Cross-Chain Atomic Swaps without Time Locks

1st Ras Dwivedi
Computer Science and Engineering
Indian Institute of Technology, Kanpur

2nd Tushar Singla
Computer Science and Engineering
Indian Institute of Technology Kanpur

3rd Prof. Sandeep Shukla
Computer Science and Engineering
Indian Institute of Technology Kanpur

***Abstract*—Since the introduction of the blockchain technology in the form of the Bitcoin [1] cryptocurrency, numerous blockchains have been introduced with applications beyond cryptocurrency. Introduction of smart contracts and tokenization of assets has made blockchain technology a decentralized, replicated and transparent program execution engine with the capability to represent real-world assets as tokens. However, with the diversity of blockchain platforms, and diversity of applications, interoperability between applications has become an important problem. Especially with the increasing use of asset tokenizations, cross-chain token transfer which guarantees safety and live-ness properties became necessary. Atomic cross chain transactions are building blocks for the cross chain execution of the smart contract. Atomic cross-chain swap is the simplest kind of the cross chain transaction. In this paper we present a novel atomic cross chain swap protocol. Our protocol does not use time locks unlike previously proposed solutions. Time-locking leaves the funds locked at the whim of the proposer, and hash-time locking protocol is probabilistic, in the sense that there is a possibility that proposer might not complete the transaction and cheat the other party. Our protocol is deterministic and ends in a finite number of steps. We also provide formal proofs of safety and liveness properties of our protocol.**

***Index Terms*—fair exchange, atomic swaps, blockchain, smart contracts, Hashed timelock Contract (HTLC), atomicity, inteoperability, distributed ledger technology**

I. Introduction

The blockchain concept was introduced in 2008 with the release of the Bitcoin [1] white paper. A large number of cryptocurrencies have been introduced in the last 15 years, taking advantage of the blockchain technology. As of 2020, there exist more than 22000 different cryptocurrencies [2]. Beyond the Bitcoin blockchain, new capabilities have been introduced to blockchains – such as smart contract support, privacy, digital identities, tokenization etc. Blockchain was designed as a self sufficient trusted ledger and hence it cannot interact with other parties trustfully. As a result when different blockchains emerge, they exist as different isolated blockchain in silos, each holding its own trusted ledger. However, for blockchain technology to thrive, applications built on different blockchains should be able to communicate with each other, exchange information, execute smart contract based on the information received from the other blockchain, and transfer assets from each other.

Currently, there are 86,000 blockchain projects proposed in GitHub and the number is growing at a rate of 20,000 projects per year. However, the average lifespan of these projects is just 1.22 years. As the blockchain ecosystem is very heterogeneous catering to different use cases, it is clear that one blockchain is not going to be satisfactory for all. An interoperable ecosystem enables multiple users from separate blockchain networks to interact without needing to migrate from their preferred blockchain. The problem of interoperability emerges due to the fact that different networks use different programming languages, privacy measures, protocols, consensus mechanisms etc. It is infeasible to achieve interoperability by making each pair of network to be compatible with each other. Therefore, we need to develop some generic frameworks which could be supported by each network and help in achieving interoperability.

In this paper we focus on a very special case of the cross chain transactions: Atomic cross chain swaps. Suppose that Alice and Bob are members of two blockchain networks, DollarNet and EuroNet. Alice and Bob want to exchange Euro and Dollar. Suppose Alice has 1 Dollar in DollarNet and Bob has an equivalent amount of Euro in EuroNet and they both want to exchange the Dollar and Euro. An Atomic swap protocol is a protocol where either Alice and Bob would retain their respective Dollar and Euro or they should be able to exchange it. There would be no scenarios where either Alice or Bob ends up with all or none of the assets.

The remainder of the paper is organized as follows. In Section 2 we provide a general overview of the blockchain followed by Section 3 describing the motivation for atomic swap. We also list various scenarios where atomic swap would be helpful. We then give a brief background on the work done on atomic swaps. We present our architecture in section 6. In Section 7 and 8 we describe the implementation followed by the protocol in detail. In section 9 and 10 we formally verify the protocol using Kripke structure and show it correctness. In Section 11 we conclude the paper.

II. Blockchain Overview

Blockchain [3] is a replicated, immutable ledger in a decentralized network, where the data is in the form of the blocks, arranged sequentially and each block is linked to its predecessor block using a cryptographic hash function. Depending upon who have read/write access to the distributed ledger, blockchain could be classified as public or private. Similarly, based on who can participate in the network, a blockchain may be permissionless or permissioned. Bitcoin [1] and Ethereum[4] are examples of the permissionless public blockchain, while Hyperledger Fabric is an example of permissioned blockchain which may be private or public. Since membership of permissioned blockchain is restricted and

verified, it is unlikely to have Sybil attack.
All participant nodes (or a majority of the nodes) in the network maintaining a blockchain have to reach a consensus on which block gets added to the blockchain. Bitcoin blockchain uses proof of work (POW) to achieve this consensus, Ethereum uses proof of stake (PoS), because they have to resist the Sybil attack. On the other hand, permissioned blockchains use less computationally expensive mechanisms like Kafka, Raft [5] or Byzantine fault-tolerant (BFT) consensus.

III. Motivation

Interoperability is the ability to exchange data/assets across multiple blockchain platforms, including those running different paradigms of blockchains, as well as with the off-chain world (EU Blockchain Observatory and Forum, 2019). For realizing the full potential of the blockchain technology, we would need interoperable systems. We can better illustrate the the need for interoperability through the following example applications of blockchain technology:

A. Central Bank Digital Currency(CBDC)

rrntly, Central Banks around the world are experimenting with central bank digital currencies in one way or the other. Several countries including India [6]have even started large scale pilots of CBDC implementation. The envisioned advantages of CBDCs are financial inclusion of unbanked citizens, faster and immediate payment settlement, easier and faster cross border transactions, etc. Since, CBDCs will be operated by different central banks, following different architecture and protocols, without blockchain interoperability solutions, it would be difficult to enable cross border payments with CBDCs. Therefore, innovative and efficient solutions will be needed to tackle this difficulty.

B. Supply Chain

Blockchain technology has great potential to disrupt supply chain management. It could improve transparency, traceability and reduce costs for companies. One major challenge faced during its implementation is the ability to run with legacy systems. Also, different vendors will be using different blockchains, so they should be able to interact with each other.

C. Health Care

In healthcare, exchange of information is required between different vendors, providers, doctors and patients across different organizations and systems. Therefore interoperable systems are required for adoption of the technology in the space.

With these examples in view, we emphasize interoperability among different blockchain systems as a crucial need in the blockchain ecosystem. It enables the seamless exchange of digital assets across different chains and facilitates the coexistence of different blockchains. Atomic cross-chain swaps have emerged as the basic building block of interoperability, allowing trustless exchange of assets between different blockchain systems. Our focus is on the development of a novel protocol that facilitates such atomic swaps and enables secure and reliable exchange of digital assets between different blockchains.

IV. Background and Related Work

Fair exchange has been studied in the literature and they were shown to be as hard as consensus [7]. It was shown that reaching a fair exchange between two parties, without involvement of trusted third party is impossible. However, protocol becomes feasible with the involvement of the third party. If there is a trusted third party that can take dollars from Alice and Euros from Bob and facilitate the exchange, that would work but will defy the idea of decentralization germane to the use of blockchain technology. For example, centralized exchanges for cryptocurrencies function on the model of a trusted third party. However, centralized exchanges are vulnerable to the distributed denial of service attacks , wallet attacks, hacking, rug pulls and various other problems.

To decentralize the exchange of assets across multiple blockchains would be to use smart contracts. Smart contract executions are replicated across all validator nodes in a blockchain network, are subject to audit, and tamper proof. However, smart contract based solution works only when both the parties are on the same blockchain. For example if Alice and Bob want to exchange their tokens on Ethereum network, Alice can write a smart contract, and Bob can verify it, and then use that smart contract to reach a fair exchange.

Reaching a fair exchange across the blockchain has been a difficult task. As mentioned above centralized exchange does not figure well due to security consideration. Distributed exchange have become popular but they only provide match making services and require P2P execution within coordinated action, like HTLC (Hashed Timelock Contract)[8], probably with the collateral deposits . The use of hash based timelock contracts for two-party cross-chain swaps across multiple blockchain is believed to have emerged from on-line discussion forum [9] and [10]. They are used to execute atomic transaction across multiple blockchains. HTLC is essentially a smart contract locked by hash of a random number. Suppose that Alice and Bob are members of two blockchain networks, DollarNet and EuroNet. Alice and Bob want to exchange Euro and Dollar. Suppose Alice has 1 Dollar in DollarNet and Bob has an equivalent amount of Euro in EuroNet and they both want to exchange the Dollar and Euro. Alice would first create a hash of a random number 'n' and then lock her smart contract by the hash value. Now this smart contract could be unlocked by Bob by revealing 'n', or Alice can unlock it after certain time lets say 't_1'. Bob would then lock his asset using the same hash value which can be claimed by Alice by revealing 'n', or Bob can unlock it after time t_2 such that $t_2 < t_1$. Since bob does not know about 'n', the asset is locked for the time period. Once all the assets are locked, Alice reveals the 'n' and thereby captures Bob's assets. Bob now knows about the 'n' and takes Alice's asset. In this way assets are exchanged between two parties, on different chain using the hash Time locks. In HTLC based protocols there is time locking and the originating party have to wait till the time locks are not released if the second party does not respond. Time locks are problematic for other reasons too, that they are difficult to implement, and for two different blockchain networks, it is difficult to create a synchronous time.

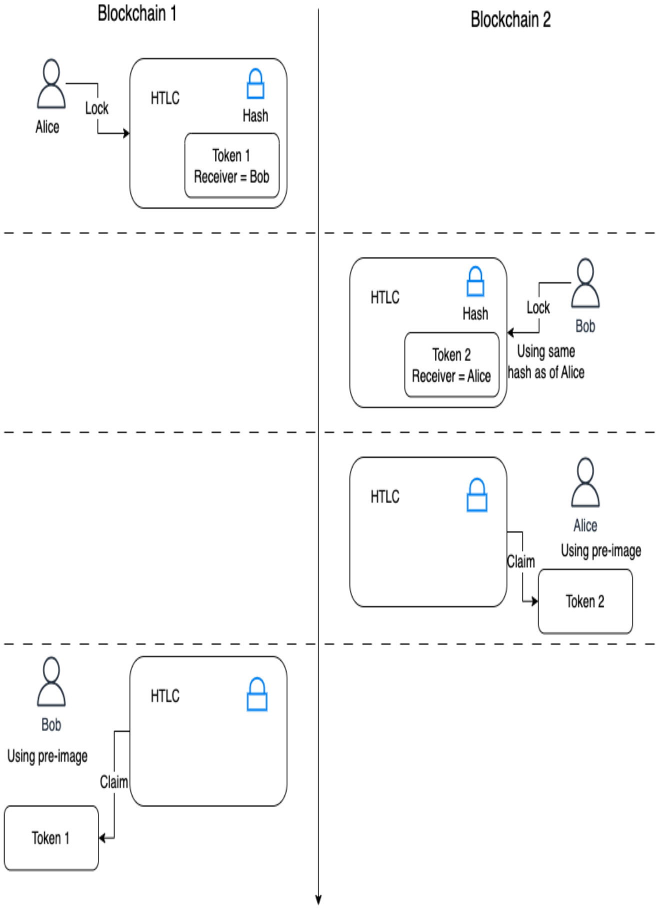


Fig. 1: Hash Time locked contract [11]

Interledger protocol(ILP) by Stefan Thomas and Evan Schwartz [12] allows interledger payments between two parties using the escrow facility provided by the ledgers. This is a trustless protocol and depends only on the security properties of the ledgers involved. In an ILP transaction from blockchain A to blockchain B, one must find escrows account having enough money on B for transfer to occur. The protocol is also based on HTLCs thus have same problems as HTLCs.

Rabin[13] proposed a probabilistic algorithm for conducting a fair exchange using a random beacon that emits a natural number after every time interval Δ. He used a digital signature as a commitment to transfer the assets.

To execute the fair exchange protocol, both parties, Alice and Bob, initiate the signing process in increasing order of natural numbers. At time t, Alice begins the protocol by signing the message that she would transfer her assets if the number emitted by the beacon at t_0 is greater or equal to i. Bob can reply with his commitment to transfer his assets to Alice if the number is greater or equal to i. Alice can then respond by committing to transfer her assets if the number is greater than $i + 1$. This process continues until the beacon emits the number at t_0.

If the number emitted by the beacon on the judgement day, N, is greater than Alice's commitment, i, she will transfer her assets. Similarly, if the number exceeds Bob's commitment, Bob will transfer his assets. If both parties transfer their assets, the atomic swap occurs. However, no exchange takes place if N is less than Alice's and Bob's commitments. It is important to note that this algorithm is probabilistic, and an unfair exchange is possible in certain scenarios. For instance, if Alice commits to number i while Bob commits to $i - 1$, and the beacon emits i, Alice must transfer her assets, whereas Bob does not need to.

Ashokan and Soup [14] also presented a way for fair exchange between two parties in the presence of a trusted third party. In their setup the third party is not actively involved and it is needed only when the one of the two parties involved in the exchange actually deviates from the protocol. There protocol involved two parties P and Q. At the beginning of the protocol P has item i_P, description of Q's item d_Q and similarly Q has the item i_Q and the description of P's item d_P. At the end of the protocol, either P has the item i_Q meeting the description d_Q or P learns nothing about d_Q and vice-versa for Q .

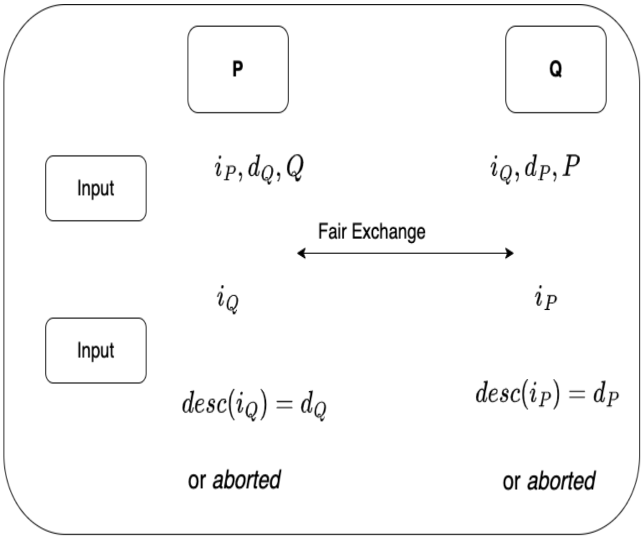


Fig. 2: A successful fair exchange

V. Our Contribution

In this paper, we present a deterministic method for executing cross-chain atomic swaps. Our protocol is inspired by the optimistic protocol for fair exchange by Ashokan and Soup [14], but with the key difference that it does not require a trusted third party. Instead, we assume that the execution of smart contracts on each blockchain network is trustworthy, and that smart contracts on the two blockchains can still execute an asynchronous fair exchange given user inputs.

Our protocol is optimistic, fast, and ends in a constant number of rounds. Unlike HTLCs, which rely on time locking, our protocol does not lock assets at any point in time. We also prove that our protocol is secure and works under asynchronous conditions. Additionally, our protocol is user agnostic, meaning that it does not depend on who initiates the protocol. Both parties are allowed to choose successive steps irrespective of who began the protocol.

Furthermore, our message exchange complexity is similar to HTLC-based atomic exchanges, but our protocol does not use time locking. Funds are never locked, and if any party wants to quit, they can do so at any point in the protocol, leading to either a fair exchange or the abortion of the protocol. The progress of the transactions is recorded on the respective blockchains.

Ashokan and Soup [14] introduced two notions of fairness for two party fair exchange. The first is strong fairness, where at the end of the protocol, either P receives i_Q or P learns nothing about i_Q. The second is weak fairness, where P either receives i_Q, learns nothing about i_Q, or can prove to an arbitrator that Q has received i_P or can still receive i_P. Although two party fair exchange with strong fairness has been shown to be at least as hard as consensus [7], which leads to an impossibility of fair exchange without a trusted third party [7], our protocol follows the notion of weak fairness, and thus, the impossibility result does not apply. Our underlying model is different in that users interact with smart contracts and all the transactions are on the blockchain which can be used to prove whether P transferred item i_P or not. Our protocol is effective, implying that if both P and Q behave correctly and do not want to abandon the exchange, the exchange will take place. Furthermore, all the transactions on the blockchain can be used to prove whether P transferred item i_P or not.

To the best of our knowledge, we are also the first one to provide a fair exchange by using two non-interacting smart contracts on two distinct blockchain networks.

VI. Architecture

A. Design Requirements

A fair exchange is an exchange of an agreed upon amount of two different assets between two parties. It should not be the case that a party receives assets of the other party without having to give up the agreed upon quantity of its own assets. It should also not be the case that both parties lose access to their assets in the process of the exchange. We would like our protocol to have following properties:

1) **Safety:** A safety property asserts that a bad state is never reached. In our case there would be two safety properties.
    - A malicious party is not able to acquire assets of honest party without parting away with its own assets. In an atomic swap, there should not be a state in which one party is in control of both the assets.
    - There should not be any deadlock. This implies that there should not be a state in which one party has no actions to take and wait indefinitely for the action of the other party. If such a state exists, then the malicious entity can disappear, locking the assets of the honest party.
2) **Liveness:** Liveness property implies that a good state is eventually reached. In out case, good state is defined as the state in which both the parties have their own assets with them, or when a fair exchange is completed, and both the parties have the assets of the other party. If from a given state, it is possible to reach one of the good states, we say that liveness property is satisfied in that state.

3) **No waiting time:** Hashed time lock protocol enables atomic swap between two blockchains, but it uses time locks, which implies that there is a maximum waiting period before which it is not possible to regard transaction as failed one. In our case, there is no waiting period. Both the parties have the option to roll back the transaction at any moment, if it feels that other party is not responding.

4) **Trustless:** The exchange should take place without the need of any trusted third party and should instead rely on the security properties of the ledgers involved.

These properties will ensure that fair exchange takes place and participating parties always have the option to abort and get their assets back.

B. Adversary Model

We assume that a malicious users will do anything in their reach to somehow acquire other party's assets. We made the following assumptions:

1) Adversaries cannot modify honest party's local copies of the ledger.
2) Cryptographic schemes being used are secure.
3) Adversaries cannot attack the ledgers/blockchains involved.
4) Adversaries cannot access honest party's secrets unless governed by the protocol.

These are quite obvious assumptions and under the scope of these assumptions we will show that our protocol is provably secure.

C. Description of Ashokan and Soup protocol

Ashokan and Soup proposed a fair exchange protocol that involves a trusted third party (T) to ensure fair exchange between Alice (A) and Bob (B). The protocol in the description is about signing the contract text between the two parties and exchanging the contract authenticators. The protocol begins with Alice signing a message containing the contract details and a commitment to the contract authenticator. Bob can choose to ignore the message or sign it along with a commitment to his contract authenticator. If Bob acts maliciously or fails to respond, Alice has the option to abort the contract. Otherwise, Alice sends the contract authenticator to Bob.

Bob's next move depends on whether he has received the contract authenticator from Alice. If he has, he can send his own contract authenticator to end the protocol and complete the exchange. However, if Bob acts maliciously and gives up before completing the exchange, Alice can call on the trusted third party T to sign the contract on behalf of both parties using the initial messages. On the other hand, if Bob has not received the contract authenticator, he can call for the intervention of the trusted third party T, who can then decide whether to abort or continue the exchange. The role of the trusted third party T is to remain passive during the protocol unless one of the two parties deviates from the protocol, in which case it is responsible for resolving or aborting the transaction.

VII. Implementation Details

We assume that a smart contract is deployed by Alice to transfer funds to Bob. Alice generates three random values, say x_1, x_2 and a and Bob also generates three random values, say y_1, y_2 and b. Both parties share the hashes of these values with each other over an off-chain secure channel. Each instance of the smart contract is identified by the three hashes and the account address of the receiver. Bob can use this information to verify whether the smart contract adheres to the agreed-upon terms or not. The smart contract is designed to have four possible states: S_0, S_1, S_2, and S_3. The smart contract has the following functions:

1) **Initialize(hashes) :** The *Initialize* function serves to establish a commitment to an exchange. This function requires input of six secret hashes, the transfer amount, and the address of the party to whom the funds are to be transferred. Prior to initializing the contract, Alice and Bob exchange the hashes through a private channel. Once the contract is initialized, the funds become locked with the hashes. It is worth noting that the contract is designed in a generic manner and can be reused with different parties, regardless of whether the protocol has been aborted or successfully terminated. Initially, upon deployment, the contract is in the state of S_0, transitioning to S_1 once it has been initialized.

2) **Quit(a) :** After deploying the contracts, Alice and Bob's state transitions to S_1, during which they may still roll back the process by invoking the 'quit' function. Invoking this function will reveal their third secret (a for Alice and b for Bob), subsequently unlocking their funds. It's important to note that our protocol does not lock up the funds, meaning that as long as Alice or Bob doesn't possess both secrets of the other party, they can always opt to quit the exchange. The act of quitting will change the state to S_0, which denotes the initial state where neither party has committed to the exchange.

3) **Freeze(x_1, y_1, y_2) :** "The 'freeze' function can only be invoked when the contract is in state S_1. In our case, Bob utilizes Alice's first secret, x_1, to freeze her contract. Once frozen, Alice is prohibited from calling the 'quit' function. However, she is still able to call 'abort' in the event that Bob rolls back his contract before she is able to claim his funds. Since Alice is unable to roll back at this point, Bob must reveal both of his secrets to enable the protocol to proceed. Subsequently, if Bob does not roll back, Alice can claim his funds. In the event that Bob does roll back, Alice can execute the 'abort' function to terminate the protocol.

4) **Resolve(x_1, x_2, y_1, y_2) :** Both Alice and Bob can call this function. It can be called only when the contract is in state S_1 or S_2. In our execution, Alice calls the resolve function when Bob's state was S_1 claiming Bob's fund. Bob uses the resolve function to claim Alice's funds when her contract is in state S_2. Note usage of the resolve protocol reveals all four secrets. Since the resolve protocol transfers the funds, there is no going back. Resolve changes the state to S_3

5) **Abort(a, b) :** This function can be called only by Alice when the smart contract is in state S_2. It needs two input parameters, namely Alice's and Bob's third secrets. It aborts the transaction and Alice gets her funds back and state is changed to S_0.

Now we define what these states represents.

1) S_0 represents the stage before the smart contract is deployed or after deploying party has quit or aborted the transaction and funds are transferred back to him..
2) S_1 represents the stage just after the smart contract is deployed and Alice's fund is locked.
3) S_2 represents the stage after freeze function is called.
4) S_3 represents the stage after funds is transferred to the other party.

It is important to mention here that when Alice and Bob decides to swap assets across two blockchains, and Alice wants to transfer an asset from the first blockchain to the second blockchain, and Bob wants to transfer an asset from Bob's account on the second blockchain to Alice's account on the first blockchain, they deploy the same smart contract on their respective blockchains. The parameters to functions described here then are to be interpreted such that where the above description mention's Bob's secret, it should be Alice's secrets and vice a versa for the smart contract deployed by Bob. Also note that, we assume that Alice and Bob must have accounts in both the blockchains. For a swap to happen, Alice transfers her asset to Bob's account on the first blockchain, and Bob transfers equivalent asset to Alice's account on the second blockchain. The exchange rate between assets is established through an oracle or through off-chain communication between Bob and Alice.

VIII. Protocol

Now we will describe the various steps of the protocol. We can describe a state during the protocol using a tuple

(A, B), where A is state of smart contract of Alice on the first blockchain and B is the state of smart contract of Bob on the second blockchain.

1) Alice will generate three random values, say x_1, x_2 and a, which are her secrets. Similarly, Bob will generate three random values, say y_1, y_2 and b, which are his secrets. Both of them share SHA256 hash values with each other using a secure private communication channel. The state is now (S_0, S_0)
2) Alice will now deploy the smart contract on DollarNet (first blockchain) locking her asset (say 1 Dollar) and call the initialize function. With the hashes of the six secrets being put into the first smart contract, the state becomes (S_1, S_0)
3) Bob will verify the correctness of Alice's smart contracts (check the hashes of the secrets) and deploy the smart contract in EuroNet (second blockchain) locking equivalent asset (say 1 Euro). Now the state becomes (S_1, S_1)
4) After verifying the correctness of Bob's contract (i.e. checking the hashes entered into Bob's contract are correct), Alice will share secret x_1 with Bob on a private communication channel and Bob will call freeze function of Alice's smart contract on the first blockchain revealing y_1 and y_2. Now the state becomes (S_2, S_1) Note that Alice can now see the plain text values of y_1 and y_2.
5) Alice will now call freeze function of Bob's smart contract therefore additionally revealing, x_2. Now the state becomes (S_2, S_2)
6) Now both of them can claim each others funds by calling resolve function in each others smart contracts on the respective blockchains. Now the state becomes (S_3, S_3)

We summarize the steps in table 1 and show the message flow sequence in fig 3.

Note that there is no timed lock in of assets during the entire process. If at any time Alice or Bob feels that the other party is not responding or is malicious, they can always get their funds back. Funds are locked after step 2. Now, if after step 2 Bob does not respond, Alice could simply call quit and get her funds back. Similarly at step 3, both of them has option to call quit. After step 4, Alice can call freeze or if Bob decides to give up, he can call quit revealing b and Alice can then call abort. After step 5, both of them can claim each other's assets.

TABLE I: Steps followed by Alice and Bob in successful transfer

Step No.	DollarNet	EuroNet
1.	Alice and Bob shares commitment to secrets	Alice and Bob shares commitment to Secrets
2.	Alice deploys the contract and locks it. She can still abort the contract by passing secret a	
3.		Bob deploys the contract and locks it. He can still abort the contract by passing secret b
4.	Alice passes x_1 to bob using private channel	
5.	Bob freezes Alice's contract with x_1, y_1, y_2.	
6.		Alice unlocks with x_i, x_2, y_1 and y_2. Alice receives 1 Euro
7.	Bob unlocks with x_1, x_2, y_1 and y_2. Bob receives 1 Dollar	

IX. Model checking using Kripke Structures

We create a Kripke structure for our protocol. A Kripke structure is defined as:

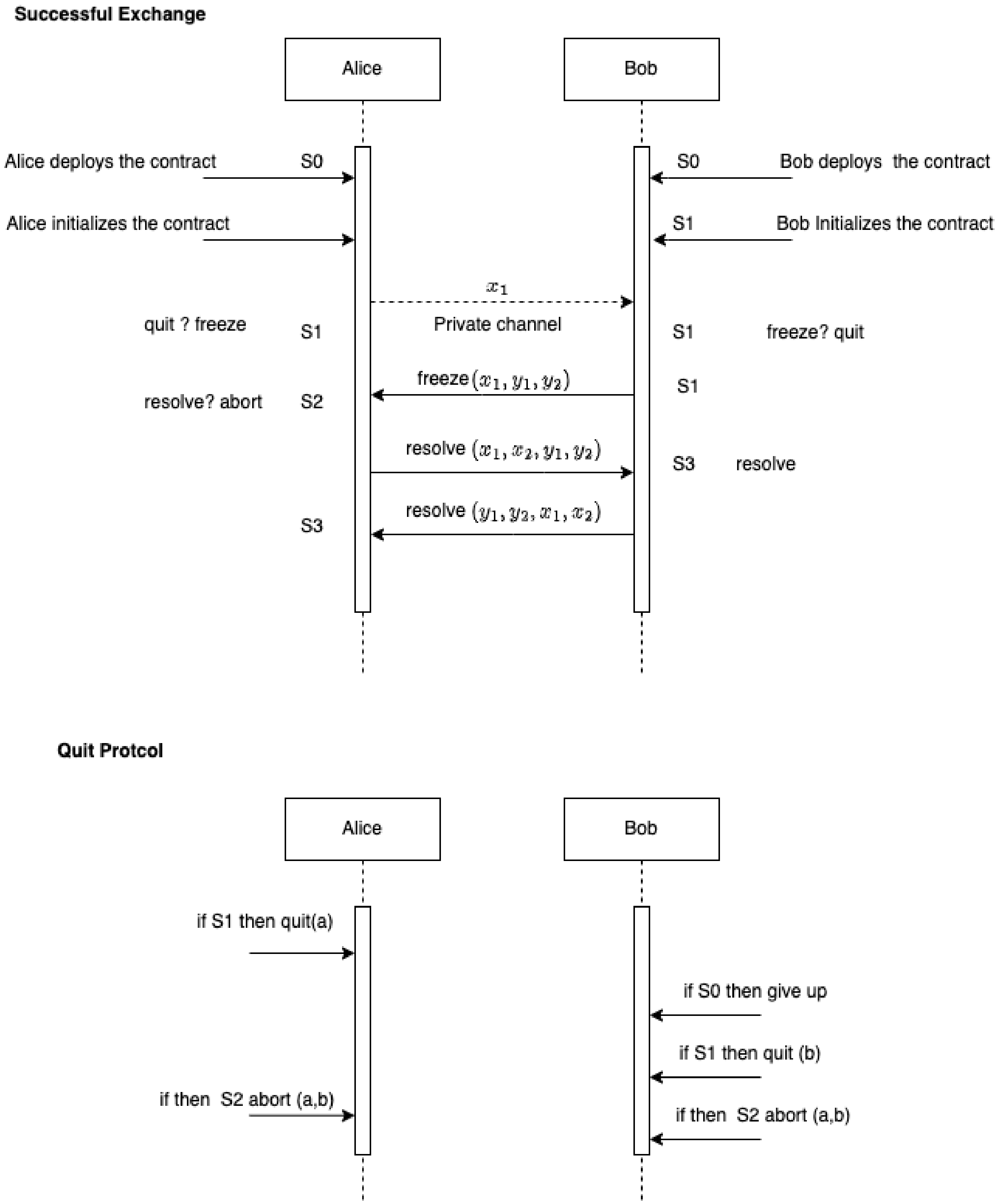
$$K = (V, S, s_0, L, R) \text{ Where,}$$

- V is a finite set of atomic propositions.
- S is a finite set of states
- $s_0 \in S$ is the initial state
- $L : S \longrightarrow 2^V$ is a labelling function that maps each state to the set of propositional variables that are true in that state.
- $R \subseteq S \times S$ is a transition relation

A transition from state p to state q is written as $p \longrightarrow q$. We model our system as Kripke Structure [15] where

- S is 8-tuple { A, B, $X1$, $X2$, $Y1$, $Y2$, S_{Alice}, S_{Bob} }, where
  - A, B, $X1$, $X2$, $Y1$, $Y2$ can take values true or false, and $S1$, $S2$ can take values 0, 1, 2 or 3.
  - A represents whether secret a of Alice is known to Bob or not. B represents whether secret b of Bob is known to Alice or not. Similarly, we define $X1$, $X2$, $Y1$ and $Y2$.
  - S_{Alice} represents the state of Alice's contract and S_{Bob} represents the state of Bob's contract.
- Transition relation $R \subseteq S \times S$ is defined as transition from one state to another in one step. In one step either Alice or Bob changes their state or Alice or Bob gets knowledge of secrets of the other party. Notice that secret can only be known, they cannot be forgotten.
- V = { good, bad } Where state $\{S_0, S_0\}$ and $\{S_3, S_3\}$ are good states, whereas state $\{S_0, S_3\}$ and $\{S_3, S_0\}$ are bad state. A good state is one in which either there is successful atomic swap, or both the parties retain their assets. If in a state one party has access to asset of the other party, and its own asset are either in freeze state or it is in his/her own possession then that state is labelled as bad state. Remaining of the states are labelled recursively as follows
  - if a bad state could be reached from a state, it is labelled as bad state
  - if a good state could be reached from a given state, and it has not been labelled as bad state it is good state
  - if a state has not been labelled, it is bad state.

  we construct the graph using states as vertices, and transition relations as edges and use depth-first-search to



Successful Exchange

Alice
Bob
Alice deploys the contract
S0
S0
Bob deploys the contract
Alice initializes the contract
S1
Bob Initializes the contract
x_1
Private channel
quit ? freeze
S1
S1
freeze? quit
freeze (x_1, y_1, y_2)
S1
resolve? abort
S2
resolve (x_1, x_2, y_1, y_2)
S3
resolve
resolve (y_1, y_2, x_1, x_2)
S3

Quit Protcol

Fig. 3: Message flow in fair exchange. The first part shows the successful exchange, while the later part shows the Quit protocol. Alice and Bob can quit either by calling the method quit when in state S1 or by aborting in state S2

label the states.

- Start state { false, false, false, false, false, false , S0, S0 } and s_0 is $\{S_0, S_0\}$, which is labelled as good state.

X. Correctness

For analysis of our protocol, we consider both safety and liveness conditions under the assumption that the participating parties are not unnecessarily sharing their secrets to each other and are sharing them only when the protocol requires them to share.

A. Safety

Our safety property is $AG \neg bad$ (i.e., In all paths, no state is a bad state).
We defined a bad state as one from where either it is possible to reach the states $(S_0, S_3), (S_3, S_0)$ or the states from where it is not possible to reach a good state. First one shows that a malicious party is not ale to acquire assets of another party, without parting with its own, and the second one shows a deadlock state where it is not possible to reach any of the end states. Both of the safety properties are combined in this CTL (Computational Tree Logic) [16] logic formulae.

To prove the safety property we constructed a graph of all possible state transitions under the assumption that the participating parties are not unnecessarily sharing their secrets. Since number of states are finite $(2^6 * 4 * 4 = 1024)$, we construct a graph G with states as vertices and transition relation as edges. We do a graph traversal starting from state (S_0, S_0). None of the malicious states were reachable from the starting state.

B. Liveness

Liveness means that eventually a desired state ((S_3, S_3) or (S_0, S_0)) is reached i.e., either swap takes place or both parties quit. We proved the property using fixed point model checking technique. We first constructed the maximal set of states from which either of the desired states could be reached and then showed that this set is equal to the set of all reachable states from (S_0, S_0). Now since from each reachable state we finally reach the desired states, the liveness property is proved.

XI. Results

We implemented our atomic swaps with Hyperledger fabric network. We created two different channels modeling two different blockchains (note that channels in Hyperledger are for all practical purposes distinct blockchains with no visibility into each other). In our implementation we use a RAFT based ordering service. We were able to reach throughput of 50 swaps per second before transactions started dropping. Since each swap needs about 5 transactions, this is equivalent of 250 TPS. We also observed that latency is high initially but then reduces as the number of swaps per second is increased. It rises again after 50 swaps per second. We reason that the initial high latency is due to the fact that a single swap needs at least 5 blocks. Since the time for 5 block has a lower limit, we have high initial latency per swap. As more and more swaps are processed in the single block, average latency decreases.

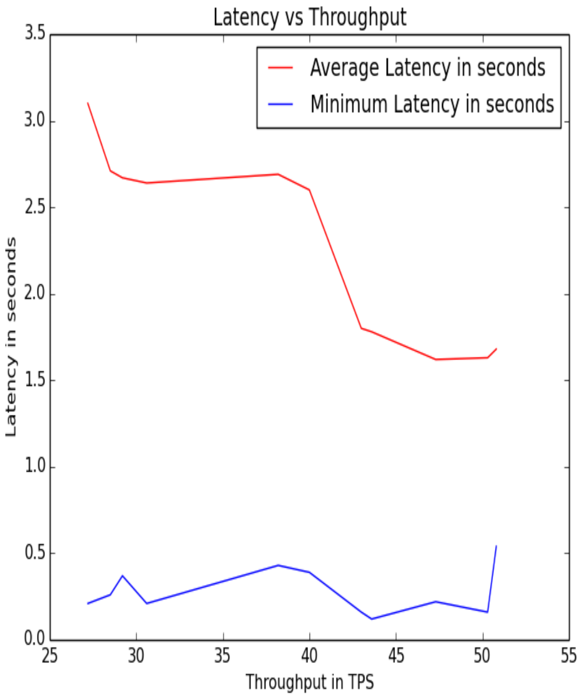


Fig. 4: Latency vs throughput

XII. Conclusion and Future Scope

The blockchain technology is currently in an evolving stage. Currently there are a lot of issues with the technology which prevents it from being adopted at a large scale in real world. A lot of research is required before all these issues could be solved. Interoperability is one such vital problem that remains to be solved. Achieving interoperability will allow multiple blockchains to co-exist and function together. We have solved only a very special case of the problem, i.e, Atomic cross chain swaps. We have presented a protocol which allows us to atomically exchange assets across different blockchains without any time locks. But one of the major limitations of our protocol is that for each atomic swap there are 3 interactions with each blockchain involved which increases the transaction latency. Further research is required to reduce the transaction time.

Bibliography

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf (2008).
2. How Many Cryptocurrencies are There In 2023? As on 13, April, 2023. \url{"https://explodingtopics.com/blog/number-of-cryptocurrencies"}.

3. Zheng, Z., Xie, S., Dai, H.-N. & Chen, X. An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. IEEE International Congress on Big Data, 557–564. https : / / www . researchgate . net / publication / 318131748_An_Overview_of_Blockchain_Technology_Architecture_Consensus_and_Future_Trends (2017).
4. Buterin, V. Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform. Accessed April 9, 2023 (2014).
5. Ongaro, D. & Ousterhout, J. In Search of an Understandable Consensus Algorithm. arXiv preprint arXiv:1407.3321. https : / / raft . github . io/raft.pdf (2014).
6. Concept Note on Central Bank Digital Currency As on 13th April, 2023. ”https : / / rbi . org . in / Scripts / PublicationReportDetails . aspx ? UrlPage = &ID = 1218 # CP2”.
7. Pagnia, H. & Darmstadt, F. C. G. On the Impossibility of Fair Exchange without a Trusted Third Party (1999).
8. Alt chains and atomic trasnfer As on 13th April, 2023. ”https://bitcointalk.org/index.php?topic=193281.0”.
9. Atomic Cross-Chain Trading As on 13th April, 2023. https://en.bitcoin.it/wiki/Atomic_cross-chain_trading.
10. Atomic swaps using cut and choose. As on 13th April, 2023. https://bitcointalk.org/index.php?topic=1364951.
11. Grano, H. Hashed time locked contract [Photograph]. Accessed: April 13, 2023. 2022. \url{https : / / miro . medium . com / v2 / resize : fit : 720 / format : webp / 1 * imzwEHi4ygfJlMr5EiN7cg.png}.
12. Thomas, S. & Schwartz, E. Interledger Protocol (ILP) in 2015 IEEE Symposium on Security and Privacy Workshops (2015), 3–3.
13. Rabin, M. Transaction protection by beacons tech. rep. 29-81 (Aiken Computation Laboratory, Harvard University, 1981).
14. Asokan, N., Shoup, V. & Waidner, M. Asynchronous Protocols for Optimistic Fair Exchange in (IEEE Computer Society, Los Alamitos, CA, USA, 1998), 86–99. ISBN: 0-8186-8287-3. https ://doi.org/10.1109/SECPRI.1998.674073.
15. Kripke, S. Semantical Considerations on Modal Logic. Acta Philosophica Fennica **16,** 83–94 (1963).
16. Clarke, E. M. & Emerson, E. A. Computation tree logic. POPL ’81: