Atomic Cross-Chain Swaps

Maurice Herlihy
Computer Science Department
Brown University
Providence, Rhode Island 02912
maurice.herlihy@gmail.com

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 is modeled as a directed graph $\mathcal{D}$, whose vertexes are parties and whose 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 feedback vertex set for $\mathcal{D}$, we give an atomic cross-chain swap protocol for $\mathcal{D}$, using a form of 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 $O(diam(\mathcal{D}))$ and space complexity (bits stored on all blockchains) $O(|A|^2)$.

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 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, illustrated in Figures 1 and 2. Let Δ be enough time for one party to publish a smart contract on any of the blockchains, or to change a contract's state, and for the other party to detect the change.

- 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 while contracts are being deployed, then all contracts eventually time out and trigger refunds. If any party halts while contracts are being 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.

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

PODC'18, Egham, United Kingdom
 978-1-4503-5795-1/18/07…$15.00
DOI: 10.1145/3212734.3212736

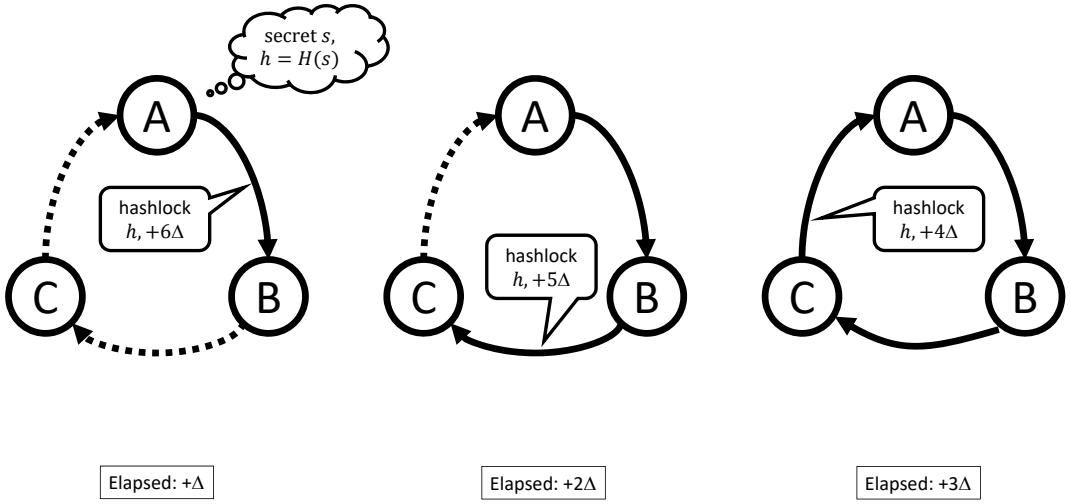

Figure 1: Atomic cross-chain swap: deploying contracts

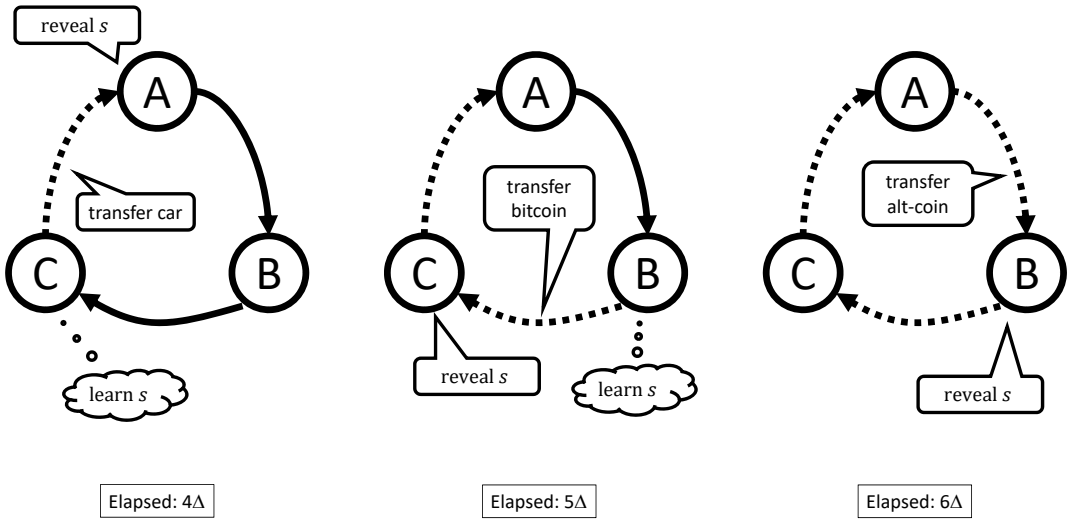

Figure 2: Atomic cross-chain swap: triggering arcs

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* [7] 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. An atomic swap protocol can be thought of as a trust-free, Byzantine-hardened form of *distributed commitment* [26]. An atomic cross-chain swap is a special case of a *distributed atomic transaction* [25], although not all atomic transactions can be expressed as cross-chain swaps.

Cross-chain swaps are well-known to the blockchain community [4, 6, 9, 20, 21, 27], 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 is modeled as a directed graph (digraph) $\mathcal{D}$, whose vertexes are parties and whose arcs are proposed asset transfers. For any pair $(\mathcal{D}, L)$, where $\mathcal{D} = (V, A)$ is a *strongly-connected* digraph and $L \subset V$ a *feedback*

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

vertex set for $\mathcal{D}$, we give an atomic cross-chain swap protocol using a form of hashed timelock contracts, where the vertexes in L, called *leaders*, generate the hashlocked secrets. (Vertexes that are not leaders are *followers*.) 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 $O(diam(\mathcal{D}))$ 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 *head* u and *tail* v. An arc *leaves* its head and *enters* its tail. An arc (u, v) *enters* a set of vertexes $W \subseteq V$ if $u \notin W$ and $v \in W$, and similarly for leaving.

A digraph C is a *subdigraph* of $\mathcal{D}$ if $V(C) \subseteq V(\mathcal{D}), A(C) \subseteq A(\mathcal{D})$ and every arc in $A(C)$ has both its head and tail in $V(C)$.

A *path* p in $\mathcal{D}$ is a sequence of vertexes $(u_0, \ldots, u_\ell)$ such that $u_0, \ldots, u_{\ell-1}$ are distinct. Path p has *length* ℓ, denoted by $|p|$. If v is a vertex, and $(u_0, \ldots, u_\ell)$ a path that does not include the arc (v, u_0), then $v + p$ denotes the path $(v, u_0, \ldots, u_\ell)$. For vertexes u, v, $D(u, v)$ is the length of the longest path from u to v in $\mathcal{D}$.

A path $(u_0, \ldots, u_\ell)$ is a *cycle* if $u_0 = u_\ell$. 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* $diam(\mathcal{D})$ is 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* 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 strongly connected, 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 timing model where there is 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.

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.

Blockchain protocols typically require parties to have public and private keys. We use $sig(x, v)$ to denote the result of v using its private key to sign x.

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*, organized so that if all parties follow the protocol, each transfer on each arc happens. Each possible outcome 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.

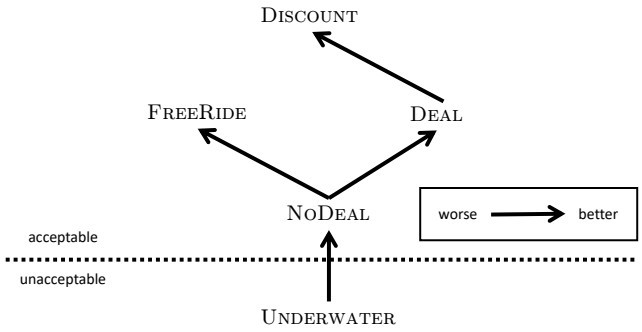

Figure 3: Partial order of protocol outcomes

Here are the outcomes for a party v, organized into classes, 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.

The protocol design incorporates certain conservative assumptions about parties' preferences. The protocol's preferred outcome is for all conforming parties to end with outcome Deal. In the presence of failures or deviation, however, it is acceptable for conforming parties to end with outcome NoDeal, the *status quo*, rendering them no worse off. Furthermore, each party is assumed to prefer Deal to NoDeal, because otherwise it would not have agreed to the swap in the first place. It follows that each party prefers any FreeRide outcome to NoDeal, because it acquires additional assets "for free", without relinquishing any assets of its own. Similarly, each party prefers any Discount outcome to Deal, since that party acquires the same assets in both outcomes, but relinquishes fewer in Discount outcomes. For these reasons, Deal, NoDeal, Discount, and FreeRide are all considered acceptable outcomes for conforming parties if the protocol execution is unable to complete because of failures or adversarial behavior.

We consider the remaining Underwater outcomes to be unacceptable to conforming parties. It is possible that in some idiosyncratic cases, a party may actually prefer particular Underwater outcomes to NoDeal. For example, a party with three entering arcs and one leaving arc may be be willing to relinquish its asset in return for acquiring only two out of three of the entering arcs' assets. We leave the design of protocols that make such fine distinctions to future work.

Definition 3.1. 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 conforming 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 3.2. 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. Recall that a *conforming* party follows the protocol, while a *deviating* party does not.

Lemma 3.3. *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 in 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, and some arc entering C must be triggered.

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) leaving C is triggered, contradicting the fact that every arc leaving C is untriggered. □

Lemma 3.4. *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 3.5. *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.

We remark that Theorem 3.5 relies on the implicit technical assumption that all value transfers are explicitly represented on some blockchain. This theorem would be falsified, for example, if Carol responds to learning Alice's secret by sending a large drone to drop her Cadillac in the middle of Alice's driveway, without ever recording that transfer in a shared blockchain. We will assume that if swaps have off-chain consequences, as they typically do, that those consequences are explicitly recorded in the form of blockchain updates.

4 AN ATOMIC SWAP PROTOCOL

4.1 Hashlocks and Hashkeys

In a simple two-party swap, each party publishes a contract that assumes temporary control of that party's asset. This *hashed timelock contract* [5] stores a pair (h, t), and ensures 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 multi-party cross-chain swaps, we will need to extend these notions in several ways.

```
contract Swap {
  Asset asset;                  /* asset to be transferred or refunded */
  Digraph digraph;              /* swap digraph */
  address[] leaders;            /* leaders */
  address party;                /* transfer asset from */
  address counterparty;         /* transfer asset to */
  uint[] timelock;              /* vector of timelocks */
  uint[] hashlock;              /* vector of hashlocks */
  bool[] unlocked;              /* which hashlocks unlocked? */
  uint start;                   /* protocol starting time */
  /* constructor */
  function Swap (Asset _asset; /* asset to be transferred or refunded */
                 Digraph _digraph;     /* swap digraph */
                 address[] _leaders;   /* leaders */
                 address _party;       /* transfer asset from */
                 address _counterparty; /* transfer asset to */
                 uint[] _timelock;     /* vector of timelocks */
                 uint[] _hashlock;     /* vector of hashlocks */
                 uint _start           /* protocol starting time */
                 ) {
    asset = _asset;                                  /* copy */
    party = _party; counterparty = _counterparty;    /* copy */
    timelock = _timelock; hashlock = _hashlock;      /* copy */
    unlocked = [false, ..., false];                  /* all unlocked */
  }
```

Figure 4: Swap contract (part one)

In the three-way swap recounted earlier, each arc had a single hashlock and a single timeout. Timeouts were assigned so that the timeout on each arc entering a follower v was later by at least Δ than the timeout on each arc leaving v. This gap ensures that if any arc leaving v is triggered, then v has time to trigger every entering arc.

If a swap digraph has only one leader, $\hat{v}$, then the subdigraph of its followers is acyclic. As in our three-way swap example, the hashlock on arc (u, v) can be given timeout $(diam(\mathcal{D}) + D(v, \hat{v}) + 1) \cdot \Delta$, where $D(v, \hat{v})$ is the length of the longest path from v to the unique leader $\hat{v}$. (See left-hand side of Figure 6.)

This formula does not work if a swap digraph has more than one leader, because the subdigraph of any leader's followers has a cycle, and it is not possible to assign timeouts across a cycle in a way that guarantees a gap of at least Δ between entering and leaving arcs. (See right-hand side of Figure 6.)

Instead, for general digraphs, we must replace timed hashlocks with a more general mechanism, one that assigns different timeouts to different paths. Pick a set $L = \{v_0, \ldots, v_\ell\}$ of vertexes, called *leaders*, forming a feedback vertex set for $\mathcal{D}$. Each leader v_i generates a secret s_i and hashlock value $h_i = H(s_i)$, yielding a *hashlock vector* $(h_0, \ldots, h_\ell)$, which is assigned to every arc.

A *hashkey* for h on arc (u, v) is a triple (s, p, σ), where s is the secret $h = H(s)$, p is a path $(u_0, \ldots, u_k)$ in $\mathcal{D}$ where $u_0 = v$ and u_k is the leader who generated s, and

$$\sigma = sig(\cdots sig(s, u_k), \ldots, u_0),$$

the result of having each party in the path sign s. A hashkey (s, p, σ) *times out* at time $(diam(\mathcal{D}) + |p|) \cdot \Delta$ after the start of the protocol. That hashkey *unlocks* h on (u, v) if it is presented before it times out. An arc is *triggered* when all of its hashlocks are unlocked. A hashlock has *timed out* on an arc when all of its hashkeys on that arc have timed out. Figure 7 shows partial hashkeys for a two-leader swap digraph.

4.2 Market Clearing

For simplicity, assume the swap digraph is constructed by a (possibly centralized) market-clearing service, which perhaps communicates with the parties through its own blockchain. The clearing service is not a trusted party, because the parties can check the consistency of the clearing service's responses.

Each party creates a secret s and matching hashlock $h = H(s)$. It sends the clearing service its hashlock, along with an offer characterizing the swaps it is willing to make. The service combines these offers, and publishes 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 *starting time* T, at least Δ in the future.

```
function unlock (int i, uint s, Path path, Sig sig) {
    require (msg.sender == counterparty); /* only from counterparty */
    if (now < start + (diam(digraph) + |path|) * Δ /* hashkey still valid? */
        && hashlock[i] == H(s)              /* secret correct? */
        && isPath(path, digraph, leader[i], counterparty) /* path valid? */
        && verifySigs(sig, s, path) { /* signatures valid? */
      unlocked[i] = true;
    }
  }
  function refund () {
    require (msg.sender == party); /* only from party */
    if (any hashlock unlocked and timed out) {
      transfer asset to party;
      halt;
    }
  }
  function claim () {
    require (msg.sender == counterparty); /* only from counterparty */
    if (every hashlock unlocked) {
      transfer asset to counterparty;
      halt;
    }
  }
}
```

Figure 5: Swap contract (part two)

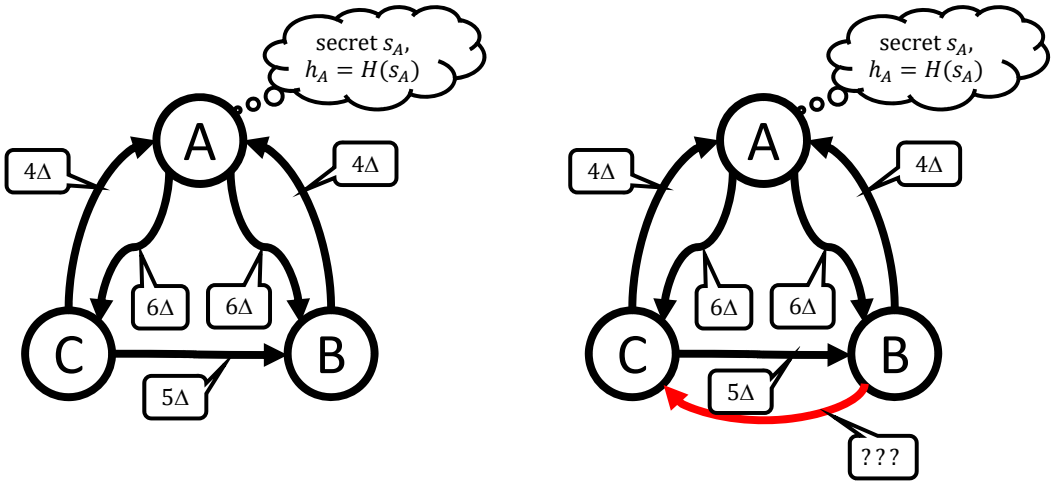

Figure 6: A is the leader, B and C followers. Timeouts can be assigned when the follower subdigraph is acyclic (left) but not when it is cyclic (right)

If all parties conform to the protocol, all contracts will be triggered before $T + 2 \cdot diam(\mathcal{D}) \cdot \Delta$, but if some parties deviate, the conforming parties' assets will be refunded by then.

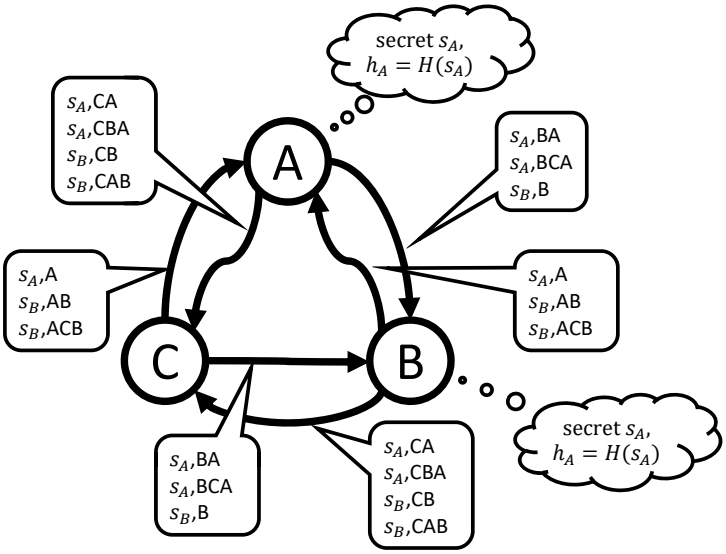

Figure 7: Hashkey paths for arcs of two-leader digraph

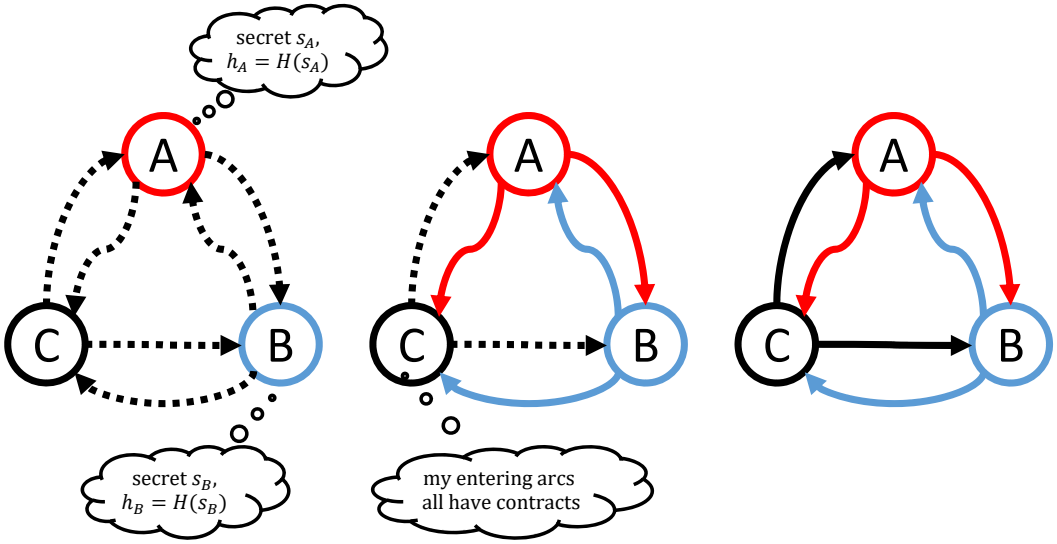

Figure 8: Concurrent contract propagation for two-leader digraph

4.3 Contracts

Figures 4 and 5 show pseudocode[3] for a hashed timelock swap contract. A smart contract resembles an object in an object-oriented programming language, providing *long-lived state* (Lines 2-9), a *constructor* to initialize that state (Lines 12-25), and one or more *functions* to manage that state (Lines 26-48).

The contract's long-lived state records the asset to be transferred or refunded (Line 2), the digraph $\mathcal{D}$ (Line 3), the digraph's set of leaders (Line 4), the party transferring the asset (Line 5), the counterparty receiving the asset (Line 6), a vector of timelocks (Line 7), a vector of hashlocks (Line 8), and a Boolean unlocked vector marking which hashlocks have been unlocked (Line 9).

When the contract is initialized, its constructor copies the fields provided into the contract's long-lived state (Lines 21-23) and sets each entry in unlocked to *false* (Line 24).

The unlock() function (Line 26), callable only by the counterparty (Line 27), takes an index i, a secret s_i, a path p, and the signature sig. The hashlock h_i is unlocked if

- the current time is less than $T + (diam(\mathcal{D}) + |p|) \cdot \Delta$ (Line 28),
- $h_i = H(s_i)$ (Line 29),

[3] This pseudocode is based loosely on the popular Solidity programming language for smart contracts [24].

- p is a path in $\mathcal{D}$ from the counterparty to the leader who generated s_i (Line 30), and
- the signature is the result of signing s_i by the parties in p (Line 31)

The refund() function (Line 35), callable only by the party, refunds the asset to the party if any unlocked hashlock has timed out. The claim() function (Line 42), callable only by the counterparty (Line 36), transfers the asset to the counterparty if all hashlocks have been unlocked.

4.4 Pebble Games

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

In the *lazy* pebble game, start by placing pebbles on the arcs leaving each leader. Place new pebbles on the arcs leaving vertex v when there is a pebble on *every* arc entering v.

In the *eager* pebble game, start by placing a single pebble on one vertex z. Place new pebbles on the arcs leaving v when there is a pebble on *any* arc entering v. Both games continue until no more pebbles can be placed.

Lemma 4.1. *In the lazy game, 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 4.2. *In the eager game, 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. Because $\mathcal{G}$ is strongly connected, there is a path from z to v. Since z has a pebble and v does not, there is an arc (w, w') on that path where w has a pebble but w' does not, so w' will get a pebble in the next step, contradicting the hypothesis that the game has stopped. □

Suppose 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.

Lemma 4.3. *In both pebble games, every arc will have a pebble in time at most $diam(\mathcal{D}) \cdot \Delta$ from when the game started.*

Proof. For the lazy game, 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.

For the eager game, it is enough to observe that in each interval of time Δ, for every vertex v, the number of unpebbled vertexes in every path from z to w shrinks by one. Because $\mathcal{D}$ is strongly connected, such a path always exists. □

Corollary 4.4. *Under the stated timing assumptions, for both games, every arc has a pebble within time $diam(\mathcal{D}) \cdot \Delta$.*

4.5 The Protocol

There are two phases. In Phase One, instances of the Swap contract (Figures 4 and 5) are propagated through $\mathcal{D}$, starting at the leaders. Each time a party observes that a contract has been published on an entering arc, it verifies that contract is a correct swap contract, and abandons the protocol otherwise.

Here is the Phase-One protocol for leaders:

(1) Publish a contract on every arc leaving the leader, then
(2) wait until contracts have been published on all arcs entering the leader.

Here is the protocol for followers:

(1) wait until correct contracts have been published on all arcs entering the vertex, then
(2) publish a contract on every arc leaving the vertex.

Figure 7 shows how contracts are propagated in a swap digraph with two leaders.

In Phase Two, the parties disseminate secrets via hashkeys. While contracts propagate in the direction of the arcs, from party to counterparty, hashkeys propagate in the opposite direction, from counterparty to party. Informally, each party is motivated to trigger the contracts on entering arcs to acquire the assets controlled by those contracts.

We now trace how the secret s_i generated by leader v_i is propagated. At the start of the phase, v_i calls unlock$(s_i, v_i, sig(s_i, v_i))$ at each entering arc's contract (here, the function's arguments are the hashkey, and v_i is a degenerate path). The first time any other party v observes that hashlock h_i on a leaving arc's contract has been unlocked by a call to unlock(s_i, p, σ), it calls unlock$(s_i, v+p, sig(\sigma, v))$ at each entering arc's contract. The propagation of s_i is complete when h_i has either timed out or has been unlocked on all arcs.

Lemma 4.5. *If all parties conform to the protocol, then every arc has a contract within time $diam(\mathcal{D}) \cdot \Delta$ of when the protocol started.*

Proof. Phase One is an instance of the lazy pebble game on $\mathcal{D}$, so the claim follows from Lemmas 4.1 and 4.3. □

Lemma 4.6. *If all parties conform to the protocol, then every arc's contract is triggered within time $2 \cdot diam(\mathcal{D}) \cdot \Delta$ of when the protocol started.*

Proof. Each secret's dissemination is an instance of the eager pebble games on $\mathcal{D}^T$, the transpose digraph. The secrets are disseminated in parallel. □

Theorem 4.7. *If all parties conform to the protocol, then every contract is triggered within time $2 \cdot diam(\mathcal{D}) \cdot \Delta$ of when the protocol started.*

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

There is a simple optimization that ensures that Phase Two completes in constant time when all parties conform to the protocol. We use a shared blockchain, perhaps that of the market-clearing service, as a broadcast medium. Each leader v_i publishes its secret s_i on the shared blockchain, and each follower monitors that blockchain, triggering its entering arcs when it learns the secret. (Logically, we create an arc from each follower directly to that leader.) Unfortunately, while this broadcasting blockchain can "short-circuit" the Phase Two protocol, it cannot replace it, because a deviating leader might refrain from publishing the secret on that blockchain, but publish it on others. (Miller *et al.* [18] propose a similar optimization for the Lightning network.)

Lemma 4.8. *If hashlock h times out on any arc entering a conforming v, then h must have timed out on every arc leaving v.*

Proof. Suppose h was triggered on (v, w) by hashkey (s, p, σ). If v does not appear in p, then $v + p$ is a path from v to the leader, and v can immediately trigger h on (u, v) using the hashkey $(s, v + p, sig(\sigma, v))$, which has not timed out. If v appears in p, then v has already received (and signed) a hashkey that triggers h on (u, v). □

Theorem 4.9. *No conforming party ends up* Underwater.

Proof. Assume by way of contradiction that some conforming party v ends up Underwater: a leaving arc (v, w) has a triggered contract, but an entering arc (u, v) does not and will not.

First, arc (u, v) must have a contract. Suppose v is a leader. Since (v, w) has been triggered, v has revealed its secret through a hashkey. But a leader issues hashkeys in Phase Two only after a contract has been published on every entering arc during Phase One. Suppose instead v is a follower. Since (v, w) has been triggered, one of the arcs leaving v has a contract. But in Phase One, a follower publishes a contract on a leaving arc only after contracts have been published on all of its incoming arcs.

Since (u, v) has a contract, one of that arc's hashlocks must have timed out. By Lemma 4.8, the arc (v, w) must also have timed out, a contradiction. □

Theorem 4.10. *For $\mathcal{D} = (V, A)$ with leaders $L \subset V$, the space complexity, measured as the number of bits stored on all blockchains, is $O(|A|^2)$.*

Proof. There are $|A|$ contracts, one on each arc, each with a copy of the digraph $\mathcal{D}$, which requires $O(|A|)$ storage. □

Finally, any atomic cross-chain swap protocol using hashed timelocks must assign secrets to a feedback vertex set.

Lemma 4.11. *In any uniform hashed timelock swap protocol, no follower v can publish a contract on an arc leaving v before contracts have been published on all arcs entering v.*

Proof. If follower v has has a contract on arc (v, w) but no contract on arc (u, v), then the parties other than v could collude to trigger the contract on (v, w), while refusing to publish a contract on (u, v), leaving v Underwater. □

Theorem 4.12. *In any uniform swap protocol based on hashed timelocks, 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 4.11 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 follower v can publish a contract on a leaving arc only if v has indegree zero in the current waits-for digraph. But no vertex on a cycle in the waits-for digraph will ever have indegree zero, a contradiction. □

4.6 Single-Leader Digraphs

As noted, in the common special case where a swap digraph needs only one leader, we can replace hashkeys with simple timeouts, reducing message sizes and eliminating the need for digital signatures. In the following, let $\mathcal{D}$ be a swap digraph with a single leader $\hat{v}$ with hashlock h.

Lemma 4.13. *If each arc (u, v) has timeout $(diam(\mathcal{D}) + D(v, \hat{v}) + 1) \cdot \Delta$, then for every conforming $v \neq \hat{v}$, the timeout on each arc (u, v) is later by at least Δ than the timeout on each arc (v, w).*

Proof. Let p be the longest path from w to the leader $\hat{v}$. Because the subdigraph of followers is acyclic, $v + p$ is a path of length $D(w, \hat{v}) + 1$ from v to $\hat{v}$, so $D(v, \hat{v}) \geq D(w, \hat{v}) + 1$. □

Lemma 4.14. *For a single-leader digraph using timeouts, if hashlock h times out on any arc entering a conforming v, then h must have timed out on every arc leaving v.*

Proof. By Lemma 4.14, once h is triggered on (v, w), v has time at least Δ to trigger (u, v). □

From this point on, the bounds on running time and proofs of safety for the single-leader-using-timeouts protocol are essentially the same as for the general protocol.

5 REMARKS

We have seen that single-leader swap digraphs do not require hashkeys and digital signatures, only timeouts. Is there a way to reduce the use of digital signatures in the general case?

Finding a minimal feedback vertex set for $\mathcal{D}$ is NP-complete [15], 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], reflecting the situation 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. We leave for future work the question whether one could require parties to post bonds, and following a failed swap. examine the blockchains to determine who was at fault (by failing to execute an enabled transition).

An interesting open problem is the extent to which this swap protocol can be modified to provide better privacy, analogous to the way the Bolt network [12] improves on Lightning.

As noted, some parties may be willing to accept certain Underwater outcomes rejected by the swap protocol presented here. Future work might investigate protocols where parties are endowed with customized objective functions to provide finer-grained control which outcomes are acceptable.

The swap protocol can be made recurrent by having the leaders distribute the next round's hashlocks in Phase Two of the previous round. If swaps are recurrent, then it would be useful to conduct swaps *off-chain* as much as possible, similar to the way that Lightning [22] and Raiden [19] networks support off-chain transactions for bitcoin and ERC20 tokens.

One limitation of the swap protocol presented here is the assumption that the swap digraph, its leaders, and their hashlocks are common knowledge among the participants. Future work might address constructing and propagating this information dynamically.

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, 20]. There is open-source code [6, 9, 21] for two-party cross-chain swap protocols between selected currencies, and proposals for applications using swaps [27].

Off-chain payment networks [8, 12, 19, 22] circumvent the scalability limits of existing blockchains by conducting multiple transactions off the blockchain, eventually resolving final balances through a single on-chain transaction. The Revive network [16] rebalances off-chain networks in a way that ensures that compliant parties do not end up worse off. 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, 10, 13] have been proposed for matching donors and recipients. Shapley and Scarf [23] consider the circumstances under which certain kinds of swap markets have strong equilibriums. Kaplan [14] 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 [11, 17] 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.

A *atomic cross-chain transaction* is a distributed task where a *sequence* of exchanges occurs at each blockchain. An atomic cross-chain swap is an atomic cross-chain transaction, but not vice-versa, because not all transactions can be expressed as swaps. In our original example, Alice could not borrow bitcoins from Bob to pay Carol, because then Alice would have to execute two steps in sequence (borrow, then spend) instead of executing a single swap. A better understanding of atomic cross-chain transactions is the subject of future work.

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. As of 9 January 2018.

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

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

[7] V. Buterin. On sharding blockchains. https://github.com/ethereum/wiki/wiki/Sharding-FAQ. As of 8 January 2018.

[8] C. Decker and R. Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In A. Pelc and A. A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18, Cham, 2015. Springer International Publishing.

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

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] R. Khalil and A. Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 439–453, New York, NY, USA, 2017. ACM.

[17] 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.

[18] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry. Sprites: Payment channels that go faster than lightning. *CoRR*, abs/1702.05812, 2017.

[19] R. Network. What is the raiden network? https://raiden.network/101.html. As of 26 January 2018.

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

[21] 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. As of 9 January 2018.

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

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

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

[25] G. Weikum and G. Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[26] Wikipedia. Two-phase commit protocol. https://en.wikipedia.org/wiki/Two-phase_commit_protocol. As of 18 May 2018.

[27] 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. As of 25 January 2018.