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 responsiveness; 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,MAA,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,MAA,Bis,ZHL+19]. Recently, the community is increasingly concerned that potential user-miner collusion can completely break the fairness guarantees promised by fair exchange protocols [TYME21,WHF19,Bon,MMS+,MHH18,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 $Bx'$) for Bob's $x'$ amount of Ethers (henceforth denoted $Ex'$).

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., $Bx'$ and $Ex'$, 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. 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 $\lambda$.

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

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 $H\mathcal{S}_C$ or $H\mathcal{S}_{-C}$ to denote the honest strategy executed by either the coalition $\mathcal{C}$ or its complement. Let $S_C$ and $S'_{-C}$ be the strategies of the coalition $\mathcal{C}$ and its complement. We use $\text{util}^C(S_C, S'_{-C})$ to denote the expected utility of $\mathcal{C}$ when the coalition $\mathcal{C}$ adopts the strategy $S_C$ and the remaining parties adopt the strategy $S'_{-C}$.

**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 $\gamma$-cooperative-strategy-proofness (or $\gamma$-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 $S_C$ of $\mathcal{C}$, there exists a negligible function $\text{negl}(\cdot)$ such that

$$\text{util}^C(S_C, H\mathcal{S}_{-C}) \leq \text{util}^C(H\mathcal{S}_C, H\mathcal{S}_{-C}) + \text{negl}(\lambda)$$

where we use $H\mathcal{S}$ 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 $1/\text{poly}(\lambda)$ fraction of the mining power is honest, except with $1 - \text{negl}(\lambda)$ 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 $1/\text{poly}(\lambda)$ 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 $\text{pre}_a$ and Bob offers to pay Alice $v$ amount of coins in exchange for the secret. For example, $\text{pre}_a$ 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 $\text{pre}_a$ is worth $v_a$ and $v_b$ to Alice and Bob, respectively. That is, Alice will lose utility $v_a$ if $\text{pre}_a$ is released to someone else, and Bob will gain $v_b$ if he learns $\text{pre}_a$. We assume that $v_b > v > v_a$, such that Alice wants to sell the secret $\text{pre}_a$ 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 $\text{pre}_a$ 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 deploying the RAPIDASH contract, Alice and Bob sample $\text{pre}_a \in \{0, 1\}^\lambda$ and $\text{pre}_b \in \{0, 1\}^\lambda$ 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 $(h_a, h_b, T_1, T_2, v, c_b, \epsilon)$ of the RAPIDASH contract must satisfy the following constraints.

**Parameter Constraints for Rapidash**

- $h_a = H(\text{pre}_a)$ and $h_b = H(\text{pre}_b)$.
- $0 < \epsilon < v$, $v < c_b > \epsilon$.
- $T_1 \geq 1$ and $T_2 \geq 1$ such that $\gamma T_2 \leq \frac{c_b}{c_b + v}$.

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 $c_b = v$. Then, we need to ensure $\gamma T_2 \leq 1/2$. This means if $\gamma = 90\%$, we can set $T_2 = 7$; and if $\gamma = 49.9\%$, we can set $\tau = 1$. 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 follows.

#### RAPIDASH contract

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

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

**Execution phase:**

**Payment:**

P1: On receive $\text{pre}_a$ from Alice such that $H(\text{pre}_a) = h_a$, send $v$ to Alice and $c_b$ to Bob.

P2: Time $T_1$ or greater: on receive $\text{pre}_b$ from Bob such that $H(\text{pre}_b) = h_b$, do nothing.

**Collateral:**

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

C2: On receive $(\text{pre}_a, \text{pre}_b)$ from anyone $P$ such that $H(\text{pre}_a) = h_a$ and $H(\text{pre}_b) = h_b$, send $\epsilon$ 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 $\text{pre}_a$ to activation point P1 at $t = 0$.

- **Bob:** If Alice failed to send $\text{pre}_a$ to P1 before time $T_1$, then Bob sends $\text{pre}_b$ 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\textsubscript{a} value included in the corresponding transaction. If P2 and C1 are successfully activated, Bob outputs ⊥.

- **Miner**: The miner watches all transactions posted to P1, P2, and C2. If the miner has observed the correct values of both pre\textsubscript{a} and pre\textsubscript{b} from these posted transactions, then it sends (pre\textsubscript{a}, pre\textsubscript{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).

**Intuition.** We now explain the intuition behind our construction.

- In addition to Bob’s intended payment $v$, he has to put down additional collateral $c_b$ into the contract.

- The contract has four activation points named P1, P2, C1, and C2. P1 is the normal path. When Alice, Bob, and the miners all act honestly, the exchange can be completed and both users get their collateral back as soon as the next block is mined. Clearly, once P1 is activated, the contract holds no more money and thus none of the remaining activation points can be activated.

- Should Alice drop out the protocol, Bob can get his deposit back in the following way: first, Bob posts pre\textsubscript{b} to P2 to express his intention to cancel the exchange; however, when P2 is activated, Bob cannot immediately get his deposit back yet. Only after a timeout $T_2$ has passed, can Bob finally get all of his deposit $v + c_b$ refunded by posting an empty message ⊥ to C1. This timeout $T_2$ is important for achieving fairness as we explain shortly.

- Finally, the activation point C2 is also called the “bomb”. Should the honest Alice post pre\textsubscript{a} to P1 and Bob still tries to get his money back by posting pre\textsubscript{b} to P2 and later invoking C1, then both secrets pre\textsubscript{a} and pre\textsubscript{b} will be publicly known. At this moment, any miner can post the pair (pre\textsubscript{a}, pre\textsubscript{b}) to C2 (i.e., trigger the bomb) to earn a positive amount $\epsilon$. At the same time, the rest of Bob’s deposit will be burnt.

We shall prove that our protocol satisfies CSP-fairness as long as the collateral $c_b$ and the timeout $T_2$ are suitably large. As mentioned earlier, we need the following two constraints to hold, and we explain the intuition below.

- $c_b > \epsilon$: this condition makes sure that a sufficient amount is burnt should the bomb C2 be triggered, such that if a Bob-miner coalition activates P2 and C2 at the same time (e.g., in the same block it mines), it will lose to the honest case.

- $\frac{c_b}{c_b + x} \leq T_2$: if the honest Alice posts pre\textsubscript{a} to P1, this condition makes sure that it is not worth it for the Bob-miner coalition to take a gamble and try to invoke both P2 and C1 to get all of Bob’s deposit back. Basically, once the Bob-miner coalition has invoked P2, both pre\textsubscript{a} and pre\textsubscript{b} become publicly known. So unless the coalition can mine all blocks within the next $T_2$ window, it cannot invoke C1. This is because any non-colluding miner who mines a block during this $T_2$ window will trigger the bomb C2 in which case Bob’s collateral will be burnt.
### 3.3 Proofs

**Lemma 3.1 (Alice-miner coalition).** Let $C$ be any coalition that consists of Alice and an arbitrary subset of miners\(^2\) (possibly no miner). Then, for any (even unbounded) coalition strategy $S_C$,

$$\text{util}^C(S_C, HS_{-C}) \leq \text{util}^C(HS_C, HS_{-C})$$

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

**Proof.** When the coalition $C$ 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 $C$ is $v - v_a$.

Now, consider the case that the coalition $C$ 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 $C$ will not invoke the smart contract. There are two possibilities:

- First, P1 is activated at some point. In this case, nothing else can be activated. Thus, the utility of $C$ is $v - v_a$, which is the same as the honest case.

- Second, P1 is never activated. The Alice-miner coalition cannot cash out from P2 or C1, it can only cash out $\epsilon$ from C2. However, when C2 is activated, pre$_a$ is publicly known, so the utility of $C$ is $v - v_a$, which is less than the honest case since $\epsilon < v$.

\(\square\)

**Lemma 3.2 (Bob-miner coalition).** Let $C$ be any coalition that consists of Bob and a subset of miners controlling at most $\gamma$ fraction of mining power. Then, as long as $\gamma^{T_2} \leq \frac{c_b}{v_b + c_b}$, for any (even unbounded) coalition strategy $S_C$, it must be that

$$\text{util}^C(S_C, HS_{-C}) \leq \text{util}^C(HS_C, HS_{-C})$$

**Proof.** The honest Alice will always send $pre_a$ to P1. Thus, when $C$ follows the protocol, P1 will be activated in the next block, and the utility of $C$ is $v_b - v$.

Now, suppose $C$ 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 nor P2 is activated. Because P2 is not activated, C1 cannot be activated. The Bob-miner coalition can only get $\epsilon$ from C2. Thus, the coalition’s utility is at most $v_b - v - c_b + \epsilon < v_b - v$ where the inequality is due to the constraint $c_b > \epsilon$.

- Second, P1 is activated. In this case, nothing else can be activated, and the utility of $C$ is $v_b - v$, which the same as the honest case.

\(\square\)

\(^2\) 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 $C$ itself could be facilitated by smart contracts, our modeling of coalition already captures any arbitrary side contract within the coalition.
• 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 \( \epsilon \) from C2 at time \( t^* \). In this case, the coalition’s utility is at most \( v_b - c_b - v + \epsilon \), and since \( c_b > \epsilon \), this is less than the honest case. Henceforth, we may assume that the coalition does not invoke C2 also at time \( t^* \). Since the honest Alice posts \( \text{pre}_a \) at \( t = 0 \) and \( t^* \geq T_1 \), both \( \text{pre}_a \) and \( \text{pre}_b \) 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 mine 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 \( v_b - c_b - v \). Else if it wins the race, then the coalition’s utility is at most \( v_b \). The probability \( p \) that the coalition wins the race is upper bounded by \( p \leq \gamma T_2 \). Therefore, the coalition’s expected utility is at most

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

For \( (v_b - c_b - v) \cdot (1 - p) + v_b \cdot p \) to exceed the honest utility \( v_b - v \), it must be that

\[
p > \frac{v_b - v}{v_b - c_b + v}
\]

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 \( \gamma T_2 \leq \frac{v_b - v}{v_b - c_b + v} \). Then, the Rapidash protocol satisfies \( \gamma \)-CSP-fairness.

Proof. Lemmas 3.1 and 3.2 proved \( \gamma \)-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 \( \gamma \)-CSP-fairness for the case when the coalition consists only of some miners whose mining power does not exceed \( \gamma \). 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 \( \text{pre}_b \) 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 \( 1/\text{poly}(\lambda) \) 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 \( 1 - \text{negl}(\lambda) \) probability, the other party’s utility should be non-negative.

Proof. Throughout the proof, for any \( X \in \{ \text{pre}_a, \text{pre}_b \} \), 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 \( \text{pre}_a \); 2) Alice drops out after she already posted a transaction containing \( \text{pre}_a \) at \( t = 0 \). In the first case, as long as \( 1/\text{poly}(\lambda) \) 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 \( \text{pre}_b \) to P2. An honest miner would include Alice’s transaction and activate P1. As long as \( 1/\text{poly}(\lambda) \) 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 \( \text{pre}_a \), 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 $v + \delta$ for an arbitrarily small $\delta$. 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 $v + \delta$ 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 $E_x$, and Alice as $x'$ amount of Bitcoins denoted $B_{x'}$. Bob wants to exchange his $E_x$ with Alice’s $B_{x'}$. 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 $\mathcal{AV}(\cdot)$ to denote the valuation function of Alice of an Alice-miner coalition; specifically, $\mathcal{AV}(E_x + B_{x'}) = v_a \cdot x + v'_{a'} \cdot x'$ 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 $\mathcal{BV}(\cdot)$ to denote the valuation function of Bob or a Bob-miner coalition, and we use $\mathcal{MV}(\cdot)$ 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 $B_{x'}$ with Bob for $E_x$, and vice versa.

**Assumption:** $\mathcal{AV}(E_x - B_{x'}) > 0, \mathcal{BV}(B_{x'} - E_x) > 0$

**Utility.** Let $d_a, E_a \geq 0$ be the cryptocurrencies that Alice or an Alice-miner coalition deposit into the smart contracts. Let $r_a, E_a \geq 0$ 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:

$$u_a = \mathcal{AV}(r_a - d_a + E_a - E_a)$$
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:

$$u_b = BV(B_r' - B_d' + E_r - E_d)$$

$$u_m = MV(B_r' - B_d' + E_r - E_d)$$

where $B_r', E_r ≥ 0$, denote the payment the Bob-miner coalition or Bob receives during the protocol, and $B_d', E_d ≥ 0$ denote the deposit the Bob-miner coalition or Bob sends to any smart contract during the protocol. The variables $B_r', E_r, B_d', E_d ≥ 0$ 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 $\lambda$.

**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 $Bx' + Bc_a'$ into RAPIDASH' where $Bx'$ denotes the intended payment and $Bc_a'$ denotes Alice’s collateral. Bob then deposits $Ex + Ec_b$ into RAPIDASH where $Ex$ denotes the intended payment and $Ec_b$ 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 $Ex$ and Bob gets back his collateral $Ec_b$. 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 $Bx'$ and Alice would get back her collateral $Bc_a'$. 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 $Ex$ and $Ec_b$ back. Similarly, if Bob fails to post $pre_a'$ to P1’ of RAPIDASH' on time, then Alice can get her intended payment $Bx'$ and $Bc_a'$ 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 $Ex$. This way, the coalition basically obtained $Ex$ 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 $\mathcal{E}x$ 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 $\mathcal{E}c'_a$ as collateral besides her intended payment to Bob $\mathcal{E}x'$; similarly, Bob needs to have an extra $\mathcal{E}c_a$ as collateral besides his intended payment $\mathcal{E}x$ 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):**

- \( h_a = H(\text{pre}_a), \ h_b = H(\text{pre}_b) \) and \( h_c = H(\text{pre}_c) \).
- \( T_1 > T_0 > T > 0 \).
- \( E_0 < E_\epsilon < E_x, \ E_c_b > E_\epsilon \)

**Constraints for Rapidash’ (on Bitcoin):**

- \( h'_b = H(\text{pre}_a) = h_a \) and \( h'_a = H(\text{pre}'_a) \).
- 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 \( T_1 \).
- \( E_0 < E_\epsilon' < E_x', \ E_{c_b}' > E_\epsilon' \)

**Choice of timeouts:**

\[ \tau \geq 1, \ \tau' \geq 1. \]
\[ \gamma \tau \leq \frac{E_{c_b}}{E_{c_b} + E_x}, \ \gamma \tau' \leq \frac{E_{c_b}'}{E_{c_b}' + E_x} \]

\(^a\)In practice, this constraint should be respected except with negligible probability despite 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 \( \Delta \) denote the network delay. Then, \( T_1 - T_0, T_0 - T, T \) amount of Ethereum time should all be larger than \( \Delta \). 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 \( \Delta \). 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 \( \tau \) and \( \tau' \) must be sufficiently large w.r.t. \( \gamma \), 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 \( E_{c_b} = E_x \). Then, we need to make sure \( \gamma \tau \leq 1/2 \). This means if
\( \gamma = 90\% \), we can set \( \tau = 7 \); if \( \gamma = 49.9\% \), we can set \( \tau = 1 \). Asymptotically, for any \( \gamma = O(1) \), \( \tau \) is a constant. Increasing \( \xi_{cb} \) helps to make \( \tau \) smaller. A similar calculation also works for \( \tau' \) and \( \xi'_{ca} \).

**Contracts.** We now describe the contracts for both chains.

<table>
<thead>
<tr>
<th>Rapidash contract (on Ethereum)</th>
<th>// Parameters: ((a, b, T, \xi, \xi_{cb}, \xi_{e}))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preparation phase:</strong> Bob deposits ( \xi x + \xi_{cb} ).</td>
<td></td>
</tr>
<tr>
<td><strong>Execution phase:</strong></td>
<td></td>
</tr>
<tr>
<td>P1: On receive ( pre_a ) from Alice and ( pre_c ) from Bob such that ( H(pre_a) = a ) and ( H(pre_c) = c ), send ( \xi x ) to Alice and ( \xi_{cb} ) to Bob.</td>
<td></td>
</tr>
<tr>
<td>P2: Time ( T ) or greater: On receive ( pre_b ) from Bob such that ( H(pre_b) = b ), do nothing.</td>
<td></td>
</tr>
<tr>
<td>C1: At least ( \tau ) after P2 is activated: on receiving ( \xi ) from anyone, send ( \xi x + \xi_{cb} ) to Bob.</td>
<td></td>
</tr>
<tr>
<td>C2: On receive ( (pre_a, pre_b, pre_c) ) or ( (pre'_a, pre'_b) ) from anyone ( P ) such that ( H(pre_a) = a ), ( H(pre_b) = b ), ( H(pre_c) = c ), and ( H(pre'_a) = a' ), send ( \xi e ) to player ( P ). All remaining coins are burnt.</td>
<td></td>
</tr>
</tbody>
</table>

\[
\xi_{cb} = 90\%, \text{ we can set } \xi_{cb} = 0.|
\]

<table>
<thead>
<tr>
<th>Rapidash’ contract (on Bitcoin)</th>
<th>// Parameters: ((a, b, T', \xi', \xi'<em>{cb}, \xi'</em>{e}))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Preparation phase:</strong> Alice deposits ( \xi' x' + \xi'_{cb} ).</td>
<td></td>
</tr>
<tr>
<td><strong>Execution phase:</strong></td>
<td></td>
</tr>
<tr>
<td>P1’: On receiving ( pre'_b ) from Bob(^a) such that ( H(pre'<em>b) = b ) or on receiving ( \xi' ) from Alice, send ( \xi' x' ) to Bob and send ( \xi'</em>{cb} ) to Alice.</td>
<td></td>
</tr>
<tr>
<td>P2’: Time ( T' ) or greater: on receiving ( pre'_a ) from Alice such that ( H(pre'_a) = a ) or on receiving ( \xi' ) from Bob, do nothing.</td>
<td></td>
</tr>
<tr>
<td>C1’: At least ( \tau' ) after P2’ is activated: on receiving ( \xi' ) from anyone, send ( \xi' x' + \xi'_{cb} ) to Alice.</td>
<td></td>
</tr>
<tr>
<td>C2’: On receiving ( (pre'_b, pre'_a) ) or ( (pre'_a, pre'_b) ) from anyone ( P ) such that ( H(pre'_b) = b ), ( H(pre'_a) = a ), and ( H(pre'_a) = a' ), send ( \xi' e ) to player ( P ). All remaining coins are burnt.</td>
<td></td>
</tr>
</tbody>
</table>

\(^a\)Bob will let \( pre'_a \) 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 \( pre_a = pre'_a \), \( pre_b \), \( pre'_a \), 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 $\mathcal{E}x + \mathcal{E}c_b$ 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 $\mathcal{E}x' + \mathcal{E}c'_a$ 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', Bob sends $pre'_a$ to P2'.
3. If $\tau'$ Bitcoin time has passed since P2' is activated, Alice sends $\_\_\_$ to C1'; similarly, if $\tau$ 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 $C$ be any coalition that consists of Alice and an arbitrary subset of miners controlling at most $\gamma$ fraction of mining power. Then, as long as $\gamma' \leq \frac{Bc'_a + Bx'}{Bc'_a + Bx'}$, for any PPT coalition strategy $S_C$,\[
\text{util}^C(S_C, HS_{-C}) \leq \text{util}^C(HS_C, HS_{-C})
\]
where $HS_{-C}$ denotes the honest strategy for everyone not in $C$.\]
Proof. When the coalition $C$ follows the protocol, $P1$ and $P1'$ will be activated, and the utility of $C$ is $\text{SAV}(E x - B x') > 0$. Now, suppose $C$ 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 $\text{pre}_c$ 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 $\text{pre}_c$ to $P1$, so except with the negligible probability that $C$ finds $\text{pre}_c$ by itself, $P1$ and $C2$ can never be activated. $C$ can earn coins from RAPIDASH only through $P1$ and $C2$. Thus, except with the negligible probability, the utility of $C$ is at most zero, which is less than the honest case.

**Bob enters the execution phase**. $C$ can earn coins from RAPIDASH either from $P1$ or from $C2$ but not both, and in either case, $C$ gains at most $E x$ since $E x < E x$. Thus, if $P1'$ is activated (in which case no other activation points of RAPIDASH’ can be activated), the utility of $C$ cannot be more than the honest case. Henceforth we assume $P1'$ is not activated. If $P2'$ is also not activated, then only $C2'$ can be activated in which case the coalition gets $B c' < B x'$ — in this case the coalition’s utility must be smaller than the honest case.

Therefore, for the coalition’s utility to exceed the honest case, the only possibility is that $P2' + C1'$ are both activated. Suppose that $P2'$ is activated at Bitcoin time $t^* \geq T_1'$. If Bob enters the execution phase, $P2'$ can only be activated by Alice posting $\text{pre}_a'$ to $P2'$. So $\text{pre}_a'$ is publicly known after Bitcoin time $t^*$. Depending on whether Alice sends $\text{pre}_a$ to $P1$ before Ethereum time $T_1$, there are two subcases.

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

$$\text{SAV}((E x - B x' - B c_a') \cdot (1 - p) + E x \cdot p).$$

(1)

Since $p \leq \gamma r' \leq \frac{B c_a'}{B c_a' + B x'}$, we have

$$\text{SAV}((E x - B x' - B c_a') \cdot (1 - p) + E x \cdot p) < \text{SAV}(E x - B x').$$

- **Case 2**: Alice does not send $\text{pre}_a$ to $P1$ before Ethereum time $T_1$. In this case, the honest Bob will send $\text{pre}_b$ to $P2$ at Ethereum time $T_1$. Because $P2'$ is activated at Bitcoin time $t^* \geq T_1'$, which is later than Ethereum time $T_1$, $\text{pre}_a'$ and $\text{pre}_b$ are both publicly known at Bitcoin time $t^*$. Thus, during Bitcoin time $(t^*, t^* + \tau')$, any miner in $-C$ will activate $C2'$ if it wins a block. As before, we say $C$ loses the race if a non-colluding miner mines a new block during Bitcoin time $(t^*, t^* + \tau')$ and $C$ wins the race otherwise. If $C$ loses the race, it gets nothing from $C1'$ or $C2'$, and it gets at most $\text{SAV}(E x - B x' - B c_a')$ which can be achieved if $P1$ is activated. Else if $C$ wins the race, then its utility is at most $\text{SAV}(E x)$ which can be achieved by activating $P2'$, $C1'$ and $P1$. By the same calculation as the previous case, since $p \leq \gamma r' \leq \frac{\text{SAV}(B c_a')}{\text{SAV}(B c_a' + B x')}$, we have $\text{SAV}((B x' - B c_a' + E x) \cdot (1 - p) + E x \cdot p) < \text{SAV}(E x - B x').$
Lemma 4.3 (Bob-miner coalition). Suppose that the hash function $H(\cdot)$ is a one-way function. Let $C$ be any coalition that consists of Bob and a subset of miners controlling at most $\gamma$ fraction of mining power. Then, as long as $\gamma \leq \frac{E_S}{E_{c_S} + E_x}$, for any PPT coalition strategy $S_C$, it must be that there is a negligible function $\text{negl}(\cdot)$ such that

$$\text{util}^C(S_C, HS_{-C}) \leq \text{util}^C(HS_C, HS_{-C}) + \text{negl}(\lambda).$$

Proof. When the coalition $C$ follows the protocol, $P1$ and $P1'$ will be activated, and the utility of $C$ is $\$BV(\beta x' - \xi x) > 0$. Now, suppose $C$ may deviate from the protocol. 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 $C$ can find $pre_a$ by itself. $C$ can earn coins from Rapidash’ only through $P1'$ and $C2'$. However, $C$ has to send $pre_a$ to $P1'$ to make it activated, while $C$ does not know $pre_a$. Thus, to make $C$’s utility positive, $C2'$ must be activated by $C$. $C2'$ can be activated by sending $(pre_b',pre_c')$ or $(pre_a',pre_b)$. Because $C$ 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 $C$. To activate $C2'$ by $(pre_a',pre_b)$, $C$ must know $pre_a'$. Ignoring the negligible probability that $C$ 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, $C1$ can be activated only if $P2$ has been activated. Thus, if $P2$ is not activated, $C2$ is the only activation point that can be activated in Rapidash, and $C$’s utility is at most $\$BV(\beta x' - \xi x - \xi c_b + \xi \epsilon)$ which is no more than the honest case $\$BV(\beta x' - \xi x)$ since $\xi c_b > \xi \epsilon$. 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 $-C$ will activate $C2$ if it wins a block. We say $C$ loses the race if a non-colluding miner mines a new block during Ethereum time $(t^*, t^* + \tau]$. Otherwise, we say $C$ wins the race. If $C$ loses the race, it gets nothing from $C1$ or $C2$, and its utility is at most $\$BV(\beta x' - \xi x - \xi c_b) + \xi \epsilon$ which can be attained if $C2'$ is activated. Else if $C$ wins the race, then its utility is at most $\$BV(\beta x')$ which can be achieved by activating $P2$, $C1$ and $C2'$. Since $p \leq \gamma \leq \frac{\$BV(\xi c_b)}{\$BV(\xi c_b + \xi \epsilon)}$ and $\beta e' < \beta x'$, we have

$$\$BV((\beta e' - \xi x - \xi c_b) \cdot (1 - p) + \beta x' \cdot p) < \$BV(\beta x' - \xi x).$$

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 $T_0$. $C$ can gain coins from Rapidash’ either from $P1'$ or from $C2'$, but not both, and in either case, $C$ gains at most $\$BV(\beta e' - \xi x - \xi c_b)$ which can be attained. Henceforth, we assume $P1$ is not activated. If $P2$ is also not activated, then $C1$ cannot be activated. Therefore, the coalition can cash out at most $\xi \epsilon < \xi c_b$ from Rapidash by invoking $C2$, and the coalition’s utility is less than the honest case.

Therefore, we may assume that $P2$ is activated at Ethereum time $t^* \geq T_1$, and $pre_b$ 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 $-C$ will activate $C2$ if it wins a block. We say $C$ loses the race if a non-colluding miner mines a new block during Ethereum time $(t^*, t^* + \tau]$. Otherwise, we say $C$ wins the race. If $C$ loses the race, it gets nothing from $C1$ or $C2$, and its utility is at most $\$BV(\beta x' - \xi x - \xi c_b)$ which can
be achieved if \( P1' \) is activated. Else if \( C \) wins the race, then its utility is at most 
\[
BV(Bx' - E_{Cb}) \cdot (1 - p) + BV(Bx' \cdot p) < BV(Bx' - E_{Cx}).
\]

**Theorem 4.4** (CSP fairness). Suppose that the hash function \( H(\cdot) \) is a one-way function. Moreover, suppose 
\[
\gamma \leq \frac{Bc}{B_c + Bx'} \quad \text{and} \quad \gamma \leq \frac{Ec_b}{Ec_b + E_{Cx}}.
\]
Then, the atomic swap protocol satisfies \( \gamma \)-CSP-fairness.

*Proof.* In Lemma 4.2 and Lemma 4.3, we show that the atomic swap protocol satisfies \( \gamma \)-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 \( \gamma \)-CSP-fairness when the coalition \( C \) only consists of miners controlling at most \( \gamma \) 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 \( C \) follows the protocol, its utility is always zero. Suppose \( C \) may deviate from the protocol. Notice that the utility of \( C \) 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 \( C \) 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 \( C \) finds \( pre_b \) by itself, \( C2 \) can never be activated. To sum up, except the negligible probability, the utility of \( C \) 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 \( C \) finds \( pre_a \) by itself, \( C2 \) and \( C2' \) can be activated only by \( (pre'_a, pre_b) \). However, Bob always sends \( _{\text{pre}} \) 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 \( C \) 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 \( C \) 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 \( 1/\text{poly}(\lambda) \) 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 \( 1 - \text{negl}(\lambda) \) 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 \( \tau \) 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 \( 1/\text{poly}(\lambda) \) 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 \( \tau \) 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 \( 1/\text{poly}(\lambda) \) 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.

• **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 \( 1/\text{poly}(\lambda) \) 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 \( \$BV(Bx’ - Ex) > 0 \).

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 \( \tau \) Ethereum time has passed since P2 is activated, Bob sends \( \_ \) to C1. Alice and Bob will not send any transaction afterward, 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 \( 1/\text{poly}(\lambda) \) 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 \( \tau’ \) 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 \( 1/\text{poly}(\lambda) \) 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 \( 1/\text{poly}(\lambda) \) fraction of the mining power is honest, P1 and
P1’ will be activated in polynomial time except with negligible probability, and her utility is \$AV(\xi - Bx') > 0.

On the other hand, if Bob does not send \textit{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 \textit{pre}_c to P1, by the same argument as the previous case, Alice’s utility is \$AV(\xi - Bx') > 0. If Bob does not send \textit{pre}_c to P1, Alice and Bob must go to the abort phase. In the abort phase, Bob will send \(t\) to P2’ at Ethereum time \(T_0\). When \(\tau\)' Bitcoin time has passed since P2’ is activated, Alice sends \(t\) to C1’. Because Alice never sends any transaction containing \textit{pre}_a, P1’ and C2’ can never be activated except the negligible probability. As long as \(\frac{1}{\text{poly}(\lambda)}\) 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.

\[ \square \]

5 Bitcoin Instantiation

In this section we describe how \textsc{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 \(Adr \in \{0,1\}^\lambda\) 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

\[ tx_A := \left( \left( (Adr_1, \Phi_1, v_1), \ldots, (Adr_n, \Phi_n, v_n) \right), \left( (Adr'_1, \Phi'_1, v'_1), \ldots, (Adr'_m, \Phi'_m, v'_m) \right) \right), \]

charges \(v_i\) coins from each input address \(Adr_i\) for \(i \in [n]\), and pays \(v'_j\) 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 \(\Phi : \{0,1\}^\lambda \rightarrow \{0,1\}\) 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 \textsc{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 \textsc{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: \( H : \{0, 1\}^* \rightarrow \{0, 1\}^\kappa \), (2) transaction timestamp verification wrt. current height of the blockchain denoted by \( \text{NOW}^5 \) and (3) digital signature verification. The signature scheme consists of the key generation algorithm \( \text{KGen}(\kappa) \) generating the signing key \( sk \) and the verification key \( pk \), the signing algorithm \( \text{Sign}(sk, m) \) generating a signature \( \sigma \) on the message \( m \) using \( sk \), and the verification algorithm \( \text{Vf}(pk, m, \sigma) \) validating the signature wrt. \( pk \). We say an address \( \text{Adr} \) (associated script \( \Phi \)) is controlled by a user if the user knows a witness \( x \) s.t. \( \Phi(x) = 1 \).

5.2 Instantiating Rapidash Single Instance

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. Basically, Bob uses the transaction \( tx_{\text{stp}} \) to put his deposit \( v + c_b \) into the address \( \text{Adr}_{\text{stp}} \). The script on the address \( \text{Adr}_{\text{stp}} \) allows three ways to spend the deposit:

1. Use \( pre_a \) to pay \( v \) amount to an address \( \text{Adr}_1^A \) owned by Alice, and \( c_b \) to an address \( \text{Adr}_1^B \) owned by Bob.

2. After a timeout \( T_1 \) since the address \( \text{Adr}_{\text{stp}} \) comes into existence, use \( pre_b \) to pay the entire deposit amount \( v + c_b \) to the address \( \text{Adr}_{\text{P2}} \), which is associated with the script \( \Phi_{\text{P2}} \). \( \Phi_{\text{P2}} \) says that when \( T_2 \) time has passed after the address comes into existence, the \( v + c_b \) coins in \( \text{Adr}_{\text{P2}} \) can be paid to Bob’s address \( \text{Adr}_{\text{P2}}^B \).

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

To make sure that Alice and Bob cannot unilaterally spend from the address \( \text{Adr}_{\text{stp}} \) and \( \text{Adr}_{\text{P2}} \), their associated scripts require signatures from both Alice and Bob to spend from these addresses. Note also that the transactions \( tx_{\text{P1}} \), \( tx_{\text{P2}} \), and \( tx_{\text{C2}} \) needed to spend from \( \text{P1} \), \( \text{P2} \), and \( \text{C2} \) are signed with different public keys of Alice and Bob, i.e., \( (pk_a, pk_b), (pk_a', pk_b'), \) and \( (pk_a'', pk_b'') \) respectively. This makes sure that each transaction can invoke only the intended activation point.

Protocol flow. Before setting up Rapidash on the blockchain, Alice and Bob agree on the setup transaction \( tx_{\text{stp}} \). The transaction must be signed by Bob to take effect. However, before signing \( tx_{\text{stp}} \), Alice and Bob agree on and sign all redeeming transactions, including \( tx_{\text{P1}}, tx_{\text{P2}}, tx_{\text{C1}}, \) and \( tx_{\text{C2}} \). Alice and Bob now broadcast all these transactions (not including \( tx_{\text{stp}} \)) and both of their signatures — notice that they cannot be published on the Bitcoin blockchain yet because the addresses they depend on, \( \text{Adr}_{\text{stp}} \) or \( \text{Adr}_{\text{P2}} \), are not ready yet.

At this moment, Bob reveals his signatures on \( tx_{\text{stp}} \). Once \( tx_{\text{stp}} \) is published on the Bitcoin blockchain, the execution phase starts. During the execution phase, either Alice reveals \( pre_a \) and publishes transaction \( tx_{\text{P1}} \) (along with signatures on the transaction), or Bob reveals \( pre_b \) and publishes transaction \( tx_{\text{P2}} \) (along with signatures on the transaction) after \( T_1 \) time has passed since the confirmation of \( tx_{\text{stp}} \). In the honest run of the protocol, if \( tx_{\text{P1}} \) is confirmed, Bob gets back his

---

4\( \kappa = 160 \) in Bitcoin when using the opcode \( \text{OP\_HASH160} \).

5Instantiated using the opcode \( \text{OP\_CHECKSEQUENCEVERIFY} \) 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 \( \text{OP\_CHECKLOCKTIMEVERIFY} \) in Bitcoin that checks if the current height of the blockchain is beyond a threshold.

6The signature scheme can be instantiated with either Schnorr or ECDSA in Bitcoin. ECDSA signatures are verified using the opcode \( \text{OP\_CHECKSIG} \) and Schnorr signatures via the taproot fork.
Table 1: Rapidash’s transactions in Bitcoin. Here $\Phi_0^B, \Phi_1^B, \Phi_2^B$ are the script that requires the signature under Bob’s public key while $\Phi_1^A$ is the script that requires the signature under Alice’s public key.

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$tx_{stp}$</td>
<td>$tx\left(\left[(Adr_0^B, \Phi_0^B, S_v + S_c_b)\right]\right)$ [\left[(Adr_{stp}, \Phi_{stp}, S_v + S_c_b)\right]]</td>
</tr>
<tr>
<td>$tx_{P1}$</td>
<td>$tx\left(\left[(Adr_{stp}, \Phi_{stp}, S_v + S_c_b)\right]\right)$ [\left[(Adr_1^A, \Phi_1^A, S_v), (Adr_1^B, \Phi_1^B, S_c_b)\right]]</td>
</tr>
<tr>
<td>$tx_{P2}$</td>
<td>$tx\left(\left[(Adr_{stp}, \Phi_{stp}, S_v + S_c_b)\right]\right)$ [\left[(Adr_{P2}, \Phi_{P2}, S_v + S_c_b)\right]]</td>
</tr>
<tr>
<td>$tx_{C1}$</td>
<td>$tx\left(\left[(Adr_{P2}, \Phi_{P2}, S_v + S_c_b)\right]\right)$ [\left[(Adr_2^B, \Phi_2^B, S_v + S_c_b)\right]]</td>
</tr>
<tr>
<td>$tx_{C2}$</td>
<td>$tx\left(\left[(Adr_{stp}, \Phi_{stp}, S_v + S_c_b)\right]\right)$ [\left[(Adr_{burn}, \Phi_{burn}, S_v + S_c_b - S_\epsilon)\right]]</td>
</tr>
</tbody>
</table>

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

P1: if $(H(pre_a) = h_a) \land (Vf(pk_a, tx, \sigma_a) = 1) \land (Vf(pk_b, tx, \sigma_b) = 1)$

then return 1

P2: if $(\text{NOW} > T_1) \land (H(pre_b) = h_b) \land (Vf(pk_a', tx, \sigma_a) = 1) \land (Vf(pk_b', tx, \sigma_b) = 1)$

then return 1

C2: if $(Vf(pk_{a}'), tx, \sigma_a) = 1) \land (Vf(pk_{b}', tx, \sigma_b) = 1) \land (H(pre_a) = h_a) \land (H(pre_b) = h_b)$

then return 1

// Values $h_a, h_b, pk_a, pk_b, T_1, pk_{a}', pk_{b}', pk_{a}'', pk_{b}''$ are hardwired

$\Phi_{P2}(tx, \sigma_a, \sigma_b)$

C1: if $(\text{NOW} > T_2) \land (Vf(pk_{a}, tx, \sigma_a) = 1) \land (Vf(pk_{b}, tx, \sigma_b) = 1)$ then return 1

// Values $T_2, pk_a, pk_b$ are hardwired

Figure 1: The description of scripts $\Phi_{stp}$ and $\Phi_{P2}$. Here $tx$ is the transaction spending from the script. Keys $pk_a, pk_a'$ and $pk_b''$ belong to Alice, $pk_b, pk_b'$ and $pk_b''$ belong to Bob.

collateral immediately. If not, Bob can redeem the collateral after waiting for time $T_1 + T_2$ using $tx_{P2}$ and $tx_{C1}$. In the event of misbehavior leading to both $pre_a$ and $pre_b$ being revealed, any miner in the system can immediately spend from the C2 branch of $\phi$, and burn all coins except $S_\epsilon$ coins as transaction fee for itself.

### 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.
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.

5.3.1 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 $\Phi_{stp}$ (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 $\Phi_{stp}$ script (described in Figure 3). The change in $\Phi_{P2}$ is that we replace $T_2$ with $\tau$.

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 $tx_{P1}, tx_{P2}, tx_{C1},$ 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 3: The description of script $\Phi_{\text{stp}}$. Here $tx$ is the transaction spending from the script. Keys ($pk_a$, $pk'_a$, $pk''_a$) and ($pk_b$, $pk'_b$, $pk''_b$) belong to Alice and Bob, respectively.

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 Φstp similar in manner to the RAPIDASH instantiation above.

Table 2: Description of additional transactions in Bitcoin for RAPIDASH’ atomic swap. Here (Φ1A, Φ2A) and Φ1B are scripts that require a signature from Alice’s and Bob’s public key, respectively.

<table>
<thead>
<tr>
<th>Description</th>
<th>$tx_{empty}^{\Phi _{P1}^A}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>x { (AdrA, ΦA, $x + $c_a) }, x { (AdrB, ΦB, $x + $c_b) }</td>
<td>$tx_{empty}^{\Phi _{P2}A}$</td>
</tr>
<tr>
<td>x { (AdrA, ΦA, $x + $c_a) }, x { (AdrB, ΦB, $x + $c_b) }</td>
<td>$tx_{empty}^{\Phi _{P1}^B}$</td>
</tr>
<tr>
<td>x { (AdrB, ΦB, $x + $c_b) }, x { (AdrA, ΦA, $x + $c_a) }</td>
<td>$tx_{empty}^{\Phi _{P2}^B}$</td>
</tr>
<tr>
<td>x { (AdrA, ΦA, $x + $c_a) }, x { (AdrB, ΦB, $x + $c_b) }</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5: The description of script Φstp.

Transactions. We describe below the different transactions needed for our RAPIDASH’ instantiation.

- We now have two additional payment redeem transactions, $tx_{P1}^{empty}$ and $tx_{P2}^{empty}$ (see Table 2) apart from $tx_{P1}$ and $tx_{P2}$ that redeem from the payment address Adrstp. We have the transaction $tx_{P1}^{empty}$ that redeems $x$ coins to Bob’s address and $c_a$ coins are paid to Alice’s address. The transaction $tx_{P2}^{empty}$ redeems $x + c_a$ coins to an auxiliary address AdrP2. The description of Φstp is given below in Figure 5 with Alice and Bob’s roles being reversed in RAPIDASH’. We set the two transactions $tx_{empty}^{P1}$ and $tx_{empty}^{P2}$ to redeem the coins from the (E1) 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 Φstp has a modification in the C2′ branch, where we require either (preA, preB) or (preA, preB) along
with the signatures of Alice and Bob. Similarly the script $\Phi_{P2'}$ of the auxiliary addresses are the same as $\Phi_{P2}$ from Rapidash, except we replace the timeout value $\tau$ with $\tau'$. 

- In addition to the collateral redeem transaction $tx_{C1'}$ we have the transaction $tx_{C1'}^\text{empty}$ and a modified $tx_{C2'}$ (see Table 2). The new transaction $tx_{C1'}^\text{empty}$ redeem coins from the C1$'$ activation point, if transaction $tx_{P2}^\text{empty}$ was activated earlier. Similar to the Rapidash instantiation, C1$'$ branch timeout of $\tau'$ for $tx_{C1'}^\text{empty}$ is implemented via the auxiliary addresses created by $tx_{P2}^\text{empty}$. Finally the transaction $tx_{C2'}$ 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 $\Phi'_{stp}$. The change in $\Phi_{P2}'$ is that we replace $T_2$ with $\tau'$.

A pictorial description of the transaction flow for payment and 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 $tx_{P1}'$, $tx_{P2}'$, $tx_{C1}'$, $tx_{C2}'$, and $tx_{C1}'_{empty}$. They broadcast all these transactions and the respective signatures, like before. However, this time Alice and Bob sign the transaction $tx_{P1}'_{empty}$ such that only Alice has both signatures. She does not broadcast the signatures and keeps it privately. Similarly, Alice and Bob sign the transaction $tx_{P2}'_{empty}$ 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 $tx_{P1}'_{empty}$ 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 $tx_{P2}'_{empty}$ along with the valid signatures he has in his possession. If $tx_{P2}'_{empty}$ is published on the blockchain, activation point C1' can be activated by $tx_{C1}'_{empty}$ after a timeout of $\tau'$ 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**


