$u_a = -\$d_a + \$r_a - \beta \cdot \$v_a$$

and Bob's utility, $\$u_b$, is defined as

$$\$u_b = -\$d_b + \$r_b + \beta \cdot \$v_b$$

Similar to Alice and Bob, we can also define the utility for any miner. Fix some 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.

3.2 Construction

Before deploy the Rapidash contract, Alice and Bob sample $pre_a \in \{0,1\}^\lambda$ and $pre_b \in \{0,1\}^\lambda$ uniformly at random, respectively. Let $\gamma \in [0,1]$ be the upper bound of the fraction of the mining power that the coalition can control. Let $H(\cdot)$ be a cryptographic hash function. The parameters $(h_a, h_b, T_1, T_2, \$v, \$c_b)$ of the Rapidash contract must satisfy the following constraints.

Parameter Constraints for Rapidash

- $h_a = H(pre_a)$ and $h_b = H(pre_b)$.
- $\$c_b > 2 \cdot \$v$.
- $T_1 \geq 1$ and $T_2 \geq 1$ such that $\gamma^{T_2} \leq \frac{\$c_b - \$v}{\$c_b}$.

In Rapidash, T_1 is the time interval such that the seller Alice can redeem the payment $\$v$. For the security, we only require $T_1 \geq 1$. In practice, however, one may want to choose a larger T_1 to account for the network delay. The choice of T_2 allows a tradeoff between waiting time and the amount of the collateral. For example, suppose $\$c_b = 2\$v + \$\epsilon$ for some small $\$\epsilon$. Then, we need to ensure $\gamma^{T_2} \leq 1/2$. This means if $\gamma = 90\%$, we can set $T_2 = 7$. Asymptotically, for any $\gamma = O(1)$, T_2 is a constant. Increasing $\$c_b$ also helps to make T_2 smaller.

The Rapidash contract is specified as follow.

Rapidash contract

Parameters: $h_a, h_b, T_1, T_2, \$v, \$c_b$

Preparation phase: Bob deposits $\$v + \$c_b$

Execution phase:

Payment:

P1: On receive pre_a from Alice such that $H(pre_a) = h_a$, send $\$v$ to Alice and $\$c_b$ to Bob.

P2: Time T_1 or greater: on receive pre_b from Bob such that $H(pre_b) = h_b$, send $\$v$ to Bob.

Collateral:

C1: At least T_2 after P2 is activated: on receiving _ from anyone, send $\$c_b$ to Bob.

C2: If P2 is activated, on receive (pre_a, pre_b) from anyone P such that $H(pre_a) = h_a$ and $H(pre_b) = h_b$, send $\$v$ to player P. All remaining coins are burnt.

The Rapidash protocol. During the preparation phase, the buyer Bob deposits $\$c_b + \$v$ into the Rapidash contract. When Bob's deposit transaction is confirmed, we define the current block number (i.e., time) to be $t = 0$. The execution phase proceeds as follows — henceforth, we use the phrase "a player sends a message to an activation point" to mean that "the player posts a transaction containing the message and destined for the activation point":

- *Alice*: Alice sends pre_a to activation point P1 at $t = 0$.
- *Bob*: If Alice failed to send pre_a to P1 before time T_1, then Bob sends pre_b to P2 at time $t = T_1$. T_2 time after P2 is activated, he sends an empty message _ to C1.

If either P1 or C2 is successfully activated, Bob outputs the corresponding pre_a value included in the corresponding transaction. If P2 and C1 are successfully activated, Bob outputs $\perp$.

- *Miner*: The miner watches all transactions posted to P1, P2, and C2. If the miner has observed the correct values of both pre_a and pre_b from these posted transactions, then it sends (pre_a, pre_b) to C2. Further, any miner always includes all outstanding transactions in every block it mines. If there are multiple transactions posted to C2, the miner always places its own ahead of others (and thus invalidating the others).

3.3 Proofs

Lemma 3.1. *Let $\mathcal{C}$ be any coalition that consists of Alice and an arbitrary subset of miners*[2] *(possibly no miner). Then, 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_a at $t = 0$, and P1 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[3] — 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, P1 is activated at some moment. Notice that C1 or C2 can be activated only if P2 is activated. Because P1 and P2 are mutually exclusive, conditioned on P1 being activated, C1 and C2 can never be activated. Thus, the utility of $\mathcal{C}$ is $\$v - \$v_a$, which the same as the honest case.

- Second, P1 is never activated. Notice that neither Alice nor the miner can get anything from P2 or C1, so the utility of $\mathcal{C}$ can be positive only if C2 is activated. However, when C2 is activated, pre_a is publicly known, so the utility of $\mathcal{C}$ is $\$v - \$v_a$, which the same as the honest case.

$\square$

Lemma 3.2. *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 $\gamma^{T_2} \leq \frac{\$c_b - \$v}{\$c_b}$, for any (even unbounded) coalition strategy $S_\mathcal{C}$, it must be that*

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

[2] 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.

[3] 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.

Proof. The honest Alice will always send pre_a to P1. Thus, when $\mathcal{C}$ follows the protocol, P1 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 3.1, 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.

- First, neither P1 or P2 is activated. In this case, C1 and C2 cannot be activated. Because Bob deposits $\$v + \$c_b$ at the beginning of the protocol, the utility of $\mathcal{C}$ is $\$v_b - \$v - \$c_b$, since $\mathcal{C}$ cannot get the deposit back.

- Second, P1 is activated. Because P1 and P2 are mutually exclusive, conditioned on P1 being activated, C1 and C2 can never be activated. Thus, the utility of $\mathcal{C}$ is $\$v_b - \$v$, which the same as the honest case.

- Third, P2 is activated. Let $t^* \geq T_1$ be the time at which P2 is activated. There are two subcases. In the first subcase, the coalition also gets $\$v$ from C2 at time t^*. In this case, the coalition's utility is at most $\$v + \$v_b - \$c_b$, and since $\$c_b > 2 \cdot \$v$, this is less than the honest case. Henceforth, we may assume that the coalition has not got $\$v$ from C2 at time t^*. Since the honest Alice posts pre_a at $t = 0$ and $t^* \geq T_1$, both pre_a and pre_b are publicly known at t^*. Since all non-colluding miners are honest, after t^*, they will activate C2 themselves when they mine a new block if C2 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 C1 or C2, and thus its utility is at most $\$v_b - \$c_b$. 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

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

  Recall that $\$c_b > 2 \cdot \$v$. Therefore, for $(\$v_b - \$c_b) \cdot (1 - p) + \$v_b \cdot p$ to exceed the honest utility $\$v_b - \$v$, it must be that $p > \frac{\$c_b - \$v}{\$c_b}$ which contradicts our assumption.

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

Theorem 3.3 (CSP fairness)**.** *Suppose that the hash function $H(\cdot)$ is a one-way function and that $\gamma^{T_2} \leq \frac{\$c_b - \$v}{\$c_b}$. Then, the* Rapidash *protocol satisfies γ-CSP-fairness.*

Proof. Lemmas 3.1 and 3.2 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 C2 is activated. However, C2 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 Rapidash is dropout resilient.

Theorem 3.4 (Dropout resilience)**.** *Suppose that $H(\cdot)$ is a one-way function and that all players are PPT machines.* Rapidash *is dropout resilient. In other words, suppose at least $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest. If either Alice or Bob plays honestly but drops out before the end of the protocol, then with $1 - \mathsf{negl}(\lambda)$ probability, the other party's utility should be non-negative.*

Proof. Throughout the proof, for any $X \in \{pre_a, pre_b\}$, 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_a; 2) Alice drops out after she already posted a transaction containing pre_a at $t = 0$. In the first case, as long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, Bob would activate P2 and C1 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 P2. An honest miner would include Alice's transaction and activate P1. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P1 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_a, and except with negligible probability, P1 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. □

Remark 3.5 (Why the protocol does not defend against external incentives)**.** As mentioned, Chung et al. [CMST22]'s protocol provides an additional fairness property called "safe participation against external incentives" — however, they also pay a price to provide resilience against external incentives in the sense that *both* parties have to put down collateral.

Specifically, Chung et al. [CMST22] show that no matter how large and how arbitrary the external incentives may be, as long as there is an explicit bound on the amount of external incentive, one can always increase the parties' collaterals accordingly such that their construction would provide resilience. By contrast, we want that Alice should not have to put down collateral. In this case, if the external incentive is sufficiently large, we cannot provide safe participation for the honest player even when we are allowed to increase Bob's collateral arbitrarily.

Suppose that Bob is honest, and the external incentive encourages the Alice-miner coalition to post pre_a to P1 after T_1, i.e., after Bob has timed out and posted pre_b. The coalition will be incentivized to do so as long as the external incentive is at least $\$v + \$\epsilon$. Unfortunately, in this case, the honest Bob will have expected negative utility. If we insist that Alice does not put down any collateral, no matter how much we increase Bob's collateral, we cannot defend against $\$v + \$\epsilon$ amount of external incentive.

4 Side-Contract-Resilient Cross-Chain Atomic Swap

4.1 Definition

We use the same problem definition as in Chung et al. [CMST22]. For completeness, below, we state the definitions using some description from Chung et al. [CMST22] verbatim.

Suppose that Bob has x amount of Ethers denoted $\text{Ξ}x$, and Alice as x' amount of Bitcoins denoted $\text{₿}x'$. Bob wants to exchange his $\text{Ξ}x$ with Alice's $\text{₿}x'$. The currencies exchanged may be other currencies but we shall use Bitcoin and Ether as an example.

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

Given some strategic player or coalition, we assume that it has some specific valuation of each unit of Bitcoin and each unit of Ether. For convenience, we use the notation $\$\mathsf{AV}(\cdot)$ to denote the valuation function of Alice of an Alice-miner coalition; specifically, $\$\mathsf{AV}(\text{Ξ}x + \text{₿}x') = \$v_a \cdot x + \$v'_a \cdot x'$

where $\$v_a \geq 0$ and $\$v'_a \geq 0$ denote how much Alice or the Alice-miner coalition values each Ether and Bitcoin, respectively. Similarly, we use the notation $\$\mathsf{BV}(\cdot)$ 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. Throughout this section, we may make the following assumption which justifies why Alice wants to exchange her $\text{₿}x'$ with Bob for $\text{Ξ}x$, and vice versa.

$$\textbf{Assumption:} \quad \$\mathsf{AV}(\text{Ξ}x - \text{₿}x') > 0, \quad \$\mathsf{BV}(\text{₿}x' - \text{Ξ}x) > 0$$

Utility. Let $\text{₿}d'_a, \text{Ξ}d_a \geq 0$ be the cryptocurrencies that Alice or an Alice-miner coalition deposit into the smart contracts. Let $\text{₿}r'_a, \text{Ξ}r_a \geq 0$ be the payment Alice or an Alice-miner coalition receive from the smart contracts during the protocol. Now, we can define the utility $\$u_a$ of Alice or the Alice-miner coalition as follows:

$$\$u_a = \$\mathsf{AV}(\text{₿}r'_a - \text{₿}d'_a + \text{Ξ}r_a - \text{Ξ}d_a)$$

Similarly, we can define the utility $\$u_b$ of Bob or a Bob-miner coalition, and the utility $\$u_m$ of a miner-only coalition as follows:

$$\$u_b = \$\mathsf{BV}(\text{₿}r'_b - \text{₿}d'_b + \text{Ξ}r_b - \text{Ξ}d_b)$$

$$\$u_m = \$\mathsf{MV}(\text{₿}r'_m - \text{₿}d'_m + \text{Ξ}r_m - \text{Ξ}d_m)$$

where $\text{₿}r'_b, \text{Ξ}r_b \geq 0$, denote the payment the Bob-miner coalition or Bob receives during the protocol, and $\text{₿}d'_b, \text{Ξ}d_b \geq 0$ denote the deposit the Bob-miner coalition or Bob sends to any smart contract during the protocol. The variables $\text{₿}r'_m, \text{Ξ}r_m, \text{₿}d'_m, \text{Ξ}d_m \geq 0$ are similarly defined but for the miner-only coalition.

Like before, we assume that the total utility of all players from the protocol cannot exceed a polynomial function in the security parameter λ.

Modeling time. In our cross-chain atomic swap application, since the two blockchains have different block intervals, we use the following convention for denoting time. Without loss of generality, we may assume that the moment the protocol execution begins, the current lengths of the Bitcoin and Ethereum chains are renamed to 0. We use the terminology Ethereum time T to refer to the moment the Ethereum chain reaches length T, and similarly, we use the terminology Bitcoin time T' to refer to the moment when the Bitcoin chain reaches length T'.

4.2 Strawman Idea

The strawman idea is to directly compose two instances of the basic Rapidash contract described in Section 3, one on Ethereum, and one on Bitcoin. We describe the strawman protocol informally leaving out some parameter details, such that it is already sufficient to see the flaw. Henceforth, we refer to the contract on Ethereum as Rapidash and the contract on Bitcoin as Rapidash'. The activation points of Rapidash are referred to as P1, P2, C1, and C2; whereas the activation points of Rapidash' are referred to as P1', P2', C1', and C2'.

During the preparation phase, Alice makes the first move and deposits $\text{₿}x' + \text{₿}c'_a$ into Rapidash' where $\text{₿}x'$ denotes the intended payment and $\text{₿}c'_a$ denotes Alice's collateral. Bob then deposits $\text{Ξ}x + \text{Ξ}c_b$ into Rapidash where $\text{Ξ}x$ denotes the intended payment and $\text{Ξ}c_b$ denotes Bob's collateral. During the execution phase, Alice posts a secret pre_a to P1 of Rapidash which allows her to get Bob's payment $\text{Ξ}x$ and Bob gets back his collateral $\text{Ξ}c_b$. Now, Bob also knows pre_a so he can post the same secret pre_a to P1' of Rapidash', such that he can get Alice's payment $\text{₿}x'$ and Alice would get back her collateral $\text{₿}c'_a$.

If Alice fails to post pre_a to P1 of Rapidash on time, then Bob can invoke P2 and C1 to get his intended payment ₿x and ₿c_b back. Similarly, if Bob fails to post pre'_a to P1' of Rapidash' on time, then Alice can get her intended payment ₿x' and ₿c'_a back by invoking P2' and C1'.

Why the strawman is insecure. It turns out that composing two instances of the basic Rapidash of Section 3 introduces tricky technicalities. The strawman protocol described above is subject to the following attack by a Alice-miner coalition. First, Alice does not post pre_a on time, such that the honest Bob will post pre_b to P2 of Rapidash. At this moment, the Alice-miner coalition suppresses Bob's transaction from being confirmed. After sufficiently long, the Alice-miner coalition can trigger P2' and C1' of Rapidash' such that they get back all of Alice's deposit (including intended payment and collateral) from Rapidash'. At this moment, the secret pre_a is no longer valuable, so the Alice-miner coalition can invoke either P1 or C2, such that it can get ₿x. This way, the coalition basically obtained ₿x for nothing.

In summary, the fundamental reason why direct composition of two basic Rapidash contracts fails is because the secret pre_a loses its value once the Rapidash' contract has been redeemed, whereas in the earlier Section 3, the assumption is that pre_a's value is permanent.

4.3 Our Construction

Intuition. A straightforward idea to fix the aforementioned problem is require that P1 of Rapidash be activated early enough. This is possible with general smart contracts such as Ethereum, but does not seem possible with Bitcoin. Instead, we want to have a unified approach that can be instantiated using either Ethereum or Bitcoin. Our idea is to ensure that Alice does not have incentives to invoke P2' of Rapidash' should the honest Bob post pre_b to P2 of Rapidash. To achieve this, we modify the C2' activation point such that Alice's collateral will be burnt if anyone submits the pair (pre'_a, pre_b).

Unfortunately, this modification harms the dropout resilience of the protocol. Suppose that both Alice and Bob are honest but some miners may act strategically so transactions confirmation may be delayed. Now, if Bob's deposit transaction into Rapidash is confirmed late in time, Bob will post pre_b to P2. At this moment, we can no longer have Alice post pre'_a to P2' to get her deposit back from Rapidash'. One idea is to let Bob post an empty message _ P2' to help Alice get her deposit back from Rapidash'— to defend against a strategic Alice-miner coalition, it is only safe for Bob to do this after he gets all of his deposit back from Rapidash. However, if the honest Bob drops offline immediately after posting pre_b to P2, then the honest Alice would have no means to get her deposit back from Rapidash'.

To fix this issue, we introduce a new *two-phase deposit* trick. Unlike all known existing atomic swap protocols, we have Bob first deposit into Rapidash, and only then will Alice deposit into Rapidash'. Importantly, at this moment, Bob must give explicit consent to unfreeze the activation points P1 and C2 of Rapidash— this is necessary since otherwise Alice can get Bob's ₿x without even depositing into Rapidash'. To achieve this, we have Bob post another secret pre_c if Alice's deposit transaction into Rapidash' is confirmed in time; and moreover, the secret pre_c is necessary for activating. P1 and C2. With the above modification, it is as if Bob's deposit into Rapidash is performed in two-phases: first by locking his coins into Rapidash and then by posting pre_c to give explicit consent to unfreeze the activation points P1 and C2.

With this two-phase deposit trick, should the execution enter the abort phase prior to the unfreezing of P1 and C2, Bob can help Alice invoke P2' first before it activates P2 to get his own deposit back — this is now safe for Bob because Bob knows that without him posting pre_c, the Alice-miner coalition cannot cash out from Rapidash. Should Bob fail to help Alice invoke P2' by some deadline (and if Bob is honest he will not have posted pre_b to P2 either), Alice can then post pre'_a to P2' to get her deposit back herself.

Finally, there is just one remaining issue: we do not want a strategic Bob-miner coalition to wait for Alice to post pre'_a and then post pre_b to cash out from C2. Thus, to make the protocol finally work, we also need to augment the activation point C2 such that Bob will be punished should (pre'_a, pre_b) be presented to C2.

Parameters. Before deploying the RAPIDASH contract, Alice samples $pre_a \in \{0,1\}^\lambda$ and $pre'_a \in \{0,1\}^\lambda$ uniformly at random. Similarly, Bob samples $pre_b \in \{0,1\}^\lambda$ and $pre_c \in \{0,1\}^\lambda$ uniformly at random. As in Section 3, let $\gamma \in [0,1]$ be the upper bound of the fraction of the mining power controlled by the coalition (for either chain), and let $H(\cdot)$ be a one-way function. In our protocol below, Alice needs to have an extra $\text{₿}c'_a$ as collateral besides her intended payment to Bob $\text{₿}x'$; similarly, Bob needs to have an extra $\text{Ɇ}c_a$ as collateral besides his intended payment $\text{Ɇ}x$ to Alice.

The parameters used in our atomic swap protocol must respect the following constraints.

Parameter Constraints for Atomic Swap

Constraints for Rapidash (on Ethereum):

- $h_a = H(pre_a)$, $h_b = H(pre_b)$ and $h_c = H(pre_c)$.
- $T_1 > T_0 > T > 0$.
- $\text{Ɇ}c_b > 2 \cdot \text{Ɇ}x$.

Constraints for Rapidash' (on Bitcoin):

- $h'_b = H(pre_a) = h_a$ and $h'_a = H(pre'_a)$.
- Ethereum time $T <$ Bitcoin time $T' <$ Ethereum time T_0, i.e., the Ethereum block of length T is mined before the Bitcoin block of length T', and the Bitcoin block of length T' is mined before the Ethereum block of length T_0.[a]
- Bitcoin time $T'_1 >$ Ethereum time T_1, i.e., the Bitcoin block of length T'_1 is mined after the Ethereum block of length T_1.
- $\text{₿}c'_a > 2 \cdot \text{₿}x'$.

Choice of timeouts: *// γ is the coalition's fraction of mining power*

- $\tau \geq 1$, $\tau' \geq 1$.
- $\gamma^{\tau'} \leq \frac{\$\mathsf{AV}(\text{₿}c'_a - 2\text{₿}x')}{\$\mathsf{AV}(\text{₿}c'_a - \text{₿}x')}$ and $\gamma^{\tau} \leq \frac{\$\mathsf{BV}(\text{Ɇ}c_b - 2\text{Ɇ}x)}{\$\mathsf{BV}(\text{Ɇ}c_b - \text{Ɇ}x)}$.

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

In the above, T, T_0, T_1, T', T'_1 are timeout values that Alice and Bob wait for during the protocol. The choices stated above are for our theoretical model where the network delay is assumed to be zero. In practice, we should adjust these parameters accordingly to account for the network delay. Concretely, let Δ denote the network delay. Then, $T_1 - T_0, T_0 - T, T$ amount of Ethereum time should all be larger than Δ. Similarly, the interval between Ethereum time T and Bitcoin time T', the interval between Bitcoin time T' and Ethereum time T_0, and the interval between Ethereum time T_1 and Bitcoin time T_1' should all be larger than Δ. Importantly, *even if the actual network delay is greater than the anticipated network delay, the CSP fairness of the protocol still holds*, but we may lose liveness (i.e., Alice and Bob may simply get refunded and not complete the exchange).

In our protocol, if both parties and the miners are honest, then the exchange completes *responsively*, i.e., as soon as a new block is mined on each chain. If, however, one of the players drops out and the exchange is cancelled as a result, the remaining player may incur some delay to get its deposit back. Similarly, if the miners are delaying Alice and/or Bob's transactions, then the exchange may be cancelled and Alice and/or Bob may need to wait to get their deposit back. Specifically, in the worst case, Bob may wait up to $T_1 + \tau$ Ethereum time to get his deposit back, and Alice may wait $T_1' + \tau'$ Bitcoin time to get her deposit back.

The timeouts τ and τ' must be sufficiently large w.r.t. γ, such that a coalition involving Alice or Bob would never want to take any gamble that would risk getting their collateral (partially) burnt. For example, suppose we choose Ɇ$c_b = 3 \cdot$ Ɇx. Then, we need to make sure $\gamma^\tau \leq 1/2$. This means if $\gamma = 90\%$, we can set $\tau = 7$. Asymptotically, for any $\gamma = O(1)$, τ is a constant. Increasing Ɇc_b helps to make τ smaller. A similar calculation also works for τ' and ₿c_a'.

Contracts. We now describe the contracts for both chains.

Rapidash contract (on Ethereum) // Parameters: (h_a, h_b, T_1, τ, Ɇx, Ɇc_b)

Preparation phase: Bob deposits Ɇx + Ɇc_b.

Execution phase:

P1: On receive pre_a from Alice such that $H(pre_a) = h_a$ and pre_c from Bob such that $H(pre_c) = h_c$, send Ɇx to Alice and Ɇc_b to Bob.

P2: Time T_1 or greater: On receive pre_b from Bob such that $H(pre_b) = h_b$, send Ɇx to Bob.

C1: At least τ after P2 is activated: on receiving _ from anyone, send Ɇc_b to Bob.

C2: If P2 is activated, on receive (pre_a, pre_b, pre_c) or (pre_a', pre_b) from anyone P such that $H(pre_a) = h_a$, $H(pre_b) = h_b$, $H(pre_c) = h_c$, and $H(pre_a') = h_a'$, send Ɇx to player P. All remaining coins are burnt.

Rapidash' contract (on Bitcoin) // Parameters: (h_b', h_a', T_1', τ', ₿x', ₿c_a')

Preparation phase: Alice deposits ₿x' + ₿c_a'.

Execution phase:

P1': On receiving pre_b' from Bob[a] such that $H(pre_b') = h_b'$ or on receiving _ from Alice, send ₿x' to Bob and send ₿c_a' to Alice.

P2': Time T_1' or greater: on receiving pre_a' from Alice such that $H(pre_a') = h_a'$ or on receiving _ from Bob, send ₿x' to Alice.

C1': At least τ' after P2' is activated: on receiving _ from anyone, send ₿c_a' to Alice.

C2': If P2' is activated, on receiving (pre_b', pre_a') or (pre_a', pre_b) from anyone P such that $H(pre_b') = h_b'$, $H(pre_a') = h_a'$ and $H(pre_b) = h_b$, send ₿x' to player P. All remaining coins are burnt.

[a]Bob will let pre_b' be the pre_a he learns in the RAPIDASH instance.

The atomic swap protocol. We describe the atomic swap protocol below.

- *Miner.* The miner's honest protocol is described below.
  - The miner watches all transactions posted to P1, P2, C1, C2, P1', P2', C1', and C2' (i.e., all the P-type and C-type activation points for both contracts), to see if they contain a valid $pre_a = pre'_b$, pre_b, pre'_a, and pre_c.
  - As soon as the miner has observed pre_a, pre_b and pre_c, it posts (pre_a, pre_b, pre_c) to C2; as soon as the miner has observed both pre'_a and pre'_b, it posts (pre'_a, pre'_b) to C2'; as soon as the miner has observed pre'_a and pre_b, it posts (pre'_a, pre_b) to C2 and C2'.
  - Whenever the miner mines a block, it always includes its own transactions ahead of others.
- *Alice and Bob.* Below, we define the honest protocol for Alice and Bob. The moment that both contracts have been posted and take effect is defined to be the start of the execution (i.e. $t = 0$). We define Ethereum time 0 and Bitcoin time 0 to be the length of Ethereum and Bitcoin when the execution starts, respectively.

Atomic Swap Protocol — Alice and Bob

Preparation Phase:

1. At $t = 0$, Bob sends the deposit transaction of $\text{Ξ}x + \text{Ξ}c_b$ to Rapidash.
2. At Ethereum time T: If Bob's deposit transaction has not been confirmed, Alice and Bob go to the abort phase; otherwise, if Bob's deposit transaction is confirmed, Alice sends the deposit transaction of $\text{₿}x' + \text{₿}c'_a$ to Rapidash'.
3. At Bitcoin time T': If Alice's deposit transaction has not been confirmed, Alice and Bob go to the abort phase; otherwise, if Alice's deposit transaction is confirmed, Bob sends pre_c to P1.
4. At Ethereum time T_0: if Bob has not sent pre_c to P1, Alice and Bob go to the abort phase; else, both parties go to the execution phase.

Execution Phase:

1. At Ethereum time T_0, Alice sends pre_a to P1. As soon as P1 has been activated, Alice sends an empty message $\_$ to P1'.
2. If Alice does not send pre_a to P1 before Ethereum time T_1, Bob sends pre_b to P2.
3. If Alice sends pre_a to P1 before Ethereum time T_1, Bob sends $pre'_b = pre_a$ to P1' at Ethereum time T_1.

Abort Phase:

1. At Ethereum time T_0, Bob sends $\_$ to P2' and pre_b to P2.
2. At Ethereum time T_1, if Bob has not sent $\_$ to P2', Alice sends pre'_a to P2'.
3. If τ' *Bitcoin time* has passed since P2' is activated, Alice sends $\_$ to C1'; similarly, if τ *Ethereum time* has passed since P2 is activated, Bob sends $\_$ to C1.

Remark 4.1. At Step 4 of the preparation phase, which phase does Alice go depends on whether Bob has sent pre_c to P1. In this work, we assume Alice can always observe the fact that Bob sent pre_c to P1 immediately if Alice and Bob are both honest. In practice, Bob can sign the transaction, and send the signature to Alice. Then, Alice is responsible for sending pre_c to P1 on behalf of Bob. Consequently, as long as Alice and Bob are both honest, they always make the same decision (regarding whether to go to the abort phase) when they both reach Step 4 of the preparation phase.

4.4 Proofs

Lemma 4.2 (Alice-miner coalition). *Suppose that the hash function $H(\cdot)$ is a one-way function. Let $\mathcal{C}$ be any coalition that consists of Alice and an arbitrary subset of miners controlling at most γ fraction of mining power. Then, as long as $\gamma^{\tau'} \leq \frac{\$\mathsf{AV}(\text{₿}c'_a - 2\text{₿}x')}{\$\mathsf{AV}(\text{₿}c'_a - \text{₿}x')}$, for any PPT 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, P1 and P1' will be activated, and the utility of $\mathcal{C}$ is $\$\mathsf{AV}(\text{Ξ}x - \text{₿}x') > 0$. Now, suppose $\mathcal{C}$ may deviate from the protocol. We analyze the possible cases depending on which phase Bob enters. Notice that if Alice's deposit transaction is confirmed before Bitcoin time T', the honest Bob always sends pre_c to P1, and enters the execution phase at Ethereum time T_0.

Bob enters the abort phase at Ethereum time T or at Bitcoin time T'. In this case, Bob never sends pre_c to P1, so except the negligible probability that $\mathcal{C}$ finds pre_c by itself, P1 and C2 can never be activated. $\mathcal{C}$ can earn coins from RAPIDASH only through P1 and C2. Thus, except the negligible probability, the utility of $\mathcal{C}$ is at most zero, which is less than the honest case.

Bob enters the execution phase. $\mathcal{C}$ can earn coins from RAPIDASH either from P1 or from C2 but not both, and in either case, $\mathcal{C}$ gain $\text{Ξ}x$. Thus, if P1' is activated, the utility of $\mathcal{C}$ can never be more than the honest case. Consequently, to make the utility of $\mathcal{C}$ be more than $\$\mathsf{AV}(\text{Ξ}x - \text{₿}x')$, P2' must be activated. Henceforth, we assume P2' is activated at Bitcoin time $t^* \geq T'_1$. If Bob enters the execution phase, P2' can only be activated by Alice posting pre'_a to P2'. So pre'_a is publicly known after Bitcoin time t^*. Depending on whether Alice sends pre_a to P1 before Ethereum time T_1, there are two subcases.

- *Case 1: Alice sends pre_a to P1 before Ethereum time T_1.* Because Ethereum time T_1 is earlier than Bitcoin time T'_1, pre_a and pre'_a are both publicly known at Bitcoin time t^*. Notice that $pre_a = pre'_b$ by definition. Thus, during Bitcoin time $(t^*, t^* + \tau']$, any miner in $-\mathcal{C}$ will activate C2' if it wins a block. We say $\mathcal{C}$ loses the race if a non-colluding miner mines a new block during Bitcoin time $(t^*, t^* + \tau']$. Otherwise, we say $\mathcal{C}$ wins the race. If $\mathcal{C}$ loses the race, it gets nothing from C1' or C2', and it gets at most $\$\mathsf{AV}(\text{₿}x' - \text{₿}c'_a + \text{Ξ}x)$ from P1 and P2'. Else if $\mathcal{C}$ wins the race, then its utility is at most $\$\mathsf{AV}(\text{Ξ}x)$ which can be achieved by activating P2', C1' and P1. The probability p that $\mathcal{C}$ wins the race is upper bounded by $p \leq \gamma^{\tau'}$. Therefore, the expected utility of $\mathcal{C}$ is upper bounded by

$$\$\mathsf{AV}((\text{₿}x' - \text{₿}c'_a + \text{Ξ}x) \cdot (1-p) + \text{Ξ}x \cdot p).$$

  Since $p \leq \gamma^{\tau'} \leq \frac{\$\mathsf{AV}(\text{₿}c'_a - 2\text{₿}x')}{\$\mathsf{AV}(\text{₿}c'_a - \text{₿}x')}$, we have

$$\$\mathsf{AV}((\text{₿}x' - \text{₿}c'_a + \text{Ξ}x) \cdot (1-p) + \text{Ξ}x \cdot p) < \$\mathsf{AV}(\text{Ξ}x - \text{₿}x').$$

- *Case 2: Alice does not send pre_a to P1 before Ethereum time T_1.* In this case, the honest Bob will send pre_b to P2 at Ethereum time T_1. Because P2' is activated at Bitcoin time $t^* \geq T_1'$, which is later than Ethereum time T_1, pre_a' and pre_b are both publicly known at Bitcoin time t^*. Thus, during Bitcoin time $(t^*, t^* + \tau']$, any miner in $-\mathcal{C}$ will activate C2' if it wins a block. As before, we say $\mathcal{C}$ loses the race if a non-colluding miner mines a new block during Bitcoin time $(t^*, t^* + \tau']$ and $\mathcal{C}$ wins the race otherwise. If $\mathcal{C}$ loses the race, it gets nothing from C1' or C2', and it gets at most $\$\mathsf{AV}(\text{₿}x' - \text{₿}c_a' + \text{Ξ}x)$ from P2' and (P1 or C2)[4]. Else if $\mathcal{C}$ wins the race, then its utility is at most $\$\mathsf{AV}(\text{Ξ}x)$ which can be achieved by activating P2', C1' and P1. By the same calculation as the previous case, since $p \leq \gamma^{\tau'} \leq \frac{\$\mathsf{AV}(\text{₿}c_a' - 2\text{₿}x')}{\$\mathsf{AV}(\text{₿}c_a' - \text{₿}x')}$, we have

$$\$\mathsf{AV}((\text{₿}x' - \text{₿}c_a' + \text{Ξ}x) \cdot (1-p) + \text{Ξ}x \cdot p) < \$\mathsf{AV}(\text{Ξ}x - \text{₿}x').$$

□

Lemma 4.3 (Bob-miner coalition). *Suppose that the hash function $H(\cdot)$ is a one-way function. 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 $\gamma^{\tau} \leq \frac{\$\mathsf{BV}(\text{Ξ}c_b - 2\text{Ξ}x)}{\$\mathsf{BV}(\text{Ξ}c_b - \text{Ξ}x)}$, for any PPT coalition strategy $S_{\mathcal{C}}$, it must be that there is a negligible function $\mathsf{negl}(\cdot)$ such that*

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

Proof. When the coalition $\mathcal{C}$ follows the protocol, P1 and P1' will be activated, and the utility of $\mathcal{C}$ is $\$\mathsf{BV}(\text{₿}x' - \text{Ξ}x) > 0$. Now, suppose $\mathcal{C}$ may deviate from the protocol. Notice that no matter which phase Alice goes, if Alice sends pre_a to P1, it must be at Ethereum time T. We analyze the two possible cases depending on whether Alice sends pre_a to P1.

Alice enters the abort phase. In this case, Alice never sends pre_a to P1. In this proof, we ignore the negligible probability that $\mathcal{C}$ can find pre_a by itself. $\mathcal{C}$ can earn coins from Rapidash' only through P1' and C2'. However, $\mathcal{C}$ has to send pre_a to P1' to make it activated, while $\mathcal{C}$ does not know pre_a. Thus, to make $\mathcal{C}$'s utility positive, C2' must be activated by $\mathcal{C}$. C2' can be activated by sending (pre_b', pre_a') or (pre_a', pre_b). Because $\mathcal{C}$ does not know $pre_b' = pre_a$, C2' must be activated by (pre_a', pre_b). Henceforth, we assume C2' is activated by (pre_a', pre_b) sent by $\mathcal{C}$. To activate C2' by (pre_a', pre_b), $\mathcal{C}$ must know pre_a'. Ignoring the negligible probability that $\mathcal{C}$ finds pre_a' by itself, Alice must send a transaction containing pre_a', which must be at Ethereum time T_1.

Next, because Alice never sends pre_a to P1, P1 can never be activated. Moreover, C1 or C2 can be activated only if P2 has been activated. Thus, if P2 is not activated, $\mathcal{C}$'s deposit is locked in Rapidash, and $\mathcal{C}$'s utility is at most $\$\mathsf{BV}(\text{₿}x' - \text{Ξ}x - \text{Ξ}c_b)$ which is no more than the honest case $\$\mathsf{BV}(\text{₿}x' - \text{Ξ}x)$. Henceforth, we assume P2 is activated at some Ethereum time $t^* \geq T_1$. Because Alice sends pre_a' at Ethereum time T_1, pre_b and pre_a' are both publicly known at Ethereum time t^*. Thus, during Ethereum time $(t^*, t^* + \tau]$, any miner in $-\mathcal{C}$ will activate C2 if it wins a block. We say $\mathcal{C}$ loses the race if a non-colluding miner mines a new block during Ethereum time $(t^*, t^* + \tau]$. Otherwise, we say $\mathcal{C}$ wins the race. If $\mathcal{C}$ loses the race, it gets nothing from C1 or C2, and it gets at most $\$\mathsf{BV}(\text{Ξ}x - \text{Ξ}c_b + \text{₿}x')$ from P2 and C2'. Else if $\mathcal{C}$ wins the race, then its utility is at most $\$\mathsf{BV}(\text{₿}x')$ which can be achieved by activating P2, C1 and C2'. The probability p that $\mathcal{C}$ wins the race is upper bounded by $p \leq \gamma^{\tau}$. Therefore, the expected utility of $\mathcal{C}$ is upper bounded by

$$\$\mathsf{BV}((\text{Ξ}x - \text{Ξ}c_b + \text{₿}x') \cdot (1-p) + \text{₿}x' \cdot p).$$

[4] Alice may send pre_a to P1 or C2 after Ethereum time T_1.

Since $p \le \gamma^\tau \le \frac{\$\mathsf{BV}(\text{Ɇ}c_b - 2\text{Ɇ}x)}{\$\mathsf{BV}(\text{Ɇ}c_b - \text{Ɇ}x)}$, we have

$$\$\mathsf{BV}((\text{Ɇ}x - \text{Ɇ}c_b + \text{₿}x') \cdot (1-p) + \text{₿}x' \cdot p) < \$\mathsf{BV}(\text{₿}x' - \text{Ɇ}x).$$

Alice enters the execution phase. In this case, Bob must have sent pre_c to P1, and Alice sends pre_a to P1 at Ethereum time T_0. $\mathcal{C}$ can gain coins from RAPIDASH' either from P1' or from C2', but not both, and in either case, $\mathcal{C}$ gains the same amount $\text{₿}x'$. Thus, if P1 is activated, the utility of $\mathcal{C}$ is at most the same as the honest case (from P1 and (P1' or C2')). Consequently, to make $\mathcal{C}$'s utility more than $\$\mathsf{BV}(\text{₿}x' - \text{Ɇ}x)$, P2 must be activated. Henceforth, we assume P2 is activated at Ethereum time $t^* \ge T_1$, so pre_b is publicly known after Ethereum time t^*. Because Alice sends pre_a to P1 at Ethereum time T_0 and $T_0 < T_1$, pre_a and pre_b are both publicly known at Ethereum time t^*. Thus, during Ethereum time $(t^*, t^* + \tau]$, any miner in $-\mathcal{C}$ will activate C2 if it wins a block. We say $\mathcal{C}$ loses the race if a non-colluding miner mines a new block during Ethereum time $(t^*, t^* + \tau]$. Otherwise, we say $\mathcal{C}$ wins the race. If $\mathcal{C}$ loses the race, it gets nothing from C1 or C2, and it gets at most $\$\mathsf{BV}(\text{Ɇ}x - \text{Ɇ}c_b + \text{₿}x')$ from P2 and (P1' or C2'). Else if $\mathcal{C}$ wins the race, then its utility is at most $\$\mathsf{BV}(\text{₿}x')$ which can be achieved by activating P2, C1 and (P1' or C2'). By the same calculation as the previous case, since $p \le \gamma^\tau \le \frac{\$\mathsf{BV}(\text{Ɇ}c_b - 2\text{Ɇ}x)}{\$\mathsf{BV}(\text{Ɇ}c_b - \text{Ɇ}x)}$, we have

$$\$\mathsf{BV}((\text{Ɇ}x - \text{Ɇ}c_b + \text{₿}x') \cdot (1-p) + \text{₿}x' \cdot p) < \$\mathsf{BV}(\text{₿}x' - \text{Ɇ}x).$$

□

Theorem 4.4 (CSP fairness)**.** *Suppose that the hash function $H(\cdot)$ is a one-way function. Moreover, suppose $\gamma^{\tau'} \le \frac{\$\mathsf{AV}(\text{₿}c'_a - 2\text{₿}x')}{\$\mathsf{AV}(\text{₿}c'_a - \text{₿}x')}$ and $\gamma^\tau \le \frac{\$\mathsf{BV}(\text{Ɇ}c_b - 2\text{Ɇ}x)}{\$\mathsf{BV}(\text{Ɇ}c_b - \text{Ɇ}x)}$. Then, the atomic swap protocol satisfies γ-CSP-fairness.*

Proof. In Lemma 4.2 and Lemma 4.3, 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 $\mathcal{C}$ only consists 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.

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 C2 or C2' is activated. There are two possible cases.

- *Case 1: Both Alice and Bob enter the execution phase.* In this case, Alice always sends pre_a to P1 at Ethereum time T_0, and she never sends any transaction containing pre'_a. Ignoring the negligible probability that $\mathcal{C}$ finds pre'_a by itself, C2' can never be activated. Moreover, once in the execution phase, Alice always sends pre_a to P1 at Ethereum time T_0, and thus Bob will not post any transaction containing pre_b. Ignoring the negligible probability that $\mathcal{C}$ finds pre_b by itself, C2 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_a. Ignoring the negligible probability that $\mathcal{C}$ finds pre_a by itself, C2 and C2' can be activated only by (pre'_a, pre_b). However, Bob always sends _ to P2' and pre_b to P2 at Ethereum time T_0, so Alice never sends any transaction containing pre'_a. Ignoring the negligible probability that $\mathcal{C}$ finds pre'_a by itself, C2 and C2' cannot be activated by (pre'_a, pre_b). To sum up, except the negligible probability, the utility of $\mathcal{C}$ is at most zero, which is the same as the honest case.

□

Theorem 4.5 (Dropout resilience of atomic swap)**.** *Suppose that $H(\cdot)$ is a one-way function and that all players are PPT machines. Our atomic swap protocol is dropout resilient. In other words, suppose at least $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest on either chain; if either Alice or Bob plays honestly but drops out before the end of the protocol, then with $1 - \mathsf{negl}(\lambda)$ probability, the other party's utility must be non-negative.*

Proof. Throughout the proof, for any $X \in \{pre_a, pre_b, pre_c, pre'_a\}$, 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. There are three possible cases.

- *Case 1: Alice drops out before she sends the deposit transaction to* Rapidash'. In this case, Bob will go to the abort phase and send pre_b to P2 at Ethereum time T_0. When τ Ethereum time has passed since P2 is activated, Bob sends _ to C1. Bob will not send any transaction afterward, so P2 and C1 are the only activation points that can be activated. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, Bob's deposit transaction will be confirmed and P2 and C1 will be activated in polynomial time except with negligible probability, and his utility is 0 since he simply gets all his deposit back.

- *Case 2: Alice already sent the deposit transaction to* Rapidash'*, but drops out at or before Ethereum time T_0.* If Bob enters the execution phase, he will send pre_b to P2 at Ethereum time T_1. If Bob enters the abort phase, he will send pre_b to P2 at Ethereum time T_0. In both cases, when τ Ethereum time has passed since P2 is activated, Bob sends _ to C1. Bob will not send any transaction afterward, so P2 and C1 are the only activation points in Rapidash that can be activated. In either case, as long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P2 and C1 must be activated in polynomial time except with negligible probability, and his utility is 0 since he simply gets all his deposit back.

- *Case 3: Alice drops out after Ethereum time T_0.* If Bob enters the execution phase, Alice will send pre_a to P1 at Ethereum time T_0, and Bob will send pre_a to P1' at Ethereum time T_1. Alice and Bob will not send any transaction afterward, so P1 and P1' are the only activation points that can be activated. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P1 and P1' will be activated in polynomial time except with negligible probability, and Bob's utility is $\$\mathsf{BV}(₿x' - Ξx) > 0$.

  On the other hand, if Bob enters the abort phase, Bob will send _ to P2' and pre_b to P2 at Ethereum time T_0. When τ Ethereum time has passed since P2 is activated, Bob sends _ to C1. Alice and Bob will not send any transaction afterwards, so P2 and C1 are the only activation points that can be activated (unless the miners can break the one-wayness of $H(\cdot)$). As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P2 and C1 will be activated in polynomial time except with negligible probability, and his utility is 0 since he simply gets all his deposit back.

Next, We first analyze the case where Bob drops out. There are four possible cases.

- *Case 1: Bob drops out before he sends the deposit transaction to* Rapidash. In this case, Alice does not even send the deposit transaction to Rapidash', so Alice has nothing to lose.

- *Case 2: Bob already sent the deposit transaction to* Rapidash*, but drops out at or before Bitcoin time T'.* In this case, Bob never sends pre_c to P1, so Alice must go to the abort phase. Since Bob drops out at or before Bitcoin time T', he will not send _ to P2'. Then, Alice will send pre'_a to P2' at Ethereum time T_1. When τ' Bitcoin time has passed since P2' is activated, Alice sends _ to C1'. Alice will not send any transaction afterward, so P2' and C1' are the only activation points that can be activated. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P2' and C1' will be activated in polynomial time except with negligible probability, and her utility is 0 since she simply gets all her deposit back.

- *Case 3: Bob drops out after Bitcoin time T' but at or before Ethereum time T_0.* If Bob already sent pre_c to P1, Alice must go to the execution phase. In the execution phase, Alice sends pre_a to P1 at Ethereum time T_0, and she sends _ to P1' as soon as P1 is activated. Alice will not send any transaction afterward, so P1 and P1' are the only activation points that can be activated. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P1 and P1' will be activated in polynomial time except with negligible probability, and her utility is $\$\mathsf{AV}(\text{Ɇ}x - \text{₿}x') > 0$.

  On the other hand, if Bob does not send pre_c to P1 before he drops out out, Alice must go to the abort phase. Then, by the same argument as the previous case, Alice's utility is 0.

- *Case 4: Bob drops out after Ethereum time T_0.* If Bob already sent pre_c to P1, by the same argument as the previous case, Alice's utility is $\$\mathsf{AV}(\text{Ɇ}x - \text{₿}x') > 0$. If Bob does not send pre_c to P1, Alice and Bob must go to the abort phase. In the abort phase, Bob will send _ to P2' at Ethereum time T_0. When τ' Bitcoin time has passed since P2' is activated, Alice sends _ to C1'. Because Alice never sends any transaction containing pre_a, P1' and C2' can never be activated except the negligible probability. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P2' and C1' will be activated in polynomial time except with negligible probability, and Alice's utility is 0.

□

5 Bitcoin Instantiation

In this section we describe how Rapidash can be instantiated in Bitcoin with its limited scripting features.

5.1 Notation and Background

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 $\mathit{Adr} \in \{0,1\}^\lambda$ 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

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

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 RAPIDASH 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 RAPIDASH instantiation only requires some of the most standard scripts used currently in Bitcoin.

We largely rely on the following scripts: (1) computation of hash function[5] $H : \{0,1\}^* \to \{0,1\}^\kappa$, (2) transaction timestamp verification wrt. current height of the blockchain denoted by _NOW[6] 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. [7] We say an address Adr (associated script Φ) is controlled by a user if the user knows a witness x s.t. $\Phi(x) = 1$.

5.2 Instantiating Rapidash Single Instance

We now describe all the transactions, addresses, and scripts needed in our RAPIDASH instantiation of Section 3.

Setup. In RAPIDASH's Bitcoin instantiation, Bob places his deposits into two addresses, the payment address Adr_{pay} and the collateral address Adr_{col}. During the preparation phase, Bob prepares the setup transaction tx_{stp} that moves his coins into the payment address Adr_{pay} and the collateral address Adr_{col}.

Payment redeem. The payment redeem transactions tx_{P1} and tx_{P2} (see Figure 2) redeem from the payment address Adr_{pay}. Transaction tx_{P1} transfers $\$v$ coins to Alice's address while transferring $\$c_b$ coins to Bob's address. On the other hand, we have transaction tx_{P2} that transfers $\$v - \$\epsilon$ coins to Bob's address and $\$\epsilon$ coins to an auxiliary address which will later be used for implementing a conditional timelock redeem. The script Φ_{pay} associated with the address Adr_{pay} provides two ways to redeem (see Figure 1), corresponding to the activation points P1 and P2 in our earlier

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

[6] 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.

[7] 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 1: RAPIDASH's transactions in Bitcoin. Φ_0^B is the script that requires the signature under Bob's public key.

	Description
tx_{stp}	$tx\begin{pmatrix}[(Adr_0^B, \Phi_0^B, \$v + \$c_b)],\\ [(Adr_{\mathsf{pay}}, \Phi_{\mathsf{pay}}, \$v), (Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_b)]\end{pmatrix}$
tx_{P1}	$tx\begin{pmatrix}[(Adr_{\mathsf{pay}}, \Phi_{\mathsf{pay}}, \$v), (Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_b)],\\ [(Adr_1^A, \Phi_1^A, \$v), (Adr_1^B, \Phi_2^B, \$c_b)]\end{pmatrix}$
tx_{P2}	$tx\begin{pmatrix}[(Adr_{\mathsf{pay}}, \Phi_{\mathsf{pay}}, \$v), (Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_b)],\\ [(Adr_1^B, \Phi_1^B, \$v - \$\epsilon), (Adr_{\mathrm{P2}}^{\mathsf{ax}}, \Phi_{\mathrm{P2}}^{\mathsf{ax}}, \$\epsilon)]\end{pmatrix}$
tx_{C1}	$tx\begin{pmatrix}[(Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_b), (Adr_{\mathrm{P2}}^{\mathsf{ax}}, \Phi_{\mathrm{P2}}^{\mathsf{ax}}, \$\epsilon)],\\ [(Adr_3^B, \Phi_3^B, \$c_b + \$\epsilon)]\end{pmatrix}$
tx_{C2}	$tx\begin{pmatrix}[(Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_b)],\\ [(Adr_{\mathsf{burn}}, \Phi_{\mathsf{burn}}, \$c_b - \$v)]\end{pmatrix}$

meta-contract. The two redeem transactions tx_{P1} and tx_{P2} redeem from each of these branches, respectively.

Collateral redeem. The collateral redeem transactions denoted tx_{C1} and tx_{C2} (see Table 1) redeem coins from the collateral address Adr_{col}. The script Φ_{col} associated with the collateral address Adr_{col} provides two ways of redeeming, corresponding to C1 and C2 in our earlier meta-contract, respectively. The transaction tx_{C2} redeems from C2 branch, paying $\$c_b - \$v$ coins to a burn-address Adr_{burn}, and the remaining to the miner who mines the block.

The transaction tx_{C1} redeems from the C1 branch, provided the transaction tx_{P2}, was activated earlier.

To implement the timeout condition associated with the C1 branch, tx_{C1} also refers to an auxiliary input address, $Adr_{\mathrm{P2}}^{\mathsf{ax}}$. This auxiliary address is the output of payment transaction tx_{P2}. Address $Adr_{\mathrm{P2}}^{\mathsf{ax}}$ has a script $\Phi_{\mathrm{P2}}^{\mathsf{ax}}$, that enforce the timeouts.

We provide the list of all transactions in Table 1, the scripts associated with all addresses in Figure 1, and the relationship between the transactions, addresses, and scripts is depicted in Figure 2.

To make sure that Alice and Bob cannot unilaterally spend from the payment address Adr_{pay}, the collateral address Adr_{col}, and the auxiliary address $Adr_{\mathrm{P2}}^{\mathsf{ax}}$, their associated scripts require *signatures from both Alice and Bob* to spend from these addresses. Note also that the transactions tx_{P1} and tx_{P2} needed to spend from P1 and P2 must be signed with different public keys of Alice and Bob, i.e., $\mathsf{pk}_a, \mathsf{pk}_b$, and pk'_a, pk'_b, respectively. This ensures that Bob cannot invoke P1 with tx_{P2} which specifies Bob, rather than Alice, as the recipient address. In summary, assuming security of the signature scheme, no other transaction is able to spend from the addresses Adr_{pay}, Adr_{col}, and the auxiliary address and $Adr_{\mathrm{P2}}^{\mathsf{ax}}$, besides those that they are connected to through a solid line in Figure 2.

5.2.1 Conditional Timelock Redeem and Conditional Burning

We highlight the ideas needed to realize conditional timelock redeem and conditional burning.

Conditional Timelock Redeem. For enforcing a timelock on the collateral coins, general smart contracts let the contracts check if T_2 time has passed since the payment coins were redeemed. In

```
$\Phi_{\mathsf{pay}}(tx, pre_a, pre_b, \sigma_a, \sigma_b)$

P1: if $(H(pre_a) = h_a) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
    then return 1
P2: 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
// Values $h_a, h_b, \mathsf{pk}_a, \mathsf{pk}_b, T_1, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired

$\Phi_{\mathsf{col}}(tx, pre_a, pre_b, \sigma_a, \sigma_b)$

C1: if $(\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
    then return 1
C2: if $(\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1) \wedge (H(pre_a) = h_a) \wedge (H(pre_b) = h_b)$
    then return 1
// Values $h_a, h_b, \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired

$\Phi^{\mathsf{ax}}_{\mathrm{P2}}(tx, \sigma_a, \sigma_b)$

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
// Values $T_2, \mathsf{pk}_a, \mathsf{pk}_b$ are hardwired
```

Figure 1: The description of scripts $\Phi_{\mathsf{pay}}, \Phi_{\mathsf{col}}$ and $\Phi^{\mathsf{ax}}_{\mathrm{P2}}$. Here tx is the transaction spending from the script. Keys pk_a and pk'_a belong to Alice, pk_b and pk'_b belong to Bob.

the absence of general smart contracts, we have to rely on the one-time spendability of Bitcoin addresses. To do this, we rely on the auxiliary address $Adr^{\mathsf{ax}}_{\mathrm{P2}}$ to ensure that the coins in the collateral address Adr_{col} are timelocked for time T_2 after the coins from the payment address Adr_{pay} are redeemed by Bob. Note that $Adr^{\mathsf{ax}}_{\mathrm{P2}}$ is an address created when the payment address is redeemed via tx_{P2} by Bob. $Adr^{\mathsf{ax}}_{\mathrm{P2}}$ is set such that its associated script only allows the coins from the address to be redeemed, if (1) both Alice and Bob sign and (2) time T_2 has passed since $Adr^{\mathsf{ax}}_{\mathrm{P2}}$ was created on the blockchain. By allowing only the transaction tx_{C1} to spend from branch C1 of Adr_{col} and requesting that they *also* spend from $Adr^{\mathsf{ax}}_{\mathrm{P2}}$, we design a mechanism such that for branch C1 the collateral coins in Adr_{col} can be redeemed only if the coins from the auxiliary address $Adr^{\mathsf{ax}}_{\mathrm{P2}}$ are redeemed *simultaneously.* This effectively enforces a timelock of T_2 (after tx_{P2} is published on the blockchain) on the redeeming of the collateral coins by Bob. We refer to this technique of enforcing a conditional timelock on the collateral coins through simultaneous spending of a timelocked auxiliary address as *conditional timelock redeem.* The address $Adr^{\mathsf{ax}}_{\mathrm{P2}}$ only hold a very small amount of value $\$\epsilon$, their only role is to enable the above mechanism. The auxiliary address script $\Phi^{\mathsf{ax}}_{\mathrm{P2}}$ is described in Figure 1.

Conditional burning. The transaction tx_{C2} achieves conditional burning. tx_{C2} transfers $\$c_a + \$c_b - \$v$ coins from Adr_{col} to a burn-address Adr_{burn} (with associated script Φ_{burn} not controlled by anyone[8]), thus leaving $\$v$ coins as transaction fee to any miner mining the transaction into his block. This transaction is set to be valid only if (1) both Alice and Bob have signed it, and (2) the values pre_a and pre_b are attached.

[8]in Bitcoin setting the script Φ_{burn} to be the opcode OP_RETURN makes the coins sent to this address to be unspendable

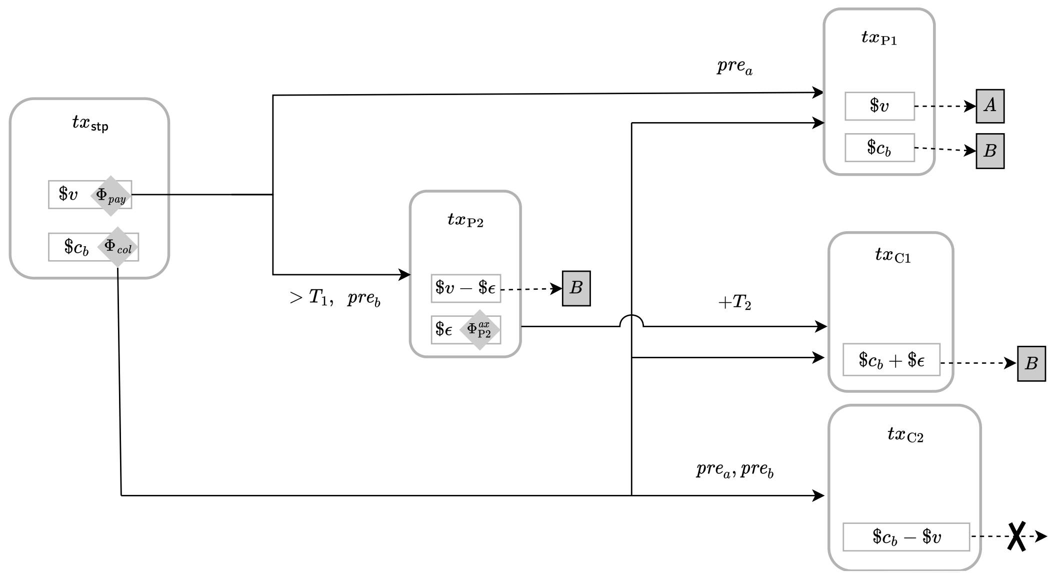

Figure 2: The transaction flow of Rapidash in Bitcoin absent external incentives. Rounded boxes denote transactions, rectangles within are outputs of the transaction. Incoming arrows denote transaction inputs, outgoing arrows denote how an output can be spent by a transaction at the end of the arrow. 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.

5.2.2 Protocol Flow of our Rapidash Instantiation

Before setting up Rapidash 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, including tx_{P1}, tx_{P2}, tx_{C1}, and tx_{C2}. 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_{pay} and Adr_{col}, are not ready yet.

At this moment, Bob reveals his signatures on tx_{stp}. Once tx_{stp} is published on the Bitcoin blockchain, the *execution phase* starts. During the execution phase, either Alice reveals pre_a and publishes transaction tx_{P1} (along with signatures on the transaction), or Bob reveals pre_b and publishes transaction tx_{P2} (along with signatures on the transaction) after T_1 time has passed since publishing tx_{stp}. In the honest run of the protocol, after either of the above redeem paths are published on the blockchain, Bob gets his collateral immediately from tx_{P1} and if not, Bob can redeem the collateral after waiting for time T_2 using either tx_{C1}. If one of the users misbehave, and try to activate both redeem paths, for instance, a strategic Alice reveals pre_a (along with tx_{P1}) when Bob has already revealed pre_b and tx_{P2}, any miner in the system can immediately spend from the C2 branch of ϕ_{col}, and burn the collateral of Alice and Bob while redeeming $\$v$ coins as transaction fee for himself. Note that this is possible, because before setting up Rapidash Alice and Bob had broadcast all redeem transactions and signatures, including the transaction tx_{C2} and their signatures on the transaction. As both pre_a and pre_b are revealed, the miner has enough information to authorize the transaction tx_{C2} on the blockchain, and thus publish the transaction, the signatures (from Alice and Bob), and the values pre_a and pre_b, in *his own* block. He obtains

```
$\Phi_{\mathsf{pay}}(tx, pre_a, pre_b, \sigma_a, \sigma_b)$
P1: if $(H(pre_a) = h_a) \wedge (H(pre_c) = h_c) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
    thenreturn 1
P2: if $(\_\mathrm{NOW} > T_1) \wedge (H(pre_b) = h_b) \wedge (\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1)$
    thenreturn 1
// Values $h_a, h_b, h_c, \mathsf{pk}_a, \mathsf{pk}_b, T_1, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired
$\Phi_{\mathsf{col}}(tx, pre_a, pre_b, \sigma_a, \sigma_b)$
C1: if $(\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
    then return 1
C2: if $(\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1) \wedge$
    $\Big(\big((H(pre'_a) = h'_a) \wedge (H(pre_b) = h_b)\big) \vee \big((H(pre_a) = h_a) \wedge (H(pre_b) = h_b) \wedge (H(pre_c) = h_c)\big)\Big)$
    then return 1
// Values $h'_a, h_a, h_b, h_c, \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired
```

Figure 3: The description of scripts Φ_{pay} and Φ_{col}. Here tx is the transaction spending from the script. Keys $(\mathsf{pk}_a, \mathsf{pk}'_a)$ and $(\mathsf{pk}_b, \mathsf{pk}'_b)$ belong to Alice and Bob, respectively.

$\$v$ coins as fee from the transaction while $\$c_b - \$v$ are burnt.

5.3 Instantiating Atomic Swap: Rapidash and Rapidash' in Bitcoin

In this section we show how we can instantiate both Rapidash and Rapidash' in Bitcoin's scripting language for the atomic swap protocol from Section 4.3. The techniques for the instantiations follow quite closely to the techniques from above.

5.3.1 Instantiating Rapidash in Bitcoin

The only difference between the Rapidash in the atomic swap compared to the single instance Rapidash is in the following transactions: a modified payment redeem transaction tx_{P1}, and a modified collateral redeem transaction tx_{C2}. For ease of understanding, we only explain the concrete modifications.

Transactions. As in the single instance case, we have tx_{stp}, tx_{P2} and tx_{C1}. The first change is in the transaction tx_{P1} which now additionally requires pre_c to be (by Bob) released along with pre_a. This is reflected in the modified P1 branch of the script Φ_{pay} (described in Figure 3).

We modify the collateral redeem transaction tx_{C2} that either requires (pre'_a, pre_b) or (pre_a, pre_b, pre_c) to be released, such that $H(pre'_a) = h'_a, H(pre_a) = h_a, H(pre_b) = h_b$, and $H(pre_c) = h_c$, apart from the signatures from Alice and Bob. This is again reflected in the C2 branch of the Φ_{col} script (described in Figure 3).

A pictorial description of the transaction flow is described in Figure 4.

Protocol Flow. Alice and Bob, proceed as before, where they first agree on the setup transaction tx_{stp} and sign all the payment redeem and collateral redeem transactions $tx_{\mathrm{P1}}, tx_{\mathrm{P2}}, tx_{\mathrm{C1}}$, and tx_{C2} as described above. They broadcast all these transactions and the respective signatures, like before. Finally, Bob signs the setup transaction tx_{stp} and publish it on the blockchain which marks the start of the execution phase. Rest of the protocol flow follows exactly the protocol for the atomic swap.

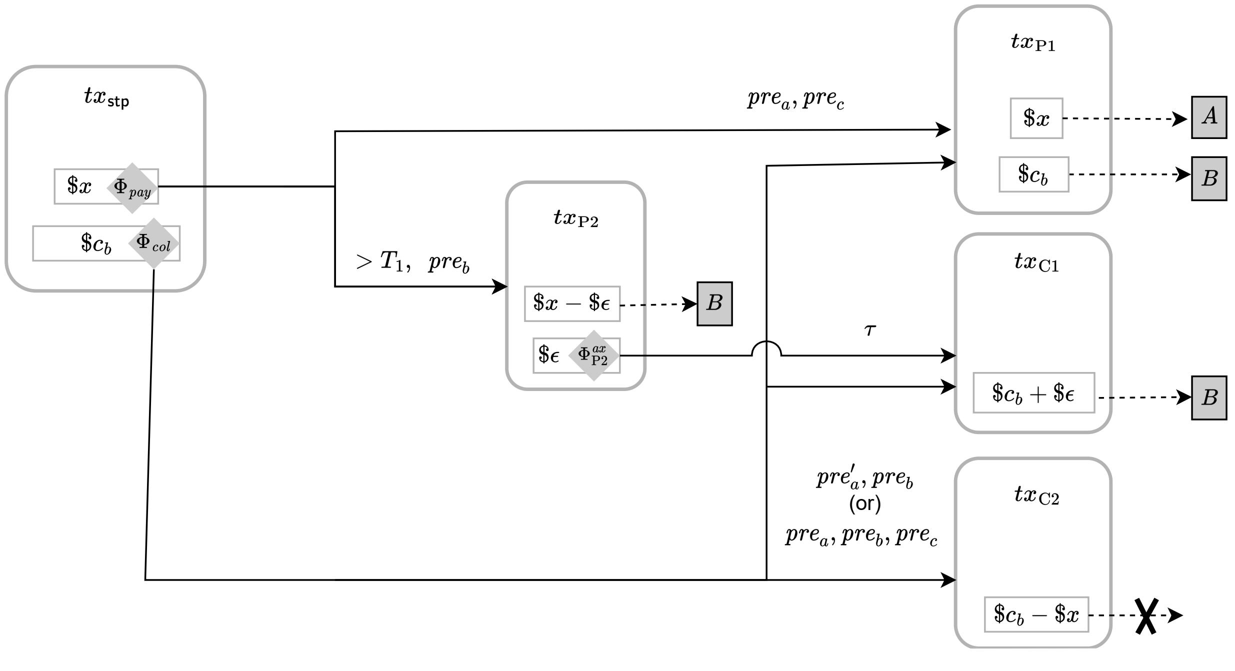

Figure 4: The transaction flow of Rapidash in Bitcoin for atomic swap. Rounded boxes denote transactions, rectangles within are outputs of the transaction. Incoming arrows denote transaction inputs, outgoing arrows denote how an output can be spent by a transaction at the end of the arrow. Solid lines 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).

5.3.2 Instantiating Rapidash' in Bitcoin

We describe all the transactions, addresses and scripts needed in the Rapidash' instantiation for the atomic swap case. Notice that roles of Alice and Bob are reversed compared to Rapidash above. Specifically, in Rapidash', Bob can use pre'_b to retrieve the coins from the payment address, while Alice can use pre'_a after a timeout of T'_1 to retrieve the coins. The main difference between this instantiation and the Rapidash instantiation above is that, in the execution phase both the payment address activation points P1$'$ and P2$'$ can be activated by empty message calls. We also have a modified collateral redeem of the C2$'$ branch of the Φ'_{col} similar in manner to the Rapidash instantiation above.

Transactions. We describe below the different transactions needed for our Rapidash' instantiation.

The setup transaction tx'_{stp} analogous to tx_{stp} creates the two addresses Adr'_{pay} and Adr'_{col}.

We now have two additional payment redeem transactions, $tx^{\mathrm{empty}}_{\mathrm{P1}'}$ and $tx^{\mathrm{empty}}_{\mathrm{P2}'}$ (see Table 2) apart from $tx_{\mathrm{P1}'}$ and $tx_{\mathrm{P2}'}$ that redeem from the payment address Adr'_{pay}. We have the transaction $tx^{\mathrm{empty}}_{\mathrm{P1}'}$ that redeems $\$x'$ coins to Bob's address and $\$c'_a$ coins are paid to Alice's address. The transaction $tx^{\mathrm{empty}}_{\mathrm{P2}'}$ redeems $\$x' - \$\epsilon$ coins to Alice's address and $\$\epsilon$ coins are paid to an auxiliary address. The description of Φ'_{pay} is given below in Figure 5 with Alice and Bob's roles being reversed in Rapidash'. We set the two transactions $tx^{\mathrm{empty}}_{\mathrm{P1}'}$ and $tx^{\mathrm{empty}}_{\mathrm{P2}'}$ to redeem the coins from the (E1)

Table 2: Additional transactions in Bitcoin for Rapidash' atomic swap.

	Description
$tx^{\text{empty}}_{\text{P1}'}$	$tx\begin{pmatrix}[(Adr'_{\mathsf{pay}},\Phi'_{\mathsf{pay}},\$x'),(Adr'_{\mathsf{col}},\Phi'_{\mathsf{col}},\$c'_a)],\\ [(Adr^B_1,\Phi^B_1,\$x'),(Adr^A_1,\Phi^A_1,\$c'_a)]\end{pmatrix}$
$tx^{\text{empty}}_{\text{P2}'}$	$tx\begin{pmatrix}[(Adr'_{\mathsf{pay}},\Phi'_{\mathsf{pay}},\$x')],\\ [(Adr^A_2,\Phi^A_2,\$x'-\$\epsilon),(Adr^{\mathsf{ax}}_{\text{P2}'},\Phi^{\mathsf{ax}}_{\text{P2}'},\$\epsilon)]\end{pmatrix}$
$tx^{\text{empty}}_{\text{C1}'}$	$tx\begin{pmatrix}[(Adr'_{\mathsf{col}},\Phi'_{\mathsf{col}},\$c'_a),(Adr^{\mathsf{ax}}_{\text{P2}'},\Phi^{\mathsf{ax}}_{\text{P2}'},\$\epsilon)],\\ [(Adr^A_3,\Phi^A_3,\$c'_a+\$\epsilon)]\end{pmatrix}$

```
$\Phi'_{\mathsf{pay}}(tx, pre'_a, pre'_b, \sigma_a, \sigma_b)$
    P1': if $(H(pre'_b) = h'_b) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
     then return 1
    P2': if $(\_\mathrm{NOW} > T'_1) \wedge (H(pre'_a) = h'_a) \wedge (\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1)$
        then return 1
    E1: if $(\mathsf{Vf}(\mathsf{pk}''_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}''_b, tx, \sigma_b) = 1)$ then return 1
    E2: if $(\_\mathrm{NOW} > T'_1) \wedge (\mathsf{Vf}(\mathsf{pk}'''_a, tx, \sigma_b) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'''_b, tx, \sigma_b) = 1)$ then return 1
    // Values $h'_a, h'_b, \mathsf{pk}_a, \mathsf{pk}_b, T_1, \mathsf{pk}'_a, \mathsf{pk}'_b, \mathsf{pk}''_a, \mathsf{pk}''_b, \mathsf{pk}'''_a, \mathsf{pk}'''_b$ are hardwired
$\Phi'_{\mathsf{col}}(tx, pre'_a, pre'_b, \sigma_a, \sigma_b)$
    C1: if $(\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
        then return 1
    C2: if $(\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1) \wedge$
        $\Big(\big((H(pre'_a) = h'_a) \wedge (H(pre_b) = h_b)\big) \vee \big((H(pre'_a) = h'_a) \wedge (H(pre'_b) = h'_b)\big)\Big)$
        then return 1
    // Values $h'_a, h'_b, h_b, \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{pk}'_a, \mathsf{pk}'_b$ are hardwired
```

Figure 5: The description of scripts Φ'_{pay} and Φ'_{col}.

and (E2) branches, respectively. These transactions will correspond to the empty message calls to the Rapidash' contract in activation points P1$'$ and P2$'$, respectively. The script Φ'_{col} is similar Φ_{col} in Rapidash except that in the C2$'$ branch, we require either (pre'_a, pre'_b) or (pre'_a, pre_b) along with the signatures of Alice and Bob. Similarly the script $\Phi^{\mathsf{ax}}_{\text{P2}'}$ of the auxiliary addresses are the same as $\Phi^{\mathsf{ax}}_{\text{P2}}$ from Rapidash, except we replace the timeout value τ with τ'.

In addition to the collateral redeem transaction $tx_{\text{C1}'}$ we have the transaction $tx^{\text{empty}}_{\text{C1}'}$ and a modified $tx_{\text{C2}'}$ (see Table 2). The new transaction $tx^{\text{empty}}_{\text{C1}'}$ redeem coins from the C1$'$ activation point, depending on whether transaction $tx^{\text{empty}}_{\text{P2}'}$ was activated earlier. Similar to the Rapidash instantiation, C1$'$ branch timeout of τ' for $tx^{\text{empty}}_{\text{C1}'}$ is implemented via the auxiliary addresses created by $tx^{\text{empty}}_{\text{P2}'}$. Finally the transaction $tx_{\text{C2}'}$ is modified to require either (pre'_a, pre'_b) or (pre'_a, pre_b), apart from the signatures from Alice and Bob. Intuitively, this transaction is set to activate the modified C2$'$ branch of Φ'_{col}. A pictorial description of the transaction flow for payment and

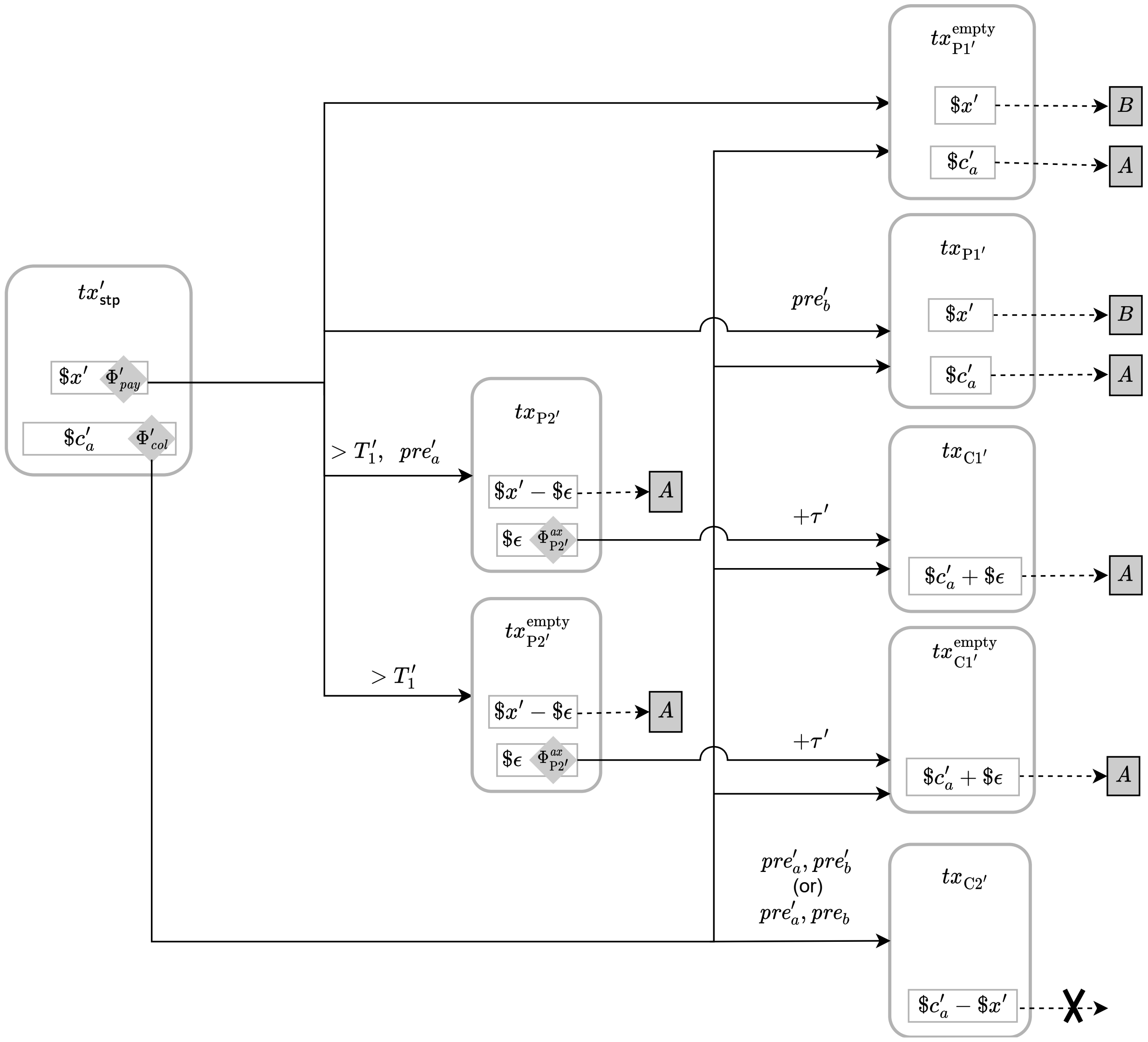

Figure 6: The transaction flow of Rapidash' in Bitcoin for atomic swap. Rounded boxes denote transactions, rectangles within are outputs of the transaction. Incoming arrows denote transaction inputs, outgoing arrows denote how an output can be spent by a transaction at the end of the arrow. Solid lines 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 τ') associated with a transaction output is written over the corresponding outgoing arrow.

collateral redeem is given in Figure 6.

Protocol Flow. Alice and Bob, first agree on the setup transaction tx'_{stp} and sign the redeeming transactions $tx_{\text{P1}'}, tx_{\text{P2}'}, tx_{\text{C1}'}, tx_{\text{C2}'}$, and $tx^{\text{empty}}_{\text{C1}'}$. They broadcast all these transactions and the respective signatures, like before. However, this time Alice and Bob sign the transaction $tx^{\text{empty}}_{\text{P1}'}$ such that only Alice has both signatures. She does not broadcast the signatures and keeps it privately. Similarly, Alice and Bob sign the transaction $tx^{\text{empty}}_{\text{P2}'}$ such that only Bob has both signatures. He keeps them privately and does not broadcast them. Notice that none of the transaction can be published on the blockchain yet as the setup transaction is not yet published. Finally, they sign the setup transaction tx'_{stp} and publish it on the blockchain, thus starting the execution phase.

Whenever Alice wishes to activate P1$'$ in Rapidash' with an empty message, she publishes the transaction $tx^{\text{empty}}_{\text{P1}'}$ along with the valid signatures she has in her possession. Similarly, whenever Bob wishes to activate P2$'$ in Rapidash' with an empty message, he publishes the transaction $tx^{\text{empty}}_{\text{P2}'}$ along with the valid signatures he has in his possession. If $tx^{\text{empty}}_{\text{P2}'}$ is published on the blockchain, activation point C1$'$ can be activated by either $tx^{\text{empty}}_{\text{C1}'}$ after a timeout of τ' time units. Conditional burning via tx'_{C2} activates C2$'$ if the parties misbehave, which proceeds exactly as the description of the atomic swap protocol.

References

[BBSU12] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better – how to make bitcoin a better currency. In *Financial Cryptography and Data Security (FC)*, 2012.

[BDM] Wacław Banasik, Stefan Dziembowski, and Daniel Malinowski. Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In *Computer Security – ESORICS 2016*.

[Bis] Bryan Bishop. Bitcoin vaults with anti-theft recovery/clawback mechanisms. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2019-August/017231.html.

[BK] Sergiu Bursuc and Steve Kremer. Contingent payments on a public ledger: Models and reductions for automated verification. In *ESORICS 2019*.

[Bon] Joseph Bonneau. Why buy when you can rent? - bribery attacks on bitcoin-style consensus. In *Financial Cryptography Workshops 2016*.

[CCWS21] Kai-Min Chung, T-H. Hubert Chan, Ting Wen, and Elaine Shi. Game-theoretic fairness meets multi-party protocols: The case of leader election. In *CRYPTO*. Springer-Verlag, 2021.

[CGGN] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *ACM CCS 2017*.

[CGL$^+$18] Kai-Min Chung, Yue Guo, Wei-Kai Lin, Rafael Pass, and Elaine Shi. Game theoretic notions of fairness in multi-party coin toss. In *TCC*, volume 11239, pages 563–596, 2018.

[CMST22] Hao Chung, Elisaweta Masserova, Elaine Shi, and Sri AravindaKrishnan Thyagarajan. Ponyta: Foundations of side-contract-resilient fair exchange. Cryptology ePrint Archive, Paper 2022/582, 2022. https://eprint.iacr.org/2022/582.

[DEFM19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *IEEE Symposium on Security and Privacy*, 2019.

[DFH18] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *ACM CCS*, CCS '18, page 949–966, 2018.

[DW15] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems*, 2015.

[Fuc] Georg Fuchsbauer. Wi is not enough: Zero-knowledge contingent (service) payments revisited. In *ACM CCS 2019*.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[GM] Matthew Green and Ian Miers. Bolt: Anonymous payment channels for decentralized currencies. In *ACM CCS 2017*.

[Ham] Matthew Hammond. Blockchain interoperability series: Atomic swaps. https://medium.com/@mchammond/atomic-swaps-eebd0fa8110d.

[Her18] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, page 245–254, New York, NY, USA, 2018. Association for Computing Machinery.

[HZ20] Jona Harris and Aviv Zohar. Flood & loot: A systemic attack on the lightning network. In *AFT*, 2020.

[JMM14] Danushka Jayasinghe, Konstantinos Markantonakis, and Keith Mayes. Optimistic fair-exchange with anonymity for bitcoin users. In *2014 IEEE 11th International Conference on e-Business Engineering*, pages 44–51, 2014.

[JSZ$^+$21] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gazi, Sarah Meiklejohn, and Edgar Weippl. Pay to win: Cheap, crowdfundable, cross-chain algorithmic incentive manipulation attacks on pow cryptocurrencies. In *FC WTSC*, 2021.

[KMSW22] Ilan Komargodski, Shin'ichiro Matsuo, Elaine Shi, and Ke Wu. $\log^*$-round game-theoretically-fair leader election. 2022.

[Max] Gregory Maxwell. The first successful zero-knowledge contingent payment. https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/.

[MBB$^+$] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In *Financial Cryptography 2019*.

[MD19] Mahdi H. Miraz and David C. Donald. Atomic cross-chain swaps: Development, trajectory and potential of non-monetary digital token swap facilities. In *Annals of Emerging Technologies in Computing (AETiC)*, 2019.

[MES16] Malte Möser, Ittay Eyal, and Emin Gün Sirer. Bitcoin covenants. In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2016.

[MHM18] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *Financial Cryptography Workshops*, 2018.

[MMA] Patrick McCorry, Malte Möser, and Syed Taha Ali. Why preventing a cryptocurrency exchange heist isn't good enough. In *Security Protocols Workshop 2018.*

[MMS$^+$] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS 2019.*

[MMSH] Patrick Mccorry, Malte Möser, Siamak F. Shahandasti, and Feng Hao. Towards bitcoin payment networks. In *Proceedings, Part I, of the 21st Australasian Conference on Information Security and Privacy - Volume 9722, 2016.*

[PD] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`.

[PS17a] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, 2017.

[PS17b] Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *CSF*, pages 115–129. IEEE Computer Society, 2017.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 643–673, 2017.

[TYME21] Itay Tsabary, Matan Yechieli, Alex Manuskin, and Ittay Eyal. MAD-HTLC: because HTLC is crazy-cheap to attack. In *IEEE Symposium on Security and Privacy*, pages 1230–1248. IEEE, 2021.

[vdM19] Ron van der Meyden. On the specification and verification of atomic swap smart contracts. In *IEEE ICBC*, 2019.

[WAS22] Ke Wu, Gilad Asharov, and Elaine Shi. A complete characterization of game-theoretically fair, multi-party coin toss. In *Eurocrypt*, 2022.

[WHF19] Fredrik Winzer, Benjamin Herd, and Sebastian Faust. Temporary censorship attacks in the presence of rational miners. In *IEEE European Symposium on Security and Privacy Workshops*, pages 357–366, 2019.

[WSZN22] Sarisht Wadhwa, Jannis Stoeter, Fan Zhang, and Kartik Nayak. He-htlc: Revisiting incentives in htlc. Cryptology ePrint Archive, Paper 2022/546, 2022. `https://eprint.iacr.org/2022/546`.

[ZHL$^+$19] A Zamyatin, D Harz, J Lind, P Panayiotou, A Gervais, and W Knottenbelt. Xclaim: trustless, interoperable, cryptocurrency-backed assets. In *IEEE S& P*, 2019.