Ponyta: Foundations of Side-Contract-Resilient Fair Exchange

Hao Chung Elisaweta Masserova Elaine Shi
Sri AravindaKrishnan Thyagarajan

Carnegie Mellon University

Abstract

Fair exchange is a fundamental primitive for blockchains, and is widely adopted in applications such as atomic swaps, payment channels, and DeFi. Most existing designs of blockchain-based fair exchange protocols consider only the users as strategic players, and assume honest miners. However, recent works revealed that the fairness of commonly deployed fair exchange protocols can be completely broken in the presence of user-miner collusion. In particular, a user can bribe the miners to help it cheat — a phenomenon also referred to as Miner Extractable Value (MEV).

We provide the first formal treatment of side-contract-resilient fair exchange. We propose a new fair exchange protocol called Ponyta, and we prove that the protocol is incentive compatible in the presence of user-miner collusion. In particular, we show that Ponyta satisfies a coalition-resistant Nash equilibrium. Further, we show how to use Ponyta to realize a cross-chain coin swap application, and prove that our coin swap protocol also satisfies coalition-resistant Nash equilibrium. Our work helps to lay the theoretical groundwork for studying side-contract-resilient fair exchange. Finally, we present practical instantiations of Ponyta in Bitcoin and Ethereum with minimal overhead in terms of costs for the users involved in the fair exchange, thus showcasing instantiability of Ponyta with a wide range of cryptocurrencies.

Contents

1 Introduction

Consider the following scenario between mutually distrusting Alice and Bob: Alice possesses something that Bob wants, and Bob possesses something that Alice wants. A fair exchange protocol enables an exchange between Alice and Bob such that either both of them get the desired item, or neither of them does. Fair exchange is a problem that has been studied for a long time [Mic03, Aso98, ASW97]. In particular, it has been shown that fair exchange is impossible to achieve without further assumptions [PG99, Mic03]. One way to circumvent this limitation is to rely on a trusted third party such as a blockchain [BBSU12, Her18, MMS+, vdM19, MD19, Max, CGGN, BDM, Fuc, BKb, MES16, MMA, Bis, ZHL+19, JMM14, TYME21, PD]. Indeed, fair exchange is a fundamental primitive in blockchain applications [BBSU12, Her18, MMS+, vdM19, MD19, Max, CGGN, BDM, Fuc, BKb, MES16, MMA, Bis, ZHL+19, JMM14, TYME21, PD], and has been widely adopted in the form of atomic swaps [Her18, MMS+, vdM19, MD19], contingent payment [Max, CGGN, BDM, Fuc, BKb], payment channels [PD, DW15, GM, MMSH, MBB+, DFH18, DEFM19], or vaults [MES16, MMA, Bis, ZHL+19].

Most existing blockchain-based fair exchange protocols consider only Alice and Bob as potentially strategic players, and the miners are assumed to be honest [EFS20, DEF18, AHS22, CGJ+17a, GKM+22, BKa]. Recently, however, the community has become increasingly concerned that potential user-miner collusion can completely break the fairness guarantees promised by fair exchange protocols [TYME21, WHF19, Bon, MMS+, MHM18a, JSZ+21, Ham]. As a concrete example, a Hash Timelock Contract (HTLC) is one of a commonly employed mechanism for realizing fair exchange in blockchain environments. Imagine that Alice has a secret s and she wants to sell it to Bob at a price of $\$v$ coins. A standard HTLC contract is parametrized with the hash of the secret $h = H(s)$, a timeout value T, and the price $\$v$. In a preparation phase, Bob deposits $\$v$ coins into the contract. The contract now allows Alice to redeem the $\$v$ coins by posting the secret s whose hash should be equal to h. However, if Alice fails to redeem $\$v$ by time T, Bob can get his deposit $\$v$ back. Since the HTLC contract can protect Bob from an offline Alice, we also say that it is dropout resilient.

Unfortunately, a number of recent works have pointed out that the standard HTLC is vulnerable to user-miner collusion. In particular, Bob may collude with some miners in an attempt to starve Alice's redeeming transaction. If Bob's coalition can suppress the transaction till the timeout T, then they can get the $\$v$ deposit back after learning the secret s! Various works have shown that such user-miner collusion is indeed possible in practice through bribery mechanisms [TYME21, WHF19, HZ20, MHM18a, JSZ+21, Ham]. Such attacks can be instantiated in various ways [TYME21, WHF19, HZ20, MHM18a, JSZ+21, Ham], e.g., by exploiting the decentralized smart contracts available in blockchain environments. Moreover, with some clever tricks, they can be instantiated in a fairly inexpensive manner [TYME21].

Tsabary et al. [TYME21] also made a pioneering attempt to try to overcome such bribery attacks. They proposed a new fair exchange mechanism called a *Mutual-Assured Destruction Hash Timelock Contract (MAD-HTLC)*. Just like the HTLC contract, in MAD-HTLC, Bob deposits $\$v$ into the contract upfront. The contract allows Alice to redeem the $\$v$ coins by revealing the secret pre_a that Bob wants to learn, and whose hash is hard-wired in the contract. If Alice does not redeem the coins by time T, Bob can claim back his deposit by revealing another secret pre_b whose hash is also hard-wired in the contract. Importantly, MAD-HTLC adds the following clever rule (henceforth called the *bomb*): anyone (including the miner of the block) who present both pre_a and pre_b can claim the $\$v$ coins for themselves. MAD-HTLC deters Bob from cheating in the following way: if a cheating Bob posts pre_b and attempts to claim his $\$v$ even after Alice has revealed pre_a, then any miner, who now has knowledge of both pre_a and pre_b, can preempt

Bob's transaction by triggering the bomb and claiming the $\$v$ coins itself. This effectively thwarts Bob's attempt. Although MAD-HTLC does seem to resist the known bribery mechanisms, it opens up some possible new attacks. For example, as soon as Alice publishes pre_a, if a Bob-miner coalition happens to mine the next block, they should claim the $\$v$ back by posting (pre_a, pre_b), and split off the gain among themselves. Indeed, the authors of the MAD-HTLC paper acknowledge themselves that MAD-HTLC does not provide any provable guarantee in the presence of user-miner collusion [TYME21].

Therefore, the following natural and fundamental question remains open:

Can we have a blockchain-based fair exchange protocol that resists user-miner collusion?

If a fair exchange protocol is incentive compatible even in the presence of user-miner coalitions, we also say that it is side-contract-resilient.

1.1 Our Results and Contributions

To the best of our knowledge, we are the *first to give a formal treatment of side-contract-resilient fair exchange*. Specifically, we make the following contributions:

Ponyta: a side-contract resilient fair exchange protocol. We propose a new mechanism called Ponyta[1], which works atop any standard Proof-of-Work blockchain or a Proof-of-Stake blockchain where the next block proposer is selected on the fly with probability proportional to the miner's stake. We prove that Ponyta achieves a game-theoretic notion called *cooperative strategy proofness* (or CSP-fairness for short), i.e., any coalition of players (that does not simultaneously contain Alice and Bob[2]) is incentivized to play honestly, as long as the rest of the world is playing honestly and the coalition does not control 100% of the mining power. In other words, the honest behavior is a coalition-resistant Nash equilibrium.

We argue why Ponyta disincentivizes coalitions of 100% mining power by analyzing the coalition formation process as a *meta-game*. We argue that a 100% coalition is not a equilibrium in this meta-game. This also justifies our modeling approach where we consider coalitions that do not wield 100% of the mining power.

Last but not the least, we use Ponyta to realize a cross-chain coin swap application, and prove that the resulting protocol also satisfies CSP fairness.

Our ideas in a nutshell. We now explain our idea in a nutshell. We are inspired by MAD-HTLC's elegant idea of using a bomb [TYME21]. The problem with MAD-HTLC is that triggering the bomb actually benefits a Bob-miner coalition. In our work, we make Alice and Bob put in extra collateral into the smart contract. If no player cheats, both players can get their collateral back at the end of the protocol. However, should Bob cheat causing the bomb to be triggered, not only can the miner triggering the bomb obtain some payment, but also part of the collateral will be *burnt* and made unrecoverable. In this way, we make sure that triggering the bomb actually hurts an Alice-miner or a Bob-miner coalition; and as a result, they would not want to behave in a way that risks triggering the bomb.

Our definitional approach. Our work helps to lay the formal groundwork for studying side-contract-resilient fair exchange protocols. In general, mechanism design in the blockchain world is

[1] Ponyta is a fire-type Pokémon who can control its flames such that its rider is not burnt. Our Ponyta contract incentivizes honest behavior and protects the players' collateral from being burnt.

[2] If Alice and Bob were in the same coalition, then they would not need to do the fair exchange.

complicated by the existence of decentralized smart contracts which can be used to openly solicit coalitions, as well as implement potentially *arbitrary* side contracts among players.

Our modeling approach follows a line of recent works [PS17a, CGL+18, WAS22, CS21] and can capture arbitrary side contracts among coalitions of players. We assume that the goal of a rational coalition is to maximize its joint utility, i.e., the sum of the utilities of all coalition members. Equivalently, we are assuming that there is some *binding* side contract that allows the coalition to split their joint gains among themselves, and moreover the enforcement of this side contract is ensured. Besides capturing arbitrary binding side contracts between different players, our modeling approach also captures the coalitions that are naturally formed when the same player controls multiple pseudonyms such as public keys. For example, Alice or Bob may well be the pseudonym of some miner (*c.f.* the assumption made in earlier works [TYME21], that Bob must not be a miner, is unenforceable in the real world).

Finally, our definitional approach models players and coalitions as interactive Turing Machines who can send and receive a special type of variables called money. This allows us to capture a most general strategy space, i.e., deviating players can not only send arbitrary messages, but also post new smart contracts on the fly during protocol execution.

Instantiation atop Bitcoin and Ethereum. We first describe Ponyta assuming an existence of generic smart contracts. We then give an instantiation of Ponyta that is compatible with the scripting language of Bitcoin. To do so, we introduce two novel transaction-level tricks which we call *conditional timelock redeem* and *conditional burn*. The former ensures that coins from an address are redeemable only if T time has passed since the redeeming of coins from another address. The latter allows miners to redeem a portion of the coins from a target address while burning the rest under some conditions. These techniques may be of independent interest and lend to other applications. We also instantiate Ponyta in Solidity [Eth22], Ethereum's smart contract language, and deploy it on the Rinkeby testnet [rin22]. We show that the cost overhead for either user in terms of *gas cost* is insignificant compared to MAD-HTLC.

1.2 Additional Related Work

We now review additional related work besides those already mentioned. Our work is also related to financially fair protocols [BK14, MB17, BZ17, CGJ+17b, KB16, KMS+16], where a common theme is using collateral and penalty mechanisms to incentivize honest behavior. For some specific applications such as lottery and leader election, a few works showed that collateral is in fact not necessary to achieve game-theoretic fairness [MB17, BZ17, CCWS21]. To the best of our knowledge, almost all prior works consider only the users as potentially strategic players, and the miners are assumed to be honest.

Miner Extractable Value (MEV) is among the often debated problems in the cryptocurrency community today. In MEV, the miner leverages its unique position, i.e., its ability to decide what to include in the block, to profit from blockchain applications. The original motivation of our work is possible user-miner collusion which is a form of MEV. Interestingly, inspired by MAD-HTLC [TYME21], we leverage MEV itself to defend against MEV — specifically, to thwart such user-miner collusion, Ponyta allows honest miners to extract value should any cheating behavior take place.

2 Technical Roadmap

2.1 Problem Statement and Assumptions

Suppose that Alice has a secret s and Bob wants to buy the secret from Alice at a price of $\$v$. We would like to design a fair exchange protocol to accomplish this. Henceforth we assume that Alice loses value $\$v_a$ for revealing s, and Bob gains value $\$v_b$ if he learns s. If $\$v_a < \$v < \$v_b$, both parties benefit from Alice selling s to Bob at price $\$v$.

Throughout this paper, we will assume that Alice or Bob may collude with a subset of the miners, and the coalition may adopt arbitrary probabilistic polynomial-time strategies to maximize its joint utility. The only restriction we impose on the strategy space is that the coalition does not perform a consensus-level or network-level attack. For example, we do not consider 51% attacks that aim to make profit through double-spending. There is an orthogonal line of work that focuses on consensus security [GKL15, PSS17, PS17b].

Therefore, we assume an idealized mining process. In every time step, an ideal functionality picks the next winning miner at random with probability proportional to its mining power. The winning miner may choose a set of transactions to include in the next block. We assume that the network delay is 0, i.e., any message posted by Alice, Bob or any new block mined will be immediately seen by other players.

In Section 3, we will formalize the notion of transactions and a smart contract execution model.

2.2 Strawman and Prior Approaches

For simplicity, in the technical roadmap, we will assume the existence of general smart contracts, although later, we will indeed show how to instantiate our protocol atop Bitcoin which supports only a restricted language for contracts.

Naïve protocol. The most straightforward idea is for Alice and Bob to agree on a smart contract which knows $h_s = H(s)$ where $H(\cdot)$ denotes a cryptographic hash function. Moreover, Bob deposits $\$v$ into the contract upfront. The smart contract's logic is very simple:

> On receiving s from Alice such that $H(s) = h_s$, send $\$v$ to Alice.

Let $T = 0$ be the moment when Bob deposited the initial $\$v$ into the contract. The honest protocol is simple: Alice sends s to the smart contract at $T = 0$, and Bob does nothing. We always want the miner's honest strategy to be consistent with the underlying consensus protocol, that is, the miner should include all outstanding transactions in the block.

Somewhat surprisingly, even this very simple protocol satisfies a coalition-resilient game theoretic notion. Consider any coalition consisting of either Alice or Bob as well as an arbitrary subset of the miners: the coalition should have no incentive to deviate from the honest protocol, if its goal is to maximize its own utility. Such a notion is often referred to as *cooperative-strategy-proofness* (or *CSP fairness*). In particular, it implies that the *honest strategy is a coalition-resistant Nash equilibrium*.

This approach, however, has two drawbacks: 1) Alice can harm Bob without incurring any cost to herself. Simply by withholding s, Bob will lose its deposit $\$v$; and 2) even if Alice is well-meaning, she may accidentally drop offline (e.g., due to an unforeseen circumstance such as losing her password) in which case Bob also loses $\$v$. Note that the first drawback can be overcome by requiring Alice to make a deposit of $\$v'$ too besides Bob's deposit of $\$v$, such that Alice can claim $\$v + \$v'$ if she sends s to the smart contract. However, this does not fix the second problem, that is, the protocol is *not dropout resilient*.

Hash timelock Contract. A line of work used hash timelock contracts (HTLCs) [atob, atoa, Her18, CGGN, Max] aimed at achieving dropout resilience, but at the price of losing CSP fairness. A standard HTLC contract works as follows, where Bob is still required to deposit $\$v$ into the contract upfront, and the contract is parametrized with a timeout T_1:

HTLC

- On receiving s from Alice such that $H(s) = h_s$, send $\$v$ to Alice.
- After T_1, on receiving `ok` from Bob: send $\$v$ to Bob.

In the above, the two activation points are mutually exclusive, i.e., only one of them can be activated and only once.

The above contract allows Bob to recover its deposit should Alice drop offline. However, the HTLC-based protocol is *not CSP fair* in the presence of user-miner coalitions. Suppose there is no transaction fee, then a coalition of Bob and some miners should never include Alice's transaction: if Alice's transaction is starved till after T_1, then Bob's coalition can get both the secret s and the $\$v$ deposit back!

If Alice offers a transaction fee of $\$f$, then as long as Bob bribes each miner $\$f + \$\epsilon$ (for some small $\$\epsilon > 0$) for excluding Alice's transaction, rational miners will take the bribe [TYME21]. This bribery attack makes sense for Bob as long as $\$v > \$f \cdot T_1$. It may seem like HTLC is secure as long as we make $T_1 \cdot \$f$ sufficiently large. However, Tsabary et al. [TYME21] showed new attacks where the cost to Bob is not dependent on T_1. We provide more details on Tsabary et al.'s attack [TYME21] in Section 4.3. In particular, using our terminology, such attacks are viewed as strategies in the coalition forming meta-game. The HTLC contract is undesirable because there exist meta-games where 100% of the miners taking the "bribe" is an equilibrium, thus encouraging 100% coalitions — see Section 4.3 for details.

MAD-HTLC. Tsabary et al. [TYME21] suggest a new contract called MAD-HTLC in an attempt to remove the undesirable 100%-miner-colluding equilibrium in the coalition-forming meta-game. As before, Bob deposits $\$v$ upfront, and the contract works as follows[3]:

MAD-HTLC

- **On receive** pre_a from Alice such that $H(pre_a) = h_a$: send $\$v$ to Alice.
- **After** T, **on receive** pre_b from Bob such that $H(pre_b) = h_b$: send $\$v$ to Bob.
- **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 P.

All activation points in the above contract are again mutually exclusive, and here we use the notation pre_a to denote the secret Alice wants to sell.

In MAD-HTLC, if Alice has disclosed pre_a and yet Bob still attempts to get his deposit back by posting pre_b, then the miner easily preempts Bob's transaction and claim $\$v$ itself by posting the pair (pre_a, pre_b). MAD-HTLC indeed defends against the simple bribery attack mentioned above as well as the attack of Tsabary et al. [TYME21] — or in our language, MAD-HTLC removes the undesirable 100%-colluding-equilibrium in the coalition forming meta-game (see Section 4.3).

[3] MAD-HTLC has some additional logic to defend against a spiteful Bob which we omit for simplicity, since the additional logic does not help mitigate the coalition attacks we point out.

Unfortunately, MAD-HTLC does not have provable security in the presence of miner-user coalitions, as the authors acknowledge themselves [TYME21]. Their game-theoretical formulation considers only individual deviations. In fact, the following is straightforward to observe:

Observation: MAD-HTLC is NOT incentive compatible in the presence of *binding side contracts*.

A binding side contract allows the coalition to split off their joint utility in a binding manner. For example, in MAD-HTLC, Bob can collude with some miners, and as soon as Alice posts pre_a, if the colluding miners happen to mine the next block, they can exclude Alice's transaction and redeem the $\$v$ coins for themselves by posting both pre_a and pre_b. Then, using the binding side contract, the coalition can split off the $\$v$ coins among its members. It could also be that Bob is a miner himself. In this case, if Bob happens to mine the next block after Alice posts pre_a, Bob can get the secret for free. As the authors of MAD-HTLC [TYME21] acknowledge themselves, MAD-HTLC provides no guarantee when Bob is a miner himself — however, in reality, since creating pseudonyms or public keys is cheap, it is difficult for Alice to check whether Bob is a miner.

2.3 Our Approach

In our Ponyta contract, during the preparation phase, Alice and Bob each deposits $\$c_a$ and $\$c_b+\$v$ into the contract respectively, where $\$c_a > \$v$ and $\$c_b > 2 \cdot \$v$. We refer to $\$c_a$ and $\$c_b$ as Alice and Bob's collateral, respectively. Note that Bob needs to deposit his collateral $\$c_b$ on top of the promised payment amount $\$v$. Given pre_a as the secret Alice wants to sell, our Ponyta contract works as follows:

Ponyta contract

Payment:

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

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 either P1 or P2 is activated: on receiving _ from anyone, send $\$c_a$ to Alice and send $\$c_b$ to Bob.

C2: 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.

In the above, assuming that the coalition consists of any constant fraction of the miners (e.g., even 99%), we can set the timeout values T_1 and T_2 to be appropriate constants as we discuss in more detail in Section 4.1.

There are two types of activation points in the contract: P1 and P2 are used to redistribute Bob's promised payment $\$v$, and C1 and C2 are used to redistribute the collateral part, that is $\$c_a$ and $\$c_b$. The activation points P1 and P2 are mutually exclusive i.e., only one can be successfully activated, and at most once. Similarly, C1 and C2 are mutually exclusive, too.

Intuition. Informally, the Ponyta contract provides game theoretic fairness due to the following intuition. Suppose that Alice posted pre_a. In this case, a Bob-miner coalition would not want pre_b

to be revealed. If so, some other miners may be able to activate C2 before C1 is activated. In this case, the coalition would lose its collateral $\$c_b > 2 \cdot \$v$. Another strategy is for the coalition to claim both P2 and C2 and get $2 \cdot \$v$ back from the contract. Had they acted honestly, however, they could get $\$c_b > 2 \cdot \$v$ back, which is strictly better.

In Section 4, we shall prove that our PONYTA protocol satisfies CSP-fairness when players do not have external incentives. Informally, this means that an Alice-miner coalition or a Bob-miner coalition cannot increase its utility by deviating from the honest protocol, as long as the coalition does not control 100% of the mining power, and moreover, the parameters $\$c_a, \$c_b$, and T_2 are appropriately set. In other words, the honest protocol is a coalition-resistant Nash equilibrium.

Ponyta disincentivizes 100% coalitions. To prove that our protocol satisfies CSP-fairness, we used the assumption that the coalition does not wield 100% of the mining power. We justify this assumption by analyzing the coalition forming meta-game in Section 4.3. In the coalition forming meta-game, imagine that Bob posts a smart contract that promises to split off the gains with any miner who helps him get the $\$v$ back, after Alice has posted pre_a. We argue that 100% miner participation is not an equilibrium in this meta-game. A miner has the option of not joining the coalition, in which case it has some probability (proportional to its mining power) of claiming the $\$v$ itself by posting (pre_a, pre_b) — should Bob ever decide to cheat. To incentivize the miners to join, Bob needs to offer more than the expected gain of the miner had it not joined the coalition. However, to do so, the cost incurred for Bob would be greater than the price $\$v$ of the secret. We defer the detailed argument to Section 4.3.

In Section 4.3, we also show that this meta-game approach is not only a good complement to our formal foundations of side-contract-resilience, but also helpful for reasoning about the known bribery attacks [TYME21, WHF19, HZ20, MHM18a, JSZ+21] that pertain to the standard HTLC contract.

3 Model

3.1 Blockchain, Transaction, and Smart Contracts

Smart contracts and transactions. We assume that *smart contracts* are *ideal functionalities* that are 1) aware of money; and 2) whose states are publicly observable. A smart contract can have one or more *activation points*. Each *transaction* is associated with a unique identifier, and consists of the following information: 1) an 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 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.

3.2 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.

3.3 Protocol Execution

In the most general case, we view the start of protocol execution (i.e., $t = 0$), as the moment that the relevant smart contract is deployed. Sometimes, it may also be convenient to view the start of protocol execution as the moment when the parties involved have made their initial deposits into the contract. Later on when we define our protocols, we shall explicitly mention the start of protocol execution (i.e., $t = 0$). In our paper, an honest protocol is always a *simple* protocol that does not create additional smart contracts in the middle of the execution. Without loss of generality, we may assume that the honest protocol employs at most one contract — since if there are multiple, we can view the union of them as a single contract. Of course, strategic parties can 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 be to 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.

3.4 Utilities

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

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.

3.5 Convention for Writing Smart Contracts

For ease of exposition, we use a simplified notation for writing *meta-contracts*. A meta-contract is a platform-independent approach to express smart contract logic. Meta-contracts expressed in our notation can easily be instantiated using a general smart contract language such as Ethereum. By contrast, not every meta-contract expressible using our notational system can be instantiated atop Bitcoin, since Bitcoin's scripting language is not Turing-complete. However, for the contracts we propose in this paper, we will show that they can indeed be instantiated even atop Bitcoin.

A meta-contract will be expressed using the following style of notation:

A toy meta-contract

- **Parameters:** T.
- **Preparation phase:** Alice and Bob each deposits $\$d_a$ and $\$d_b + \$d'_b$, respectively.
- **Execution phase:**

A1: **On receive** (msg, $\$c$) from Alice: send $\$d < \$d_a + \$d_b$ to Bob.

A2: **After** T, **on receive** (msg, $\$c$) from Bob: send $\$d_a + \$d_b - \$d$ to Alice.

B1: **On receive** (msg, $\$c$) from Bob: send $\$d'_b$ to Bob.

In our notational system, every activation point is given a unique name that consists of a letter followed by a number. The leading letter defines the *type* of the activation point. All activation points of the same *type* are *mutually exclusive*. For example, if A1 has been invoked, then neither A1 nor A2 can be invoked any more; however, B1 can still be invoked (as long as it has not been invoked yet). If an activation point constrained some time interval (e.g., after T), then any attempted invocation that happens outside the specified time interval is considered invalid and not counted.

Our example toy meta-contract above has a standard *prepation phase* where Alice and Bob each deposits some coins into the contract. In a practical implementation, the contract should allow each player to withdraw its deposit if the other player has not made its deposit yet. However, once both players have made their deposits, the redistribution of money is only possible through the activation points of the execution phase. Later in our paper, the preparation phase may also have customized logic — in this case, we will spell out the logic of the preparation phase explicitly.

To instantiate our meta-contracts in practice, each party involved is actually identified by their public keys. The party can sign the message sent to the activation point to authenticate itself. In a practical instantiation, each party may also need to pay a typically small *transaction fee* for their transaction to be confirmed. For simplicity, we ignore the transaction fee in our theoretical model since we need not rely on transaction fees to achieve our game theoretic guarantees. Adding an ϵ-small transaction fee in a practical instantiation will only introduce $O(\epsilon)$-slack to our game theoretic guarantees.

We leave the concrete instantiation of our meta-contract to Section 6 and Section 7. In our meta-contract notation, we simply assume that there is an authenticated channel from each user to the smart contract.

3.6 Incentive Compatibility Definitions

Henceforth, we 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 $\mathsf{honest}_{\mathcal{C}}$ or $\mathsf{honest}_{-\mathcal{C}}$ 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 first 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 3.1 (CSP fairness)**.** We say that a fair exchange 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.

Dropout resilience. In practice, a dropout can happen due to mistakes, misconfiguration, or unforeseen circumstances, e.g., Alice may lose her hardware wallet. We define a notion called dropout resilience which requires the following. Suppose that at least $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest. Then, if either Alice or Bob plays honestly but drops out before the end of the protocol, then with $1 - \mathsf{negl}(\lambda)$ probability where $\mathsf{negl}(\cdot)$ is a negligible function, the other party's utility must be non-negative.

4 Ponyta: A CSP-Fair Fair Exchange Protocol

4.1 Construction

We now describe our Ponyta contract and protocol. Below, we use pre_a to denote the secret that Alice wants to sell, and we assume that pre_a is sampled by Alice uniform at random from $\{0,1\}^\lambda$.

Ponyta contract

Parameters: $h_a, h_b, T_1, T_2, \$v, \$c_a, \$c_b$ such that $\$c_a > \$v$ and $\$c_b > 2 \cdot \$v$.

Preparation phase: Alice deposits $\$c_a$, 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.

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 either P1 or P2 is activated: on receiving _ from anyone, send $\$c_a$ to Alice and send $\$c_b$ to Bob.

C2: 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.

As mentioned earlier, the activation points of the same type are mutually exclusive; that is, only one of the them can be activated, and at most once.

We now describe the honest protocol.

The Ponyta protocol. We assume that at the very beginning, Alice knows pre_a and Bob knows pre_b. During the preparation phase, Alice and Bob each makes a deposit of $\$c_a$ and $\$c_b + \$v$ into the Ponyta contract, respectively. Each party can freely withdraw its deposit as long as the other party has not deposited yet. However, once both parties have deposited, money redistribution is only possible through the activation points of the execution phase. The moment both parties have deposited into the contract, the protocol executoin starts, and we rename the current time t to be 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 $t = 0$. T_2 time after either P1 or P2 successfully completes, she posts an empty message _ to C1.
- *Bob*: If Alice failed to post a transaction containing a correct pre_a destined for P1 before time T_1, then Bob sends pre_b to P2 at time $t = T_1$. T_2 time after either P1 or P2 successfully completes, he sends an empty message _ to C1.

  If either P2 or C2 is successfully activated, Bob outputs the corresponding pre_a value included in the corresponding transaction. If C1 and P2 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).

Choice of timeouts. We now discuss the choice of the timeout values T_1 and T_2 in the Ponyta contract. For our game theoretic proofs (see Section 4.2) to hold, we can set $T_1 = 1$, and T_2 should be set such that $\gamma^{T_2} < \frac{\$c_b - \$v}{\$c_b}$ where γ is the maximum percentage of mining power that can form a coalition with Bob. 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.

In practice, the network may be unstable (for either Alice or Bob), or there may be a spike in transaction volume causing congestion. In these cases, we can still set $T_1 = 1$, but Bob can choose to wait longer before posting pre_b. Alice and Bob can also coordinate through an offline channel to inform each other that they are going to post either pre_a or pre_b, to avoid the situation that both pre_a and pre_b are posted, leading to part of their collateral being burnt.

4.2 Proofs

Lemma 4.1. *Let $\mathcal{C}$ be any coalition that consists of Alice and an arbitrary subset of miners (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. Throughout the proof, when we compute the coalition $\mathcal{C}$'s utility, we may ignore the $-\$d_a = -\$c_a$ part of the utility since this is a constant offset that is fixed during the preparation phase prior to the protocol execution, and has no effect on the coalition's behavior.

Suppose everyone not in $\mathcal{C}$ plays the honest strategy. Then, the coalition $\mathcal{C}$ can play honestly, i.e., post pre_a to P1 at $t = 0$, and post _ to C1 at time T_2. In this case, $\mathcal{C}$ obtains utility $\$v+\$c_a-\$v_a$.

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 and deposit money into it[4] (see the definition of strategy space in Section 3.2) — 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:

1. First, P1 is successfully activated at some point. Since C1 and C2 are mutually exclusive, and $\$c_a \geq \$v$, $\$v + \$c_a$ is the maximal amount that the coalition can redeem from the PONYTA contract. In this case, the honest Bob will output pre_a and thus the coalition $\mathcal{C}$'s utility is at most $\$v + \$c_a - \$v_a$, which is the same as following the honest strategy.

2. Second, P1 is never activated. In this case, it is impossible for the coalition to redeem any value from P1 or P2, and its utility is at most $\$c_a$. Since $\$v > \$v_a$ by our assumption, following the honest strategy maximizes the utility of $\mathcal{C}$.

□

Lemma 4.2. *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[5]) coalition strategy $S_{\mathcal{C}}$, it must be that*

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

Proof. Just like in the proof of Lemma 4.1, we may ignore the $-\$d_b = -(\$v + \$c_b)$ part of the utility since this a constant fixed prior to the start of protocol execution.

Suppose everyone not in $\mathcal{C}$ plays the honest strategy. If the coalition follows the honest strategy as well, Bob outputs pre_a, and the coalition gets $\$c_b$ from branch C1. Thus, the utility of the coalition is $\$c_b + \$v_b$.

Now, consider the case where the coalition $\mathcal{C}$ deviates from the honest strategy. Just like in the proof of Lemma 4.1, we may assume that the coalition does not post any new contract during the protocol execution.

We may also assume that either P1 or P2 is successfully invoked, since otherwise, the coalition $\mathcal{C}$'s utility is at most $\$v_b + \$c_b$ which is no more than the honest case. There are two possibilities.

First, P1 is successfully activated. In this case, the maximal value the coalition can get from the contract is to activate the C1 branch, and the utility of the coalition is at most $\$v_b + \$c_b$, which is the same as the honest case.

Second, P2 is activated. Let t^* 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^* or earlier. In this case, the coalition's utility is at most $2 \cdot \$v + \$v_b$, 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^* or earlier. Since the honest Alice posts pre_a at $t = 0$, 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

[4] 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.

[5] Bob's coalition can be unbounded as long as the mining process is idealized as in our model.

coalition loses the race, then it gets nothing from C1 or C2, and thus its utility is at most $\$v + \$v_b$. Else if it wins the race, then the coalition's utility is at most $\$v + \$c_b + \$v_b$. 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

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

Recall that $\$c_b > 2 \cdot \$v$. Therefore, for $(\$v + \$v_b) \cdot (1 - p) + (\$v + \$c_b + \$v_b) \cdot p$ to exceed the honest utility $\$c_b + \$v_b$, it must be that $p > \frac{\$c_b + \$v}{\$c_b}$ which is not true by our assumption. We thus conclude that $\mathcal{C}$ cannot increase its utility through any deviation.

□

Theorem 4.3 (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* Ponyta *protocol satisfies γ-CSP-fairness.*

Proof. Lemmas 4.1 and 4.2 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. This is easy to see: since both Alice and Bob are honest, the coalition's utility is 0 unless it can 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 Ponyta is dropout resilient.

Theorem 4.4 (Dropout resilience)**.** Ponyta *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. 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. 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, an honest miner would include Alice's transaction and activate P1. Then, T_2 after P1 is activated, Bob would send an empty message to C1. As long as $1/\mathsf{poly}(\lambda)$ fraction of the mining power is honest, P1 and C1 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 and C1 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. □

4.3 Ponyta Disincentivizes a 100% Coalition

So far, to prove our coalition-resistant fairness notions, we assumed that the coalition wields strictly less than 100% of the mining power. If Bob can solicit a coalition of 100% of the mining power, then its best strategy is to wait for Alice to post pre_a, and then activate P2 and C1. In this way, Bob and the coalition effectively learns the secret pre_a for free.

4.3.1 The Meta-Game of Coalition Formation

We argue that Ponyta *provides strong disincentives for such a 100% coalition to form*, even in a world where one can post bribery contracts [Bon16, JSZ+19, MHM18b, WHF19] or other smart contracts in an attempt to openly solicit everyone.

To see this, we can view the process of soliciting coalition members as a meta-game. Imagine that Bob openly invites miners to join, e.g., through some smart contract. If the coalition wins, the additional gain of at most $\$v$ (relative to the honest case) will be split off among all miners proportional to their mining power, e.g., distributed to each miner who mined a block that starves P1 and C2.

In this meta-game, each miner has two choices, to join or not to join the coalition. It is not hard to see that everyone joining is not an equilibrium of this meta-game. Specifically, if every other miner joined, then I would be strictly better off *not* joining the coalition. If I do not join, the moment P2 is activated, I have a T_2 lead in time to mine a block in which I can redeem $\$v$ from the C2 branch. In particular, we may assume that every miner in the coalition commits to starving C2 in every block they mine, e.g., by placing a collateral that it will honor its commitment — if not, then the coalition will not be stable since a coalition member will be incentivized to defect from the coalition and claim C2 itself. This means that if I mine a block during the T_2 window after the activation of P2, I can claim $\$v$ from C2 for myself. Now, suppose I have γ fraction of mining power, and suppose that $T_2 > 1$. The probability that I mine a block in a window of T_2 length is $1 - (1-\gamma)^{T_2} > \gamma$. Therefore, if I do not join the coalition, my expected gain would be strictly greater than $\$v \cdot \gamma$ which is the expected gain had I joined the coalition. This means that if everyone else joins the coalition, my best strategy is not to join. In other words, a 100% coalition is not an equilibrium of the coalition-forming meta-game.

4.3.2 Comparison with Prior Approaches

Using this coalition formation meta-game perspective, we can give a (hopefully clearer) re-exposition of some incentive attacks described in prior works [MHM18a, Bon, WHF19]. In particular, the earlier work of MAD-HTLC [TYME21] is motivated by the fact that the standard HTLC contract (see Section 2.2), has some coalition formation meta-games in which a 100% coalition is the equilibrium.

Meta-games for HTLC. Bob can post a bribery contract soliciting participation of miners: if Alice's redeeming transaction is censored until Bob claims the $\$v$ back through pre_b, then, Bob will equally re-distribute $\$(v-\epsilon)$ to every miner that helped to mine a block that starved Alice's transaction where $\$\epsilon$ is a small amount Bob keeps for himself. Suppose the transaction fees are 0, then every miner's best strategy is to join the coalition, and thus a 100% coalition is an equilibrium of the meta-game.

If Alice is offering a transaction fee of $\$f$ for her transaction, and assuming that $\$f < \$v/T_1$. Then, Bob can offer $\$(v-\epsilon)/T_1$ to everyone who helps to censor Alice's transaction until Bob could claim the $\$v$ back for himself. In this case, every miner's best strategy is to take the bribe which also effectively leads to a 100% coalition.

Tsabary et al. [TYME21] also describes the following meta-game. Suppose that the mining power of the smallest miner is $\gamma_{\min}$. Bob can offer to pay $\$f'$ to whoever mines the block immediately after T_1 that helps him claim his $\$v$ back. Let $\$f$ be the fee offered by Alice. Suppose that $\$f' \cdot \gamma_{\min} > \$f$, then everyone joining is also an equilibrium of this meta-game (assuming non-myopic miners), effectively leading to a 100% coalition. Roughly speaking, this is because if I give up on my immediate gain $\$f$ right now, there is at least $\gamma_{\min}$ probability that I will be the miner who mines the first block after T_1 which allows me to claim the richer reward $\$f'$ instead.

MAD-HTLC. The result of MAD-HTLC [TYME21] can be viewed as follows: by allowing the miner to claim $\$v$ itself through (pre_a, pre_b), it removes the undesirable 100%-coalition equilibrium in the coalition formation meta-game — the design of Ponyta is inspired by this elegant idea. Unfortunately, the design of MAD-HTLC incentivizes coalitions (with binding side contracts) to deviate in the protocol game itself. As mentioned earlier in Section 2.2, Bob colluding with a miner should always deviate: if it happens to be the miner when Alice posts pre_a, the coalition should always starve Alice's transaction and claim the $\$v$ itself by posting (pre_a, pre_b).

5 Application: Atomic Swap

Atomic swap allows two parties to exchange cryptocurrencies in the same chain or accross two different chains. In this section, we describe how to realize atomic swap by using Ponyta, and we show that the protocol satisfies CSP fairness.

We will describe our protocol for a cross-chain scenario since cross-chain swap is more general than over the same chain. We take Bitcoin and Ethereum as an example. We use $\text{₿}x'$ to denote x' units of Bitcoin, and use $\text{Ξ}x$ to denote x units of Ether. We use the notation $\text{₿}x' + \text{Ξ}x$ to denote x' units of Bitcoin and x units of Ether. In general, our atomic swap protocol can be used on any two chains where the mining process picks block proposers at random proportional either to their mining power or stake.

5.1 Model and Utility

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 oalition, 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'.

Execution model. Since the cross-chain swap application involves two different chains, we need to make a few changes to the execution model. Recall that earlier in Section 3, users can post messages or smart contracts at the beginning of each time step t, and then a miner will be chosen to mine the block of the current time step t. The block mined can contain messages posted at any time less than or equal to t.

Now, we may imagine that users can post messages or at any (wall-clock) time. At any wallclock time t when there is a mining event for either the Bitcoin or Ethereum chain, a miner is chosen, and the newly mined block may contain any message posted at any time less than or equal to t.

In this section, without loss of generality, we may assume that our honest protocol involves one contract on each of the two chains — if there are multiple contracts on the same chain, we can always merge them into a single one. The moment both contracts are posted to their respective blockchains and take effect, we define the current time to be 0, and this marks the start of the protocol execution. The deposit phase in which users deposit money into the two contracts is considered part of the execution.

5.2 Construction

Overview and notations. To realize a cross-chain atomic swap, we use two Ponyta contract instances, denoted Ponyta and Ponyta', for the Ethereum and Bitcoin chains, respectively. The idea is that Alice would act first and redeem Ethers from Ponyta by revealing pre_a, and pre_a is exactly the secret Bob needs to redeem Bitcoins from Ponyta'.

In Section 4.1, we specify the parameters for a Ponyta contract to be a 7-tuple: (h_a, h_b, T_1, T_2, \$$v$, \$$c_a$, \$$c_b$). Since we need two contracts now, we denote the parameters for Ponyta as (h_a, h_b, T_1, τ, $\text{Ξ}x$, $\text{Ξ}c_a$, $\text{Ξ}c_b$), and the parameters for Ponyta' as $(h'_b, h'_a, T'_1, \tau', \text{฿}x', \text{฿}c'_b, \text{฿}c'_a)$. Here, T_1 and T'_1 are the absolute Ethereum time and Bitcoin time, respectively. By contrast, τ and τ' are the number of time intervals measured in terms of the number of Ethereum and Bitcoin blocks, respectively.

Before deploying the contracts, Alice samples pre_a and pre'_a from $\{0,1\}^\lambda$, and Bob samples pre_b from $\{0,1\}^\lambda$ uniformly at random. They then post the Ponyta and Ponyta' contracts on the Ethereum and Bitcoin blockchains, respectively. At this moment, the execution begins and we may rename the current time to be 0. During the protocol execution, Alice and Bob will have a T' window measured in Bitcoin time to deposit Bitcoin into the Ponyta' contract first, and then they will have a T window measured in Ethereum time to deposit Ether into the Ponyta contract.

Parameter choices. The parameters must satisfy the following constaints for the protocol to

satisfy CSP-fairness against coalitions with at most γ fraction of mining power.

Parameter Constraints for Atomic Swap

Constraints for Ponyta (on Ethereum):

- $h_a = H(pre_a)$ and $h_b = H(pre_b)$.
- $T_1 > T$.
- $\ddot{\text{E}}c_a > \ddot{\text{E}}x$ and $\ddot{\text{E}}c_b > 2 \cdot \ddot{\text{E}}x$.

Constraints for Ponyta' (on Bitcoin):

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

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

- $\tau \geq 1$, $\tau' \geq 1$.
- $\gamma^\tau \leq \frac{\$\mathsf{AV}(\ddot{\text{E}}c_a - \ddot{\text{B}}x')}{\$\mathsf{AV}(\ddot{\text{E}}c_a)}$, $\gamma^{\tau'} \leq \frac{\$\mathsf{AV}(\ddot{\text{B}}c'_a - \ddot{\text{B}}x')}{\$\mathsf{AV}(\ddot{\text{B}}c'_a)}$, and $\gamma^\tau \leq \frac{\$\mathsf{BV}(\ddot{\text{E}}c_b - \ddot{\text{E}}x)}{\$\mathsf{BV}(\ddot{\text{E}}c_b)}$.

[a] Notice that $h_a = h'_b$ and Alice holds both pre_a and pre'_a initially.

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

Intuitively, the collateral values must be sufficiently large such that a coalition involving either Alice or Bob would never want to trigger the "bombs" in the protocol that cause their collateral to be (partially) burnt. 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. Finally, the choices of T_1 and T'_1 account for the fact that we want the users to deposit into Ponyta' first before depositing into Ponyta; but the execution order would be the opposite, i.e., Ponyta is executed first and and then Ponyta'.

The contracts. We call the activation points of Ponyta as P1, P2, C1 and C2. Similarly, we call the activation points of Ponyta' as P1', P2', C1' and C2'. We now specify the two contract instances needed for realizing cross-chain atomic swap. Like before, activation points of the same type are mutually exclusive. Although not explicitly noted, we assume that before both parties have made deposits, the contract lets any party back out and withdraw its deposit. When both parties have made deposits, however, redistribution of the money is only possible by invoking one or more activation points of the contract. We assume that the Ponyta contract speaks Etheurem time and the Ponyta' contract speaks Bitcoin time, respectively.

Ponyta contract (on Ethereum) // Parameters: (h_a, h_b, T_1, τ, $\ddot{\text{E}}x$, $\ddot{\text{E}}c_a$, $\ddot{\text{E}}c_b$)

Alice deposits $\ddot{\text{E}}c_a$, and Bob deposits $\ddot{\text{E}}x + \ddot{\text{E}}c_b$.

P1: On receiving pre_a from Alice such that $H(pre_a) = h_a$, send $\ddot{\mathbb{E}}x$ to Alice.

P2: Time T_1 or greater: on receiving pre_b from Bob such that $H(pre_b) = h_b$, or on receiving _ from Alice, send $\ddot{\mathbb{E}}x$ to Bob.

C1: At least τ after either P1 or P2 is activated: on receiving _ from anyone, send $\ddot{\mathbb{E}}c_a$ to Alice and send $\ddot{\mathbb{E}}c_b$ to Bob.

C2: On receive (pre_a, pre_b) from anyone P such that $H(pre_a) = h_a$ and $H(pre_b) = h_b$, send $\ddot{\mathbb{E}}x$ to player P. All remaining coins are burnt.

Ponyta' contract (on Bitcoin) // Parameters: $(h'_b, h'_a, T'_1, \tau', \ddot{\mathbb{B}}x', \ddot{\mathbb{B}}c'_b, \ddot{\mathbb{B}}c'_a)$

Alice deposits $\ddot{\mathbb{B}}x' + \ddot{\mathbb{B}}c'_a$, and Bob deposits $\ddot{\mathbb{B}}c'_b$.

P1': On receiving pre'_b from Bob[a] such that $H(pre'_b) = h'_b$ or on receiving _ from Alice, send $\ddot{\mathbb{B}}x'$ to Bob.

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 $\ddot{\mathbb{B}}x'$ to Alice.

C1': At least τ' after either P1' or P2' is activated: on receiving _ from anyone, send $\ddot{\mathbb{B}}c'_a$ to Alice and send $\ddot{\mathbb{B}}c'_b$ to Bob.

C2': On receiving (pre'_b, pre'_a) from anyone P such that $H(pre'_b) = h'_b$ and $H(pre'_a) = h'_a$, send $\ddot{\mathbb{B}}x'$ to player P. All remaining coins are burnt.

[a] Bob will let pre'_b be the pre_a he learns in the PONYTA instance.

In the above, allowing Alice to invoke P1' by sending _ and allowing Bob to invoke P2' by sending _ are needed for dropout resilience — we will discuss these subtleties in more detail after describe the full coin swap protocol. Additionally, we let Alice post an empty message _ to invoke P2.

Coin swap protocol. We now describe our coin swap protocol.

- *Miner.* In the honest protocol, a miner watches all transactions posted to P1, P2, C2, P1', P2', and C2', and see if they contain correct values for $pre_a = pre'_b$, pre_b, and pre'_a. As soon as it observes both pre_a and pre_b, it posts (pre_a, pre_b) to C2. As soon as it observes both pre'_a and $pre'_b = pre_a$, it posts (pre'_b, pre'_a) to C2'.
- *Alice and Bob.* Below, we define the honest protocol for Alice and Bob. Unless otherwise noted, Alice and Bob speak absolute wall-clock time. As mentioned, the start of the execution (i.e., time 0) is defined to be the moment both contracts have been posted and take effect.

Atomic Swap Protocol

Deposit Phase:

1. At $t = 0$, Alice and Bob deposit $\ddot{\mathbb{B}}x' + \ddot{\mathbb{B}}c'_a$ and $\ddot{\mathbb{B}}c'_b$ into PONYTA', respectively.
2. At Bitcoin time T': if at least one party has not deposited the appropriate amount into PONYTA', go to the abort phase; else, Alice and Bob deposit $\ddot{\mathbb{E}}c_a$ and $\ddot{\mathbb{E}}x + \ddot{\mathbb{E}}c_b$ into PONYTA,

respectively.

3. At Ethereum time T: if at least one party has not deposited the appropriate amount into Ponyta, then go to the abort phase; else go to the swap phase.

Swap Phase:

1. At Ethereum time T, 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 As soon as P2 has been activated, Bob sends an empty message _ to P2'.
3. When τ *Ethereum time* has passed after P1 or P2 is activated, Alice and Bob posts an empty message _ to C1.
4. 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.
5. When τ' *Bitcoin time* has passed after P1' or P2' is activated, Alice and Bob each posts an empty message _ to C1'.

Abort Phase:

1. For either Ponyta or Ponyta': if Alice (or Bob) has submitted a deposit transaction, then submit a corresponding withdrawal transaction[a].
2. Alice posts pre'_a to P2'. Once P2' has been activated, she posts _ to P2.
3. Bob posts pre_b to P2 (regardless of whether deposits have been made). Once P2 has been activated or Bob has withdrawn his deposit from Ponyta or Bob entered the abort phase prior to depositing into Ponyta, Bob posts _ to P2';
4. If τ' *Bitcoin time* has passed since P1' or P2' is activated, send _ to C1'; similarly, if τ *Ethereum time* has passed since P1 or P2 is activated, send _ to C1.

[a] We may assume that Alice's withdrawal transaction is valid only if it is included after Alice's deposit transaction, and moreover, Bob's deposit transaction has not been included. The same holds for Bob's withdrawal transaction.

As mentioned, the coin swap protocol consists of two contract instances, one running on each blockchain. Once the players have deposited money into the contracts, Alice will redeem Ether from the Ponyta contract first by posting pre_a to P1. This in turn allows Bob to redeem Bitcoin from Ponyta' by posting $pre'_b = pre_a$ to P1'.

Some subtleties. Observe that users deposit into Ponyta' first before Ponyta, but the execution order is reversed, i.e., Ponyta is executed first before Ponyta'. This is to prevent the attack where Alice redeems the Ethers from Ponyta first, but does not deposit into Ponyta' which prevents Bob from redeeming Bitcoins from Ponyta'.

In comparison with our contract in Section 4.1, there are a couple small modification in Ponyta'. We now additionally allow Alice to trigger P1' by posting an empty message _, and allow Bob to trigger P2' by posting _. These modifications are needed for dropout resilience. Specifically, the latter modification is needed in the following situation: if Alice aborts prior to posting her pre_a, we want to make sure that Bob can activate P2' (after activating P2 or withdrawing his deposit from Ponyta), so that he can get his deposit back from Ponyta'. The former modification is needed in the following situation. Suppose Alice has posted pre_a to P1 but Bob

aborts. In this case, Alice should be able to trigger either P1' or P2'. Since Alice has already posted pre_a, it is not safe for her to post pre'_a to P2' since making both pre_a and pre'_a public can cause the bomb to be triggered. Therefore, we want Alice to be able to trigger P1' by posting $\_$ after P1 is activated, such that she can eventually get her deposit back from PONYTA'.

5.3 Proof of CSP-Fairness

Lemma 5.1. *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 $\gamma^\tau \leq \frac{\$\mathsf{AV}(\text{Ð}c_a - \text{Ð}x')}{\$\mathsf{AV}(\text{Ð}c_a)}$ and $\gamma^{\tau'} \leq \frac{\$\mathsf{AV}(\text{Ð}c'_a - \text{Ð}x')}{\$\mathsf{AV}(\text{Ð}c'_a)}$, 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. If $\mathcal{C}$ follows the honest strategy, P1, C1, P1', and C1' will be activated, and the utility of $\mathcal{C}$ is $\$\mathsf{AV}(\text{Ð}x - \text{Ð}x') > 0$. Now, we analyze the cases if $\mathcal{C}$ deviates from the honest strategy, and show that the utility never exceeds the honest case. We may assume that the coalition does not post any new smart contract on the fly and deposit money into it (see the definition of strategy space in Section 3.2) — if it did so, it cannot recover more than its deposit since any player not in $\mathcal{C}$ will not invoke the smart contract. In the following, we do the case analysis for each phase.

Bob goes to the abort phase at Bitcoin time T'. In this case, Alice fails to make her deposit into PONYTA' by Bitcoin time T'. Further, Bob never deposits anything into PONYTA, so the activation points of PONYTA are never invoked. Henceforth we may assume that $\mathcal{C}$ can withdraw whatever it deposits into PONYTA. If Alice deposits $\text{Ð}x' + \text{Ð}c'_a$ into PONYTA' before Bob withdraws $\text{Ð}c'_b$, both parties make their deposits into PONYTA'. Then, $\text{Ð}x' + \text{Ð}c'_a$ is the optimal amount that $\mathcal{C}$ can get from PONYTA' (from P2' and C1'), which is the same amount that Alice deposits into PONYTA'. On the other hand, if Alice does not deposit $\text{Ð}x' + \text{Ð}c'_a$ into PONYTA' before Bob withdraws $\text{Ð}c'_b$, the activation points of PONYTA' are never invoked. No matter in which case, the utility of $\mathcal{C}$ is 0.

Bob goes to the abort phase at Ethereum time T. In this case, both parties make their deposits into PONYTA', and Bob also already deposited $\text{Ð}x + \text{Ð}c_b$ into PONYTA. If Alice does not deposit $\text{Ð}c_a$ into PONYTA before Bob withdraws $\text{Ð}x + \text{Ð}c_b$, the activation points of PONYTA are never invoked. Then, the optimal utility of $\mathcal{C}$ is 0 by activating P2' and C1'.

Below, we focus on the case when Alice has deposited $\text{Ð}c_a$ into PONYTA before Bob withdraws $\text{Ð}x + \text{Ð}c_b$, i.e., both parties make their deposits into PONYTA. Since $\text{Ð}c_a > \text{Ð}x$ and $\text{Ð}c'_a > 2 \cdot \text{Ð}x'$, the utility of $\mathcal{C}$ exceeds $\$\mathsf{AV}(\text{Ð}x - \text{Ð}x')$ only if P1, C1, P2', and C1' are activated, where $\mathcal{C}$ has the utility $\$\mathsf{AV}(\text{Ð}x)$. In the following, we will show that, because $\mathcal{C}$ only controls at most γ fraction of mining power, its expected utility cannot exceed $\$\mathsf{AV}(\text{Ð}x - \text{Ð}x')$ if P1 is activated.

Suppose P1 is activated at some Ethereum time t^*. Because Bob goes to the abort phase at Ethereum time T, Alice must make her deposit into PONYTA at some Ethereum time $t^* \geq T$. Since Bob sends pre_b to P2 at Ethereum time T as part of the abort procedure, both pre_a and pre_b are publicly known after t^*. Thus, during Ethereum time $t = (t^*, t^* + \tau]$, any miner in $-\mathcal{C}$ will activate C2 if it mines 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, so its utility is $\$\mathsf{AV}(\text{Ð}x - \text{Ð}c_a)$ (from P1, P2' and C1'). Else if $\mathcal{C}$ wins the race, then its utility is at most $\$\mathsf{AV}(\text{Ð}x)$, and the maximum utility can be obtained only by activating P1, C1,

P2' and C1'. 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

$$\$\mathsf{AV}((\ddot{\mathbb{E}}x - \ddot{\mathbb{E}}c_a) \cdot (1-p) + \ddot{\mathbb{E}}x \cdot p).$$

Since $p \leq \gamma^{\tau} \leq \frac{\$\mathsf{AV}(\ddot{\mathbb{E}}c_a - \ddot{\mathbb{B}}x')}{\$\mathsf{AV}(\ddot{\mathbb{E}}c_a)}$, we have

$$\$\mathsf{AV}((\ddot{\mathbb{E}}x - \ddot{\mathbb{E}}c_a) \cdot (1-p) + \ddot{\mathbb{E}}x \cdot p) < \$\mathsf{AV}(\ddot{\mathbb{E}}x - \ddot{\mathbb{B}}x').$$

Bob goes to the swap phase. In this case, Alice and Bob both make their deposits into Ponyta and Ponyta', so the four activiation points of each contract can be activated. In the swap phase, $\mathcal{C}$ can behave in the following ways.

1. Suppose Alice sends pre_a to P1 before Ethereum time T_1. As we have shown, the utility of $\mathcal{C}$ exceeds $\$\mathsf{AV}(\ddot{\mathbb{E}}x - \ddot{\mathbb{B}}x')$ only if P1, C1, P2', and C1' are activated. Suppose P2' is activated at some Bitcoin time $t^* \geq T_1'$, and pre_a' is publicly known afterward. Moreover, because the parameter constraints guarantee that the Bitcoin block of height T' is mined after the Ethereum block of height T is mined, when P2' is activated, pre_a is also publicly known. 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 $\$\mathsf{AV}(\ddot{\mathbb{E}}c_a)$ from the union of C1 and C2 since $\ddot{\mathbb{E}}c_a > \ddot{\mathbb{E}}x$. Thus, its utility is at most $\$\mathsf{AV}(\ddot{\mathbb{E}}x - \ddot{\mathbb{B}}c_a')$ (from P1, C1 and P2'). Else if $\mathcal{C}$ wins the race, then its utility is at most $\$\mathsf{AV}(\ddot{\mathbb{E}}x)$ which can be achieved by activating P1, C1, P2' and C1'. 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

   $$\$\mathsf{AV}((\ddot{\mathbb{E}}x - \ddot{\mathbb{B}}c_a') \cdot (1-p) + \ddot{\mathbb{E}}x \cdot p).$$

   Since $p \leq \gamma^{\tau'} \leq \frac{\$\mathsf{AV}(\ddot{\mathbb{B}}c_a' - \ddot{\mathbb{B}}x')}{\$\mathsf{AV}(\ddot{\mathbb{B}}c_a')}$, we have

   $$\$\mathsf{AV}((\ddot{\mathbb{E}}x - \ddot{\mathbb{B}}c_a') \cdot (1-p) + \ddot{\mathbb{E}}x \cdot p) < \$\mathsf{AV}(\ddot{\mathbb{E}}x - \ddot{\mathbb{B}}x').$$

2. Suppose Alice does not send pre_a to P1 before Ethereum time T_1. In this case, Bob will send pre_b to P2 at Ethereum time T_1. If it turns out that P2 is activated, the utility of $\mathcal{C}$ is at most 0 (from C1, P2' and C1'), which is less than the honest case. On the other hand, suppose P1 is activated at some Ethereum time $t^* \geq T_1$. Then, both pre_a and pre_b are publicly known after Ethereum time t^*. Using the same argument we used for the case "Bob goes to the abort phase at Ethereum time T", since $p \leq \gamma^{\tau} \leq \frac{\$\mathsf{AV}(\ddot{\mathbb{E}}c_a - \ddot{\mathbb{B}}x')}{\$\mathsf{AV}(\ddot{\mathbb{E}}c_a)}$, the utility of $\mathcal{C}$ is upperbounded by

   $$\$\mathsf{AV}((\ddot{\mathbb{E}}x - \ddot{\mathbb{E}}c_a) \cdot (1-p) + \ddot{\mathbb{E}}x \cdot p) < \$\mathsf{AV}(\ddot{\mathbb{E}}x - \ddot{\mathbb{B}}x').$$

□

Lemma 5.2. *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 $\gamma^{\tau} \leq \frac{\$\mathsf{BV}(\ddot{E}c_b - \ddot{E}x)}{\$\mathsf{BV}(\ddot{E}c_b)}$, 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. If $\mathcal{C}$ follows the honest strategy, P1, C1, P1', and C1' will be activated, and the utility of $\mathcal{C}$ is $\$\mathsf{BV}(\text{₿}x' - \text{Ð}x) > 0$. Now, we analyze the cases if $\mathcal{C}$ deviates from the honest strategy, and show that the utility never exceeds the honest case. We may assume that the coalition does not post any new smart contract on the fly and deposit money into it — if it did so, it cannot recover more than its deposit since any player not in $\mathcal{C}$ will not invoke the smart contract.

Alice aborts. In this case, Alice never sends pre_a to P1. Since $H(\cdot)$ is one-way and the coalition $\mathcal{C}$ is PPT, except with negligible probability, P1' is never activated. Since $\text{₿}c'_b > \text{₿}x'$, the coalition $\mathcal{C}$ can get at most $\text{₿}c'_b$ from Ponyta' which is the same as its deposit.

We now analyze the Ponyta contract. There are three cases:

- Alice never deposited into Ponyta— this can happen if Alice aborts at Bitcoin time T'. In this case, the coalition $\mathcal{C}$ cannot gain anything from Ponyta.
- Alice deposited into Ponyta (which can happen if Alice aborts at Ethereum time T) but Bob fails to deposit into Ponyta before Alice makes her withdrawal from Ponyta. In this case, the four activation points of Ponyta are never activated, and $\mathcal{C}$ cannot gain anything from Ponyta.
- Alice deposited into Ponyta and Bob manages to deposit into Ponyta before Alice makes her withdrawal from Ponyta. Since $\text{Ð}c_b > \text{Ð}x$, the coalition $\mathcal{C}$ can get at most $\text{Ð}(c_b + x)$ from Ponyta which is the same its deposit.

In all cases, $\mathcal{C}$'s utility is at most 0.

Alice goes to the swap phase. In this case, Alice and Bob both make their deposits into Ponyta and Ponyta', so the four activiation points of each contract can be activated. Recall that if $\mathcal{C}$ plays honestly, P1, C1, P1', and C1' will be activated and its utility is $\$\mathsf{BV}(\text{₿}x' - \text{Ð}x)$. Because $\text{Ð}c_b > 2 \cdot \text{Ð}x$ and $\text{₿}c'_b > \text{₿}x'$, the utility of $\mathcal{C}$ exceeds $\$\mathsf{BV}(\text{₿}x' - \text{Ð}x)$ only if P2, C1, P1', and C1' are activated, where $\mathcal{C}$ has the utility $\$\mathsf{BV}(\text{₿}x')$.

Because Alice is honest, she always sends pre_a to P1 at Ethereum time T. Now, suppose P2 is activated at t^*. Notice that P2 can be activated only after $T_1 > T$, at which pre_a is already publicly known. Notice that the activation of P2 implies pre_b is publicly known. Thus, during Ethereum time $(t^*, t^* + \tau]$, any miner in $-\mathcal{C}$ will activate C2 if it mines 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 since $\text{₿}c'_b > \text{₿}x'$, $\mathcal{C}$ can get at most $\$\mathsf{BV}(\text{₿}c'_b)$ from the union of C1' and C2'. Thus its utility is at most $\$\mathsf{BV}(\text{₿}x' - \text{Ð}c_b)$ which is achieved by activating from P2, P1' and C1'. Else if $\mathcal{C}$ wins the race, then its utility is at most $\$\mathsf{BV}(\text{₿}x')$ which is achieved by invoking P2, C1, P1' and C1'. The probability p that $\mathcal{C}$ wins the race is upper bounded by $p \le \gamma^\tau$. Therefore, the expected utility of $\mathcal{C}$ is upper bounded by

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

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

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

□

Theorem 5.3 (CSP fairness for coin swap)**.** *Suppose that the hash function $H(\cdot)$ is a one-way function. Moreover, suppose $\gamma^\tau \le \frac{\$\mathsf{AV}(\text{Ð}c_a - \text{₿}x')}{\$\mathsf{AV}(\text{Ð}c_a)}$, $\gamma^{\tau'} \le \frac{\$\mathsf{AV}(\text{₿}c'_a - \text{₿}x')}{\$\mathsf{AV}(\text{₿}c'_a)}$, and $\gamma^\tau \le \frac{\$\mathsf{BV}(\text{Ð}c_b - \text{Ð}x)}{\$\mathsf{BV}(\text{Ð}c_b)}$. Then, the atomic swap protocol satisfies γ-CSP-fairness.*

Proof. Lemma 5.1 and Lemma 5.2 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.

Henceforth, we consider the miner-only coalition, so Alice and Bob always follow the protocol. In the protocol, Alice and Bob decide whether they will go to the abort phase at Bitcoin time T' and at Ethereum time T, i.e., they make this decision using on-chain state when the blockchains have reached lengths T' and T, respectively. Therefore, both Alice and Bob go to the swap phase or both of them go to the abort phase. Now, a miner-only coalition can get positive utility only if C2 or C2' is activated (or both). If both Alice and Bob go to the swap phase, they would never post any transaction containing pre_b or pre'_a. Since $H(\cdot)$ is one-way and the coalition is PPT, C2 and C2' are never activated except with negligible probability. On the other hand, if both Alice and Bob go to the abort phase, they would never post any transaction containing $pre_a = pre'_b$. Since $H(\cdot)$ is one-way and the coalition is PPT, C2 and C2' are never activated except with negligible probability. To sum up, except with negligible probability, the utility of the coalition is at most 0, which is the same as the honest case. □

5.4 Proof of Dropout Resilience

We now prove dropout resilience.

Theorem 5.4 (Dropout resilience of coin swap)**.** *Our atomic swap protocol 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. We first analyze the case where Alice drops out. There are three possible cases:

1. Bob goes to the abort phase at Bitcoin time T'.
2. Bob goes to the abort phase at Ethereum time T.
3. Bob goes to the swap phase.

In the first case, Alice does not make her deposit into Ponyta', and both parties have not deposited into Ponyta. By assumption, Bob can always withdraw his deposit in Ponyta'. In the second case, both parties already made their deposits into Ponyta', while Alice does not make her deposit into Ponyta. Then, Bob can always withdraw his deposit in Ponyta. Moreover, Bob will send _ to P2' after he has withdrawn his deposit from Ponyta, and send _ to C1' sufficiently long after P2' is activated. Therefore, except with negligible probability, P2' and C1' will always be activated. Thus, Bob simply gets all his deposit back. In the third case, if Alice sends pre_a to P1 before she drops out, Bob learns $pre'_b = pre_a$ so he can send it to P1'. Consequently, P1, C1, P1' and C1' will always be activated except with negligible probability. Then, Bob's utility is $\$\mathsf{BV}(\text{₿}x' - \text{Ξ}x) > 0$. On the other hand, if Alice drops out before sending pre_a to P1, Bob will send pre_b to P2 at $t = T_1$ and he will send _ to P2' when P2 has been activated. Except with negligible probability, P2, C1, P2' and C1' will always be activated. Then, Bob's utility is 0, since he simply gets all his deposit back.

Next, we analyze the case where Bob drops out. There are three possible cases:

1. Alice goes to the abort phase at Bitcoin time T'.

2. Alice goes to the abort phase at Ethereum time T.

3. Alice goes to the swap phase.

In the first case, Bob does not make his deposit into Ponyta', and both parties have not deposited into Ponyta. By assumption, Alice can always withdraw her deposit in Ponyta'. In the second case, both parties already made their deposits into Ponyta', while Bob does not make his deposit into Ponyta. Then, Alice always sends pre'_a to P2' and $\_$ to C1'. Except with negligible probability, P2' and C1' will always be activated. Moreover, Alice can always withdraw her deposit in Ponyta. Thus, Alice simply gets all her deposit back. In the third case, both parties already made their deposits into Ponyta' and Ponyta. In this case, Alice always sends pre_a to P1, and except with negligible probability, P1 and C1 will be activated in polynomial-time. Then, Alice will sends an empty message $\_$ to P1', and except with negligible probability, P1' and C1' will be activated in polynomial-time. Consequently, because P1, C1, P1' and C1' are activated, Alice's utility is $\$\mathsf{AV}(\ddot{\mathrm{E}}x - \ddot{\mathrm{B}}x') > 0$. □

6 Instantiating Ponyta in Bitcoin

First, we show how to instantiate Ponyta using Bitcoin scripts. We start by introducing notation for payments in Bitcoin.

6.1 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*. 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.

Transactions can be posted on the Bitcoin 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. 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 $i \in [n]$, where Φ_i's were the scripts associated with the input addresses of the transaction. A transaction is confirmed if it is included in the blockchain.

Thus, for Bitcoin, the logic of Ponyta 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 Ponyta 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[6] $H : \{0,1\}^* \to \{0,1\}^\kappa$, (2) transaction timestamp verification wrt. current height of the blockchain denoted by _NOW[7] 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. [8] We say an address Adr (associated script Φ) is controlled by a user if the user knows a witness x s.t. $\Phi(x) = 1$.

6.2 High-Level Idea

Recall that earlier we described that Alice and Bob deposit coins into a smart contract and the coins are held by the smart contract. In Bitcoin, Alice and Bob instead prepare a setup transaction tx_{stp} which deposit their coins into a payment address Adr_{pay} and a collateral address Adr_{col}. They also prepare and sign all the redeeming transactions upfront which allow them to later redeem from the addresses Adr_{pay} and Adr_{col}. The script associated with Adr_{pay} specifies the two conditions under which the coins in the address Adr_{pay} can be redeemed, corresponding to the earlier P1 and P2, respectively. The script associated with Adr_{col} encodes the spending conditions C1 and C2, respectively.

For some parts of the Ponyta protocol, the limited scripting capability of Bitcoin presents a technical challenge. Specifically, it is not immediately clear how to instantiate *conditional timelock redeem*, that allows the players to redeem from Adr_{col} only some time after Adr_{pay} has been spent, and how to achieve *conditional burn*, that allows players to burn a fraction of the coins in Adr_{col} if either Alice or Bob misbehaves. To achieve the former, we introduce two auxiliary addresses which are specified as output addresses of the payment redeem transactions. The auxiliary addresses receive a small ϵ amount of coin (e.g., 1 Satoshi) when the payment redeem transactions spend from Adr_{pay}. The collateral redeem transactions that redeem from Adr_{col} through C1 additionally specify one of the auxiliary addresses as an input address (depending on whether P1 or P2 was invoked). The script associated with the auxiliary addresses can now specify the timelock condition for the collateral redeem transactions. To achieve the latter we ensure that a redeeming transaction that redeems the collateral coins according to C2 is prepared in advance by Alice and Bob, and is made available to every other player in the network. This transaction is set such that the coins to be burnt in Adr_{col} are transferred to some *irredeemable* address, while the rest of the coins are left as transaction fee for the miner including the transaction in his block. We discuss the instantiation in more detail below.

6.3 Addresses, Scripts, and Transactions for Ponyta

We now describe all the transactions, addresses, and scripts needed in Ponyta's Bitcoin instantiation.

Setup. In Ponyta's Bitcoin instantiation, the players place their deposits into two addresses, the payment address Adr_{pay} and the collateral address Adr_{col}. During the preparation phase, Alice and

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

[7] 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.

[8] 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: PONYTA's transcations in Bitcoin.

	Description
tx_{stp}	$tx\left(\begin{matrix}[(Adr_0^A, \Phi_0^A, \$c_a), (Adr_0^B, \Phi_0^B, \$v + \$c_b)], \\ [(Adr_{\mathsf{pay}}, \Phi_{\mathsf{pay}}, \$v), (Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_a + \$c_b)]\end{matrix}\right)$
tx_{P1}	$tx\left(\begin{matrix}[(Adr_{\mathsf{pay}}, \Phi_{\mathsf{pay}}, \$v)], \\ [(Adr_1^A, \Phi_1^A, \$v - \$\epsilon), (Adr_{\mathrm{P1}}^{\mathsf{ax}}, \Phi_{\mathrm{P1}}^{\mathsf{ax}}, \$\epsilon)]\end{matrix}\right)$
tx_{P2}	$tx\left(\begin{matrix}[(Adr_{\mathsf{pay}}, \Phi_{\mathsf{pay}}, \$v)], \\ [(Adr_1^B, \Phi_1^B, \$v - \$\epsilon), (Adr_{\mathrm{P2}}^{\mathsf{ax}}, \Phi_{\mathrm{P2}}^{\mathsf{ax}}, \$\epsilon)]\end{matrix}\right)$
tx_{C1}^1	$tx\left(\begin{matrix}[(Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_a + \$c_b), (Adr_{\mathrm{P1}}^{\mathsf{ax}}, \Phi_{\mathrm{P1}}^{\mathsf{ax}}, \$\epsilon)], \\ [(Adr_2^A, \Phi_2^A, \$c_a + \$\epsilon), (Adr_2^B, \Phi_2^B, \$c_b)]\end{matrix}\right)$
tx_{C1}^2	$tx\left(\begin{matrix}[(Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_a + \$c_b), (Adr_{\mathrm{P2}}^{\mathsf{ax}}, \Phi_{\mathrm{P2}}^{\mathsf{ax}}, \$\epsilon)], \\ [(Adr_3^A, \Phi_3^A, \$c_a), (Adr_3^B, \Phi_3^B, \$c_b + \$\epsilon)]\end{matrix}\right)$
tx_{C2}	$tx\left(\begin{matrix}[(Adr_{\mathsf{col}}, \Phi_{\mathsf{col}}, \$c_a + \$c_b)], \\ [(Adr_{\mathsf{burn}}, \Phi_{\mathsf{burn}}, \$c_a + \$c_b - \$v)]\end{matrix}\right)$

Bob prepare the setup transaction tx_{stp} that moves their 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}, with $\$v - \$\epsilon$ coins paid to Alice's or Bob's address, respectively, and $\$\epsilon$ coins paid to an auxiliary address which will later be needed 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 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}^1, tx_{C1}^2 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 $\$c_a + \$c_b - \$v$ coins to a burn-address Adr_{burn}, and the remaining to the miner who mines the block.

The transactions tx_{C1}^1 and tx_{C1}^2 redeem from the C1 branch, depending on which payment transaction tx_{P1} or tx_{P2} respectively, was activated earlier.

To implement the timeout condition associated with the C1 branch, tx_{C1}^1 and tx_{C1}^2 also each refer to an auxiliary input address, $Adr_{\mathrm{P1}}^{\mathsf{ax}}$ and $Adr_{\mathrm{P2}}^{\mathsf{ax}}$, respectively. These two auxiliary addresses are outputs of payment transactions tx_{P1} and tx_{P2} respectively. Addresses $Adr_{\mathrm{P1}}^{\mathsf{ax}}$ and $Adr_{\mathrm{P2}}^{\mathsf{ax}}$ have scripts $\Phi_{\mathrm{P1}}^{\mathsf{ax}}$ and $\Phi_{\mathrm{P2}}^{\mathsf{ax}}$, respectively 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 addresses $Adr_{\mathrm{P1}}^{\mathsf{ax}}$ and $Adr_{\mathrm{P2}}^{\mathsf{ax}}$, 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,

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

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** $(\_\text{NOW} > T_1) \wedge (H(pre_b) = h_b)$
$\wedge (\mathsf{Vf}(\mathsf{pk}'_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}'_b, tx, \sigma_b) = 1)$
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** $(\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1) \wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$
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

$\Phi^{\mathsf{ax}}_{\text{P1}}(tx, \sigma_a, \sigma_b)$

if $(\_\text{NOW} > T_2) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1)$
$\wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$ **then return** 1
// Values $T_2, \mathsf{pk}_a, \mathsf{pk}_b$ are hardwired

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

if $(\_\text{NOW} > T_2) \wedge (\mathsf{Vf}(\mathsf{pk}_a, tx, \sigma_a) = 1)$
$\wedge (\mathsf{Vf}(\mathsf{pk}_b, tx, \sigma_b) = 1)$ **then return** 1
// Values $T_2, \mathsf{pk}_a, \mathsf{pk}_b$ are hardwired

Figure 1: The description of scripts $\Phi_{\mathsf{pay}}, \Phi_{\mathsf{col}}, \Phi^{\mathsf{ax}}_{\text{P1}}$ and $\Phi^{\mathsf{ax}}_{\text{P2}}$. 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. In Φ_{pay} (and Φ_{col}) either activation point P1 (C1) or activation point P2 (C2) must be satisfied for the script to return 1.

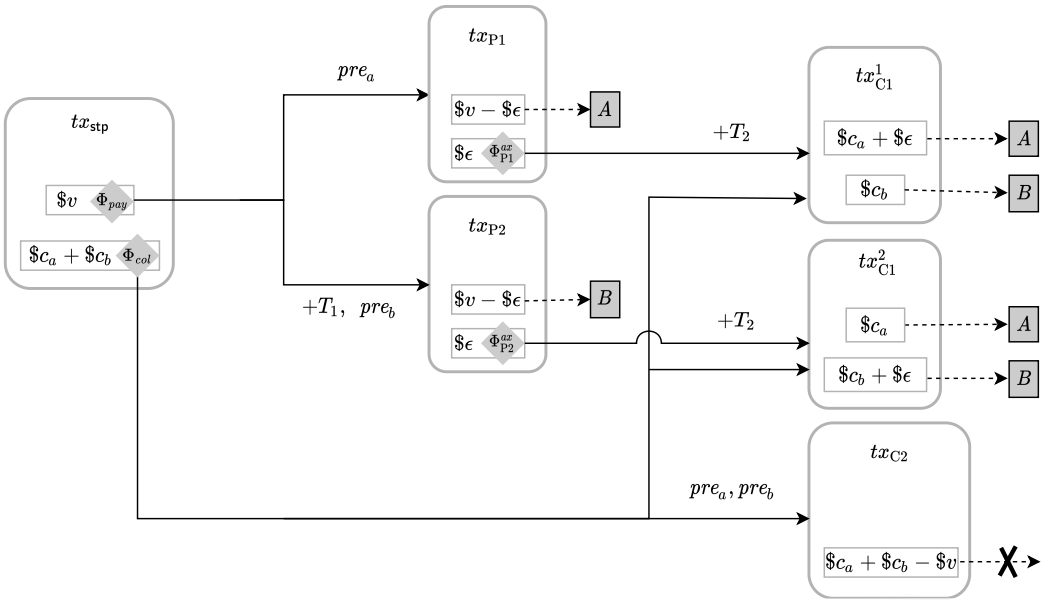

Figure 2: The transaction flow of PONYTA in Bitcoin. Rounded boxes denote transactions, rectangles within are outputs of the transaction. Incoming arrows denote transaction inputs, outgoing arrows denote how an output can be spent by a transaction at the end of the arrow. Solid lines indicate the transaction output can be spent only if both users sign the spending transaction. Dotted lines 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.

assuming security of the signature scheme, no other transaction is able to spend from the addresses Adr_{pay}, Adr_{col}, and the auxiliary addresses $Adr^{\mathsf{ax}}_{\text{P1}}$ and $Adr^{\mathsf{ax}}_{\text{P2}}$, besides those that they are connected to through a solid line in Figure 2.

6.4 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 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 $Adr^{\mathsf{ax}}_{\text{P1}}$ ($Adr^{\mathsf{ax}}_{\text{P2}}$) 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 Alice (Bob). Note that $Adr^{\mathsf{ax}}_{\text{P1}}$ ($Adr^{\mathsf{ax}}_{\text{P2}}$) is an address created when the payment address is redeemed via tx_{P1} by Alice (tx_{P2} by Bob). $Adr^{\mathsf{ax}}_{\text{P1}}$ 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 $Adr^{\mathsf{ax}}_{\text{P1}}$ was created on the blockchain (similar for $Adr^{\mathsf{ax}}_{\text{P2}}$). By allowing only transactions tx^1_{C1} and tx^2_{C1} to spend from branch C1 of Adr_{col} and requesting that they *also* spend from $Adr^{\mathsf{ax}}_{\text{P1}}$ or $Adr^{\mathsf{ax}}_{\text{P2}}$, 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 $Adr^{\mathsf{ax}}_{\text{P1}}$ or $Adr^{\mathsf{ax}}_{\text{P2}}$ are redeemed *simultaneously*. This effectively enforces a timelock of T_2 (after tx_{P1} or tx_{P2} is published on the blockchain) on the redeeming of the collateral coins by Alice and Bob. We refer to this technique of enforcing a conditional timelock on coins through simultaneous spending of a timelocked address

as *conditional timelock redeem*. The addresses $Adr^{\mathsf{ax}}_{\mathrm{P1}}$ and $Adr^{\mathsf{ax}}_{\mathrm{P2}}$ only hold a very small amount of value $\$\epsilon$, their only role is to enable the above mechanism. The auxiliary address scripts $\Phi^{\mathsf{ax}}_{\mathrm{P1}}$ and $\Phi^{\mathsf{ax}}_{\mathrm{P2}}$ are 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[9]), 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.

6.5 Protocol

During the *preparation phase*, Alice and Bob prepare the setup transaction tx_{stp}. The transaction must be signed by both Alice and Bob to take effect. However, before signing tx_{stp}, Alice and Bob prepare and sign all redeeming transactions, including tx_{P1}, tx_{P2}, tx^1_{C1}, tx^2_{C1}, and tx_{C2}. Alice and Bob now broadcast all[10] these transactions 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, Alice and Bob both reveal their 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, Alice and Bob can redeem the collateral after waiting for time T_2 using either tx^1_{C1} (if Alice redeemed the payment) or tx^2_{C1} (if Bob redeemed the payment). 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 at the end of the preparation phase 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 $\$v$ coins as fee from the transaction while $\$c_a + \$c_b - \$v$ are burnt.

6.6 Estimated Transaction Costs

Standard Bitcoin transactions let addresses of Alice and Bob to be *pay-to-public key-hash* scripts, and the scripts Φ_{pay} and Φ_{col} can be initiated using *pay-to-script-hash* scripts as done in [TYME21]. We can optimize the size of the scripts by using shared public keys between Alice and Bob instead of separate keys. For example, instead of requiring signatures wrt. pk_a and pk_b for the payment redeem tx_{P1}, we require a single signature wrt. pk_{ab} which is a shared public key between Alice and Bob. This means that users need to jointly generate a single signature instead of separately generating 2 signatures under independent public keys. With this, we estimate the size of our payment address script to be 121 bytes and our collateral address script to be 115 bytes. Excluding the standard

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

[10] In practice, Alice and Bob only need to broadcast the transaction tx_{stp}, and the transaction tx_{C2} and its signatures.

Bitcoin transaction overhead like version number, transaction id, number of inputs, outputs etc, the size of the setup transaction tx_{stp} is 320 bytes, our payment redeem transactions (both tx_{P1} and tx_{P2}) are 290 bytes each when using ECDSA signatures. Our collateral redeem transactions tx^1_{C1} and tx^2_{C1} each are of size 366 bytes, while tx_{C2} is of size 228 bytes, again excluding the standard overheads.

7 Instantiating Ponyta in Ethereum

We implemented Ponyta in Solidity, the smart contract language used in Ethereum which supports general smart contracts. In Ethereum, the price of a transaction depends on its *gas* usage, which describes the cost of each operation performed by the transaction in units specific to Ethereum implementation. In Table 2, we compare the gas costs of various operations of Ponyta with those of MAD-HTLC and HTLC. The gas cost for the initial deployment of the contract far outweighs those of the redeem transactions. As expected, Ponyta incurs costs that are a bit higher than those of MAD-HTLC, as Ponyta contains slightly more code (103 LoC in Ponyta vs. 72 in MAD-HTLC).

Our implementation in Ethereum is straight-forward and consists of a single contract which includes the initialization as well as all redeem paths.

We deployed Ponyta on Rinkeby, Ethereum's testnet. We posted transactions redeeming the payment through paths P1 and P2 (see Section 4.1), as well as the transaction redeeming the collateral through path C1, and the transaction which sends a payout to a user and causes coin burning through path C2 (Section 4.1).

Table 2: Ethereum gas cost comparison.

Contract	**Deploy (Gas)**	**Redeem path**	**Gas**
HTLC	380,159	Alice redeem	35,851
		Bob redeem	34,932
MAD-HTLC	581,002	Dep. Alice	60,239
		Dep. Bob	62,345
		Dep. Miner	61,008
		Col. Bob	42,266
		Col. Miner	46,063
Ponyta	759,859	Dep. Alice	67,748
		Dep. Bob	69,785
		Col. Alice/Bob	53,698
		Col. Miner	51,936

References

[AHS22] Sepideh Avizheh, Preston Haffey, and Reihaneh Safavi-Naini. Privacy-preserving fairswap: Fairness and privacy interplay. *Proc. Priv. Enhancing Technol.*, 2022.

[Aso98] N. Asokan. *Fairness in Electronic Commerce*. PhD thesis, 1998.

[ASW97] N. Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *ACM CCS*, 1997.

[atoa] Submarine swap in lightning network. https://wiki.ion.radar.tech/tech/research/submarine-swap.

[atob] What is atomic swap and how to implement it. https://www.axiomadev.com/blog/what-is-atomic-swap-and-how-to-implement-it/.

[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.

[BKa] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In *CRYPTO*.

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

[BK14] Iddo Bentov and Ranjit Kumaresan. How to Use Bitcoin to Design Fair Protocols. In *CRYPTO*, 2014.

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

[Bon16] Joseph Bonneau. Why buy when you can rent? In *International Conference on Financial Cryptography and Data Security*, pages 19–26. Springer, 2016.

[BZ17] Massimo Bartoletti and Roberto Zunino. Constant-deposit multiparty lotteries on bitcoin. In *Financial Cryptography and Data Security*, 2017.

[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*.

[CGJ+17a] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *ACM CCS*, 2017.

[CGJ+17b] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 719–728, New York, NY, USA, 2017. Association for Computing Machinery.

[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.

[CS21] Hao Chung and Elaine Shi. Foundations of transaction fee mechanism design, November 2021. arXiv:2111.03151. URL: https://arxiv.org/pdf/2111.03151.pdf.

[DEF18] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *ACM CCS*, 2018.

[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.

[EFS20] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. Optiswap: Fast optimistic fair exchange. In *ASIA CCS*, 2020.

[Eth22] Ethereum. The Solidity contract-oriented programming language, 2022. URL: https://github.com/ethereum/solidity.

[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.

[GKM+22] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *PKC*, 2022.

[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+19] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gazi, Sarah Meiklejohn, and Edgar R Weippl. Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies. *IACR Cryptol. ePrint Arch.*, 2019:775, 2019.

[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.

[KB16] Ranjit Kumaresan and Iddo Bentov. Amortizing secure computation with penalties. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 418–429, New York, NY, USA, 2016. Association for Computing Machinery.

[KMS+16] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 839–858, 2016.

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

[MB17] Andrew Miller and Iddo Bentov. Zero-collateral lotteries in bitcoin and ethereum. In *EuroS&P Workshops*, 2017.

[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.

[MHM18a] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *Financial Cryptography Workshops*, 2018.

[MHM18b] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2018.

[Mic03] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, 2003.

[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.

[PG99] Henning Pagnia and Felix C. Gartner. On the impossibility of fair exchange without a trusted third party. Technical report, 1999.

[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.

[rin22] Rinkeby testnet, 2022. URL: https://www.rinkeby.io/.

[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 *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 357–366, 2019.

[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.