Atomic Cross-Chain Swaps

Maurice Herlihy
Computer Science Department
Brown University

January 30, 2018

Abstract

An *atomic cross-chain swap* is a distributed coordination task where multiple parties exchange assets across multiple blockchains, for example, trading bitcoin for ether.

An *atomic* swap protocol guarantees (1) if all parties conform to the protocol, then all swaps take place, (2) if some coalition deviates from the protocol, then no conforming party ends up worse off, and (3) no coalition has an incentive to deviate from the protocol.

A cross-chain swap has an associated directed graph $\mathcal{D} = (V, A)$. For any pair $(\mathcal{D}, L)$, where $\mathcal{D} = (V, A)$ is a strongly-connected directed graph and $L \subset V$ a feedback vertex set for $\mathcal{D}$, we give an atomic cross-chain swap protocol using hashed timelock contracts, where the vertexes in L generate the hashlocked secrets. We show that no such protocol is possible if $\mathcal{D}$ is not strongly connected, or if $\mathcal{D}$ is strongly connected but L is not a feedback vertex set. The protocol has time complexity $O(\mathit{diam}(\mathcal{D}))$ and communication complexity (bits published on blockchains) $O(|A| \cdot |L|)$.

1 Motivation

Carol wants to sell her Cadillac for bitcoins. Alice is willing to buy Carol's Cadillac, but she wants to pay in an "alt-coin" cryptocurrency. Fortunately, Bob is willing to trade alt-coins for bitcoins. Alice, Bob, and Carol need to arrange a three-way swap: Alice will transfer her alt-coins to Bob, Bob will transfer his bitcoins to Carol, and Carol will transfer title of her Cadillac to Alice[1]. Of course, no one trusts anyone else. How can we devise a protocol that ensures that if all parties behave rationally, in his or her

[1] Naturally, they live in a state that records automobile titles in a blockchain.

own self-interest, then all assets are exchanged, but if some parties behave irrationally, then no rational party will end up worse off?

In many blockchains, assets are transferred under the control of so-called *smart contracts* (or just *contracts*), scripts published on the blockchain that establish and enforce conditions necessary to transfer an asset from one party to another. For example, let $H(\cdot)$ be a cryptographic hash function. Alice might place her alt-coins in escrow by publishing on the alt-coin blockchain a smart contract with *hashlock* h and *timelock* t. Hashlock h means that if Bob sends the contract a value s, called a *secret*, such that $h = H(s)$, then the contract irrevocably transfers ownership of those alt-coins from Alice to Bob. Timelock t means that if Bob fails to produce that secret before time t elapses, then the escrowed alt-coins are refunded to Alice.

Here is a simple protocol for Alice, Bob, and Carol's three-way swap. Let Δ be enough time for one party to publish a smart contract on any of the blockchains, and for the other party to detect that the contract has been published.

- Alice creates a secret s, $h = H(s)$, and publishes a contract on the alt-coin blockchain with hashlock h and timelock 6Δ in the future, to transfer her alt-coins to Bob.

- When Bob confirms that Alice's contract has been published on the alt-coin blockchain, he publishes a contract on the Bitcoin blockchain with the same hashlock h but with timelock 5Δ in the future, to transfer his bitcoins to Carol.

- When Carol confirms that Bob's contract has been published on the Bitcoin blockchain, she publishes a contract on the automobile title blockchain with the same hashlock h, but with timeout 4Δ in the future, to transfer her Cadillac's title to Alice.

- When Alice confirms that Carol's contract has been published on the title blockchain, she sends s to Carol's contract, acquiring the title and revealing s to Carol.

- Carol then sends s to Bob's contract, acquiring the bitcoins and revealing s to Bob.

- Bob sends s to Alice's contract, acquiring the alt-coins and completing the swap.

What could go wrong? If any party halts during the first phase, when contracts are deployed, then all contracts eventually time out and trigger refunds. If any party halts during the second phase, when contracts are triggered, then only that party ends up worse off. For example, if Carol halts without triggering her contract, then Alice gets the Cadillac and Bob gets a refund, so Carol's misbehavior harms only herself.

The order in which contracts are deployed matters. If Carol were to post her contract with Alice before Bob posts his contract with Carol, then Alice could take ownership of the Cadillac without paying Carol.

Timelock values matter. If Carol's contract with Bob were to expire at the same time as Bob's contract with Alice, then Carol could reveal s to collect Bob's bitcoins at the very last moment, leaving Bob no time to collect his alt-coins from Alice.

What if parties behave irrationally? If Alice (irrationally) reveals s before the first phase completes, then Bob can take Alice's alt-coins, and perhaps Carol can take Bob's bitcoins, but Alice will not get her Cadillac, so only she is worse off.

A *atomic swap protocol* guarantees (1) if all parties conform to the protocol, then all swaps take place, (2) if some parties deviate from the protocol, then no conforming party ends up worse off[2], and (3) no coalition has an incentive to deviate from the protocol. Alice, Bob, and Carol's swapping adventure suggests broader questions: when are atomic cross-chain swaps possible, how can we implement them, and what do they cost?

While swapping digital assets is the immediate motivation for this study, atomic cross-chain swap protocols have other possible applications. *Sharding* [21] splits one blockchain into many for better load-balancing and scalability. Most of the time, activities on different shards proceed independently. When they cannot, an atomic swap protocol can coordinate needed cross-chain updates. In a decentralized distributed system, *upgrades* from one software version to another, or from one data schema to another, could benefit from atomic cross-chain swaps. In general, an atomic swap protocol is a trust-free, Byzantine-hardened form of *distributed commitment.*

Cross-chain swaps are well-known to the blockchain community [4, 6, 7, 16, 17, 22], but to our knowledge, this is the first systematic analysis of the theory underlying such protocols. We make the following contributions. A cross-chain swap has an associated directed graph (digraph) $\mathcal{D} = (V, A)$, whose vertexes are parties and arcs are proposed asset transfers. For any pair $(\mathcal{D}, L)$, where $\mathcal{D} = (V, A)$ is a *strongly-connected* directed graph and $L \subset V$ a

[2] Other than the inconvenience of having assets temporarily locked up.

feedback vertex set for $\mathcal{D}$, we give an atomic cross-chain swap protocol using hashed timelock contracts, where the vertexes in L, called *leaders*, generate the hashlocked secrets. We also show that no such protocol is possible if $\mathcal{D}$ is not strongly connected, or if $\mathcal{D}$ is strongly connected but the set of leaders L is not a feedback vertex set. The protocol has time complexity $O(\mathit{diam}(\mathcal{D}))$ and communication complexity (bits published on blockchains) $O(|A| \cdot |L|)$.

2 Model

2.1 Digraphs

A *directed graph* (or *digraph*) $\mathcal{D}$ is a pair (V, A), where V is a finite set of *vertexes*, and A is a finite set of ordered pairs of distinct vertexes called *arcs*. We use $V(\mathcal{D})$ for $\mathcal{D}$'s set of vertexes, and $A(\mathcal{D})$ for its set of arcs. An arc (u, v) has *end-vertexes* u and v, where u is called the arc's *head* and v its *tail.* An arc *leaves* its head and *enters* its tail. Similarly, an arc *enters* or *leaves* a set of vertexes $W \subseteq V$ if it enters or leaves a vertex of W.

A digraph $\mathcal{C}$ is a *subdigraph* of $\mathcal{D}$ if $V(\mathcal{C}) \subseteq V(\mathcal{D}), A(\mathcal{C}) \subseteq A(\mathcal{D})$ and every arc in $A(\mathcal{C})$ has both its head and tail in $V(\mathcal{C})$. A subdigraph $\mathcal{C}$ of $\mathcal{D}$ is *strict* if $V(\mathcal{C}) \subset V(\mathcal{D})$. If every arc of $\mathcal{D}$ with both end-vertexes in $\mathcal{C}$ is an arc of $\mathcal{C}$, then $\mathcal{C}$ is *induced.*

A *path* of length ℓ in $\mathcal{D}$ is a sequence of arcs $(a_0, \ldots, a_\ell)$, for $\ell > 0$, where for each $i \in 0, \ldots, \ell - 1$, a_i's tail is a_{i+1}'s head, and the arcs' heads are distinct. A path is a *cycle* if a_ℓ's tail is a_0's head. A digraph is *acyclic* if it has no cycles. Vertex v is *reachable* from vertex u if there is a path from u to v. $\mathcal{D}$'s *diameter* $\mathit{diam}(\mathcal{D})$ the length of the longest path from any vertex to any other. $\mathcal{D}$ is *connected* if its underlying graph is connected, and *strongly connected* (or *strong*) if, for every pair u, v of distinct vertexes in $\mathcal{D}$, u is reachable from v and v is reachable from u. A *feedback vertex set* is a subset of V whose deletion leaves $\mathcal{D}$ acyclic.

The *transpose* $\mathcal{D}^T$ is the digraph obtained from $\mathcal{D}$ by reversing all arcs. If $\mathcal{D}$ is strong, so is $\mathcal{D}^T$, and any feedback vertex set for $\mathcal{D}$ is also a feedback vertex set for $\mathcal{D}^T$.

2.2 Blockchains and Smart Contracts

For our purposes, a *blockchain* is a distributed service that allows clients to publish transactions to a publicly-readable, tamper-proof distributed ledger. Our analysis is independent of the particular blockchain algorithm. We assume a semi-synchronous model with a known duration Δ long enough for one party to publish a contract to a blockchain, and for a second party to confirm that the contract has been published.

The owner of an asset (such as a unit of cryptocurrency or an automobile title) can create a smart contract to transfer ownership of that asset to a *counterparty* if specified conditions are met. A contract is *published* when its creator places it on a blockchain ledger. Once a contract is published, it is irrevocable: neither the contract's creator nor any other party can remove the contract nor tamper with its terms.

Although many contract programming languages are Turing-complete, we restrict our attention to a specialized (but powerful) form of contract called *hashed timelock* contracts [5].

A *hashlock* h prevents a contract from transferring an asset until the contract receives a matching *secret*, s, where $h = H(s)$, for H an agreed-upon cryptographic hash function. A *timelock* t prevents a contract from transferring an asset until a specified future time t. A *hashed timelock contract* contract works as follows. A party publishes a contract that assumes temporary control of an asset's ownership. The contract is given values (h, t), such that if the contract receives the matching secret s, $h = H(s)$, before time t has elapsed, then the contract is *triggered*, irrevocably transferring ownership of the asset to the counterparty. If the contract does not receive the matching secret before time t has elapsed, then the asset is *refunded* to the original owner.

For arbitrary swaps, we will need *vectors* of locks: a hashlock vector $(h_0, \ldots, h_\ell)$, and a timelock vector $(t_0, \ldots, t_\ell)$, A contract is triggered if it receives each secret s_i, $h_i = H(s_i)$ before timelock t_i has elapsed. If any t_i elapses before receiving s_i, that contract is refunded.

A *rational* party acts in its own self-interest, deviating from a protocol only if it is profitable to do so. Rational parties can collude with one another to disadvantage other parties. An *irrational* party may deviate from a protocol even if it is not profitable to do so. Parties may behave irrationally out of spite, because they were hacked, or because they profit in ways not foreseen by the protocol designers.

3 Swap Digraphs and Games

A cross-chain swap is given by a digraph $\mathcal{D} = (V, A)$, where each vertex in V represents a party, and each arc in A represents a proposed asset transfer from the arc's head to its tail via a shared blockchain. (We assume without loss of generality that $\mathcal{D}$ is connected, because a disconnected digraph can be treated as multiple swaps.) Henceforth, we use *party* and *vertex*, *blockchain* and *arc*, interchangeably, depending on whether we emphasize roles or digraph structure.

In the terminology of game theory, a swap $\mathcal{D}$ is a *cooperative game*, each of whose outcomes is given by a subdigraph $\mathcal{E} = (V, A')$ of $\mathcal{D}$. If a proposed transfer $(u, v) \in A$ is also in $(u, v) \in A'$, then that transfer happened. For short, we say arc (u, v) was *triggered*.

A protocol is a *strategy* for playing a game: a set of rules that determines which step a party takes at any stage of a game. To model real-world situations where multiple parties are secretly controlled by a single adversary, the swap game is *cooperative*: parties can form *coalitions* where coalition members commit to a common strategy.

Here are the payoffs for a party v, organized into classes, and ranked in preference order, most preferable first. For brevity, each class has a shorthand name.

- The party acquires assets without paying: at least one arc entering v is triggered, but no arc leaving v is triggered (FreeRide).
- The party acquires assets while paying less than expected: all arcs entering v are triggered, but at least one arc leaving v is not triggered (Discount).
- The party swaps assets as expected: All arcs entering and leaving v are triggered (Deal).
- No assets change hands: no arc entering or leaving v is triggered (NoDeal).
- The party pays without acquiring all expected assets: at least one arc entering v is not triggered, and at least one arc leaving v is triggered (Underwater).

Payoffs for a coalition $C \subset V$ are defined by replacing v with C in the definitions above.

Definition 1. *A swap protocol* $\mathbb{P}$ *is* uniform *if it satisfies:*

- *If all parties follow* $\mathbb{P}$*, they all finish with payoff* Deal*.*
- *If any coalition cooperatively deviates from* $\mathbb{P}$*, no non-deviating party finishes with payoff* Underwater*.*

A uniform protocol is not useful if rational parties will not follow it. A swap protocol is a *strong Nash equilibrium strategy* if no coalition improves its payoff when its members cooperatively deviate from that protocol.

Definition 2. *A swap protocol $\mathbb{P}$ is* atomic *if it is both uniform and a strong Nash equilibrium strategy.*

This definition formalizes the notion that if all parties are rational, all swaps happen, but if some parties are irrational, the rational parties will never end up worse off. We say that a *compliant* party follows the protocol, while a *deviating* party does not.

Lemma 1. *If $\mathcal{D}$ is strongly connected, then any uniform swap protocol $\mathbb{P}$ is atomic.*

Proof. If a deviating coalition $C \subset V$ achieves a better payoff than Deal, then that payoff is either FreeRide or Discount. It follows that some arc that enters C is triggered, and some arc that leaves C is untriggered. Moreover, if any arc that enters C is untriggered, then all arcs that leave C are untriggered.

A conforming party $v \notin C$ cannot end up Underwater, so if an arc entering v is untriggered, then every arc leaving v must be untriggered, and if an arc leaving v is triggered, then every arc entering v must be triggered.

Let (c, v) be an untriggered arc leaving C. Since v is conforming, every arc leaving v is untriggered. Because $\mathcal{D}$ is strongly connected, there is a path $(v, v_0), (v_1, v_2), \ldots, (v_k, c_0)$ where each $v_i \notin C$, and $c_0 \in C$. By a simple inductive argument, each arc in this path is untriggered, so the arc (v_k, c_0) that enters C is untriggered, so every arc leaving C must be untriggered.

Let (v, c) be a triggered arc entering C. Since v is conforming, every arc entering v is triggered. Because $\mathcal{D}$ is strongly connected, there is a path $(c_1, v_0), (v_0, v_1), \ldots, (v_k, c)$ where each $v_i \notin C$, and $c_1 \in C$. By a simple inductive argument, each arc in this path is triggered, so the arc (c_1, v_0) that leaves C is triggered, contradicting the claim that every arc leaving C is untriggered. □

Lemma 2. *If $\mathcal{D}$ is not strongly connected, then no uniform swap protocol is atomic.*

Proof. Because $\mathcal{D}$ is not strongly connected, it contains vertexes x, y such that y is reachable from x, but not vice-versa. Let Y be the set of vertexes reachable from y, and X the rest: $X = V \setminus Y$. X is non-empty because it contains x. Because y is reachable from x, there is at least one arc from X to Y, but no arcs from Y to X.

Coalition X can improve its payoff by triggering all arcs between vertexes in X, but no arcs from X to Y, yielding payoff FreeRide for X, since it triggers strictly fewer arcs leaving X, without affecting any arcs entering X. In fact, the payoff for each individual vertex in X is either the same or better than Deal. □

We have just proved:

Theorem 3. *A uniform swap protocol for $\mathcal{D}$ is atomic if and only if $\mathcal{D}$ is strongly connected.*

Informally, if $\mathcal{D}$ is not strongly connected, then rational parties will deviate from any uniform protocol. In practice, such a swap would never be proposed, because the parties in X would never agree to a swap with the free riders in Y. Henceforth, $\mathcal{D}$ is assumed strongly connected.

4 An Atomic Swap Protocol

For simplicity, assume the swap digraph is constructed by a (possibly centralized) market-clearing service. (The clearing service is not a trusted party, because the parties can check the consistency of the setup information.) Each participant v_i creates a secret s_i and matching hashlock $h_i = H(s_i)$. It sends the clearing service its hashlock, along with an offer characterizing the swaps it is willing to make. The service combines these offers to construct a swap digraph $\mathcal{D} = (V, A)$, a vector $L \subset V$ of *leaders* forming a feedback vertex set, a vector of those leaders' hashlocks $h_0, \ldots, h_\ell$, and a *global deadline* T, which must be at least $2 \cdot \mathit{diam}(\mathcal{D}) \cdot \Delta$ after the protocol's starting time.

If all parties follow the protocol, all contracts with be triggered at or before the deadline T, but if some parties deviate, the conforming parties' assets will be refunded by then.

4.1 Contracts

Figure 1 shows pseudocode[3] for a hashed timelock contract. A smart contract resembles an object in an object-oriented programming language, pro-

[3] This pseudocode is based loosely on the popular Solidity programming language for smart contracts [20].

```
contract Swap {
  Asset asset;                 /* asset to be transferred or refunded */
  address party;               /* transfer asset from */
  address counterparty;        /* transfer asset to */
  uint[] timelock;             /* vector of timelocks */
  uint[] hashlock;             /* vector of hashlocks */
  bool[] revealed;             /* which secrets revealed? */
  /* constructor */
  function Swap (Asset _asset; /* asset to be transferred or refunded */
                 address _party;        /* transfer asset from */
                 address _counterparty; /* transfer asset to */
                 uint[] _timelock;      /* vector of timelocks */
                 uint[] _hashlock;      /* vector of hashlocks */
                 ) {
    asset = _asset;                                    /* copy */
    party = _party; _counterparty = _counterparty;     /* copy */
    timelock = _timelock;                              /* copy */
    hashlock = _hashlock;                              /* copy */
    revealed = [false, ..., false];                    /* no secrets yet */
  }
  function revealSecret (int i, uint s) {
    if (timelock[i] < now) { /* timelock expired? */
      refund asset to party;
      halt;
    } else {
      if (hashLock[i] == H(s)) { /* Is secret correct? */
        revealed[i] = true;
        if (all secrets revealed) {
          transfer asset to counterparty;
          halt;
        }}}}
}
```

Figure 1: Pseudocode for hashed timelock contract

viding *long-lived state* (Lines 2-7), a *constructor* (Lines 9-20) to initialize that state, and one or more *functions* (Lines 21-31) to manage that state.

The contract's long-lived state records the asset to be transferred or refunded (Line 2), the party transferring the asset (Line 3), the counterparty receiving the asset (Line 4), a vector of timelocks (Line 5), a vector of hashlocks (Line 6), and a Boolean vector marking which secrets have been revealed (Line 7).

When the contract is initialized, its constructor copies the fields provided into the contract's long-lived state (Lines 15-18).

This contract provides a revealSecret () function that takes an index and a secret. If the matching timelock duration has elapsed, the asset is refunded to the original owner, and the contract halts (Lines 22-24). Otherwise, if the proffered secret matches the hashlock, that secret is marked as revealed (Lines 26-27). When all secrets have been revealed, the asset is transferred to the counterparty and the contract halts (Lines 28-30).

4.2 Timelocks

When a Swap contract is initialized, it is provided a timelock vector that depends on the position of that arc (v, w) in the swap digraph. For leaders $L = \{v_0, \ldots, v_\ell\}$, let $d_i(v)$ be the length of the longest path from v to v_i. The timelock vector $(t_0, \ldots, t_\ell)$ for the contract on arc (v, w) has i^{th} component $(T - d_i(v) \cdot \Delta)$, where T is the swap's global deadline.

4.3 Pebble Games

We analyze the protocol using two variations on a simple pebble game. We are given a strong digraph $\mathcal{D} = (V, A)$, and a vertex feedback set $L \subset V$ of *leaders*.

In both games, start by placing pebbles on the arcs leaving each leader. In the *lazy* variation of the game, place new pebbles on the arcs leaving vertex v when there is a pebble on *every* arc entering v. In the *eager* variation, place new pebbles on the arcs leaving v when there is a pebble on *any* arc entering v. The game continues until no more pebbles can be placed.

Lemma 3. *In both variations, every arc in $\mathcal{D}$ eventually has a pebble.*

Proof. Suppose by way of contradiction, the game stops in a state where an arc (u, v) has no pebble. There must be a pebble-free arc (u', u) entering u, because otherwise the game would have placed a pebble on (u, v). Continuing in this way, build a longer and longer pebble-free path until it becomes a pebble-free cycle. But leaders form a feedback vertex set, so every cycle in $\mathcal{D}$ includes a leader, and the arcs leaving that leader have pebbles placed in the first step. □

Lemma 4. *In both variations, if there is a worst-case delay Δ between when the last pebble is placed on any arc entering v, and when the last pebble is placed on any arc leaving v, then every arc will have a pebble in time at most $\mathit{diam}(\mathcal{D}) \cdot \Delta$ from when the game started.*

Proof. It is enough to show that in each interval of time Δ, the longest pebble-free path shrinks by one. At any time after the first step, let $a_0, \ldots, a_k$ be a pebble-free path of maximal length. That path cannot be a cycle, because then it would include a leader, who would have placed a pebble on a_0 in the first step. It follows that every arc entering the head of a_0 must have a pebble, because otherwise we could construct a longer pebble-free path. By hypothesis, within time Δ, a_0 will have a pebble, and the path will have shrunk by one. □

Corollary 4. *Under the stated timing assumptions, every arc has a pebble within time $\mathit{diam}(\mathcal{D}) \cdot \Delta$.*

4.4 The Protocol

There are two phases. In Phase One, instances of the Swap contract (Figure 1) are propagated through the swap digraph, starting at the leaders. Timelocks are initialized as in Section 4.2, and hashlocks as distributed by the clearing service, with a global deadline at least $2 \cdot \mathit{diam}(\mathcal{D}) \cdot \Delta$ in the future. Each time a party reads a contract on an entering arc, it verifies that contract is correct, and abandons the protocol otherwise.

In Phase One, leaders follow this protocol:

1. Publish a contract on every arc leaving the leader.
2. Wait until contracts have been published on all arcs entering the leader.

Non-leaders follow this protocol:

1. Wait until contracts have been published on all arcs entering the vertex.
2. Publish a contract on every arc leaving the vertex.

In Phase Two, the parties disseminate secrets. While contracts propagate in the direction of the arcs, from party to counterparty, secrets propagate in the opposite direction, from counterparty to party. Each party is motivated to trigger its entering arcs to acquire those arcs' assets.

At the start of Phase Two, each leader v_i sends its secret s_i to the contract on each entering arc (via $\mathsf{revealSecret}\,(i, s_i)$).

For every party, the first time a secret s_i appears on an arc leaving that vertex, that party sends s_i to the contract on each arc entering that vertex. Phase Two ends when the contracts on all arcs entering and leaving all vertexes have either been triggered or have timed out.

Theorem 5. *If all parties follow the protocol, then every contract is triggered within time $2 \cdot \mathit{diam}(\mathcal{D}) \cdot \Delta$ of when the protocol started.*

Proof. Phase One, contract dissemination, is an instance of the lazy pebble game on $\mathcal{D}$, so every arc in $\mathcal{D}$ has a published contract within time $\mathit{diam}(\mathcal{D}) \cdot \Delta$ of when the protocol started.

In Phase Two, the dissemination of each secret s_i is an instance of the eager pebble game on $\mathcal{D}^T$, the transpose graph, so every secret is delivered to the contract on every arc of $\mathcal{D}$ within time $2 \cdot \mathit{diam}(\mathcal{D}) \cdot \Delta$ of when the protocol started. □

The deadline $2 \cdot \mathit{diam}(\mathcal{D}) \cdot \Delta$ bounds the time assets can be held in escrow when things go wrong. In practice, one would expect actual running times to be shorter.

Theorem 6. *No compliant party ends up* Underwater.

Proof. Suppose v is compliant, the arc (v, w) is triggered, but the arc (u, v) is not.

First, (u, v) must have a contract. If v is a leader with secret s_i, then v will not release s_i until every arc entering v has a contract. If v is not a leader, then it will not publish a contract on an arc leaving v until contracts have been published on every arc entering v.

Suppose the contract on (u, v) failed to trigger because it never received s_i. If v is the leader who generated s_i, v would have sent s_i to (u, v) at the start of Phase Two. If v is not that leader, then let t_i be the timelock for s_i on (v, w), and t'_i the timelock on (u, v). Because v did not generate s_i, $t'_i \geq t_i - \Delta$. Secret s_i was delivered to (v, w) before t_i, leaving sufficient time, at least Δ, for v to propagate s_i to (u, v) before t'_i. □

Theorem 7. *For $\mathcal{D} = (V, A)$ with leaders $L \subset V$, the communication complexity, measured as the number of bits published on blockchains, is $O(|A| \cdot |L|)$.*

Proof. Phase One has communication complexity $|A| \cdot |L|$, since a contract containing vectors of size L is published on each arc. Phase Two has communication complexity $|A| \cdot |L|$, since $|L|$ secrets are published. □

The asymptotic complexity is unchanged if we measure blockchain transactions instead of bits.

Finally, any atomic cross-chain swap protocol using hashed timelock contracts cannot look very different from the one we presented.

Lemma 5. *In any uniform hashed timelock swap protocol, no non-leader v can publish a contract on an arc leaving v before contracts have been published on all arcs entering v.*

Proof. If v has has a contract on arc (v, w) but no contract on arc (u, v), then the leaders could irrationally broadcast all secrets, triggering the contract on (v, w), and u could refuse to publish a contract on (u, v), leaving v with payoff UNDERWATER. □

Theorem 8. *In any uniform swap protocol based on hashed timelock contracts, the set L of leaders is a feedback vertex set in $\mathcal{D}$.*

Proof. Suppose, instead, there is a uniform swap protocol where the leaders do not form a vertex feedback set.

At any step in the protocol, the *waits-for* digraph W is the subdigraph of $\mathcal{D}^T$ where (v, u) is an arc of W if (u, v) has no published contract. Informally, Lemma 5 implies that v must be waiting for u to publish a contract on (u, v) before u can publish any contracts on its own outgoing arcs. In the initial state, if $\mathcal{D} \setminus L$ contains a cycle, so does W. At each protocol step, a non-leader v can publish a contract on a leaving arc only if v has indegree zero in the current waits-for graph. But no vertex on a cycle in the waits-for digraph will ever have indegree zero, a contradiction. □

5 Remarks

Finding a minimal feedback vertex set for $\mathcal{D}$ is NP-complete [13], although there exists an efficient 2-approximation [3].

The protocol is easily extended to a model where there may be more than one arc from one vertex to another, so-called *directed multi-graphs* [2], where Alice wants to transfer assets on distinct blockchains to Bob.

The swap protocol is still vulnerable to a weak denial-of-service attack where an adversarial party repeatedly proposes an attractive swap, and then fails to complete the protocol, triggering refunds, but temporarily rendering assets inaccessible. One could require parties to post bonds, and following a failed swap. one could examine the blockchains to determine who was at fault (by failing to execute an enabled transition).

6 Related Work

The use of hashed timelock contracts for two-party cross-chain swaps is believed to have emerged from an on-line discussion forum in 2016 [4, 16]. There is open-source code [6, 7, 17] for two-party cross-chain swap protocols between selected currencies, and applications using swaps include the Enigma Catalyst proposal [22].

Off-chain payment networks [15, 10, 18] circumvent the scalability limits of existing blockchains by conducting multiple transactions off the blockchain, eventually resolving final balances through a single on-chain transaction. These algorithms also use hashed timelock contracts, but they address a different set of problems.

Multi-party swaps arise when matching kidney donors and recipients. A transplant recipient with an incompatible donor can swap donors to ensure that each recipient obtains a compatible organ. A number of algorithms [1, 8, 11] have been proposed for matching donors and recipients. Shapley and Scarf [19] consider the circumstances under which certain kinds of swap markets have strong equilibriums. Kaplan [12] describes a polynomial-time algorithm that given a set of proposed swaps, constructs a swap digraph if one exists. These papers and many others focus on “the clearing problem”, roughly analogous to constructing a swap digraph, but not on how to execute those swaps on blockchains.

The *fair exchange* problem [9, 14] is a precursor to the atomic cross-chain swap problem. Alice has a digital asset Bob wants, and vice-versa, and at the end of the protocol, either Alice and Bob have exchanged assets, or they both keep their assets. In the absence of blockchains, trusted, or semi-trusted third parties are required, but roles of those trusted parties can be minimized in clever ways.

References

[1] D. J. Abraham, A. Blum, and T. Sandholm. Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM Conference on Electronic Commerce*, EC '07, pages 295–304, New York, NY, USA, 2007. ACM.

[2] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms, and Applications*. Monographs in Mathematics. Springer, 2001.

[3] A. Becker and D. Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1):167 – 188, 1996.

[4] bitcoinwiki. Atomic cross-chain trading. https://en.bitcoin.it/wiki/Atomic_cross-chain_trading. Accessed: 9 January 2018.

[5] bitcoinwiki. Hashed timelock contracts. https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts. Accessed: 8 January 2018.

[6] S. Bowe and D. Hopwood. Hashed time-locked contract transactions. https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki. Accessed: 9 January 2018.

[7] DeCred. Decred cross-chain atomic swapping. https://github.com/decred/atomicswap. Accessed: 8 January 2018.

[8] J. P. Dickerson, D. F. Manlove, B. Plaut, T. Sandholm, and J. Trimble. Position-indexed formulations for kidney exchange. *CoRR*, abs/1606.01623, 2016.

[9] M. K. Franklin and G. Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *Financial Cryptography*, 1998.

[10] M. Green and I. Miers. Bolt: Anonymous payment channels for decentralized currencies. Cryptology ePrint Archive, Report 2016/701, 2016. https://eprint.iacr.org/2016/701.

[11] Z. Jia, P. Tang, R. Wang, and H. Zhang. Efficient near-optimal algorithms for barter exchange. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 362–370, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.

[12] R. M. Kaplan. An improved algorithm for multi-way trading for exchange and barter. *Electronic Commerce Research and Applications*, 10(1):67 – 74, 2011. Special Section: Service Innovation in E-Commerce.

[13] R. M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, pages 85–103, 1972.

[14] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, PODC '03, pages 12–19, New York, NY, USA, 2003. ACM.

[15] R. Network. What is the raiden network? `https://raiden.network/101.html`. Accessed: 26 January 2018.

[16] T. Nolan. Atomic swaps using cut and choose. `https://bitcointalk.org/index.php?topic=1364951`. Accessed: 9 January 2018.

[17] T. K. Organization. The barterdex whitepaper: A decentralized, open-source cryptocurrency exchange, powered by atomic-swap technology. `https://supernet.org/en/technology/whitepapers/BarterDEX-Whitepaper-v0.4.pdf`. Accessed: 9 January 2018.

[18] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`, Jan. 2016. Accessed: 29 December 2017.

[19] L. Shapley and H. Scarf. On cores and indivisibility. *Journal of Mathematical Economics*, 1(1):23–37, 1974.

[20] Solidity documentation. `http://solidity.readthedocs.io/en/latest/index.html`.

[21] A. Zamyatin. On sharding blockchains. https://github.com/ethereum/wiki/wiki/Sharding-FAQ. Accessed: 8 January 2018.

[22] G. Zyskind, C. Kisagun, and C. FromKnecht. Enigma catalyst: a machine-based investing platform and infrastructure for crypto-assets. https://www.enigma.co/enigma_catalyst.pdf. Accessed: 25 January 2018.