Atomic Cross Chain Swaps via Relays and Adapters

Léonard Lys
llys@palo-it.com
LIP6, Sorbonne Université
Palo IT
Paris, France

Arthur Micoulet
amicoulet@palo-it.com
Palo IT
Paris, France

Maria Potop-Butucaru
maria.potop-butucaru@lip6.fr
LIP6, Sorbonne Université
Paris, France

ABSTRACT

Blockchain technologies have proven their potential when it comes to store assets and value. However, swapping assets across chains, for example trading ethers for bitcoins is still a challenging problem to solve. Current solutions widely rely on trusted third parties such as exchanges, which is not acceptable for a distributed technology.

Atomic Cross Chain Swap protocols that use hash time locked contract have been proposed as a solution for inter-chain exchanges but the atomiticity of the transaction can be violated in case of client crash or packet loss. Furthermore those protocols are impractical as they require the user to perform complex audits and transactions on several blockchains over relatively long period of time.

In this paper we propose to solve the shortcomings of hash time locked cross chain transaction by using relays and adapters in order to automate contract auditing and user actions as well as preventing atomicity violation.

CCS CONCEPTS

• **Computer systems organization** → *Peer-to-peer architectures*; *n-tier architectures*; • **Networks** → *Network design and planning algorithms.*

KEYWORDS

Blockchain interoperability, Blockchain Relays, Atomic Swap, Two Phase Commit, Atomic Commitment

ACM Reference Format:
Léonard Lys, Arthur Micoulet, and Maria Potop-Butucaru. 2020. Atomic Cross Chain Swaps via Relays and Adapters. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

It is no longer uncommon to hear Bitcoin compared to gold as a store of value. In addition, more and more projects are reporting on the potential of open blockchains as a medium of exchange [1] [3]. However, one obvious use case is still difficult to achieve without a trusted intermediary : inter-chain exchanges. An inter-chain exchange is when some asset (coin or token) of a blockchain is exchanged for an other asset hosted on an other chain. For example exchanging bitcoins for ethers.

Today, and although this is the use case that generates the most volume, inter-chain exchanges mainly rely on notary schemes implying trusted intermediaries (i.e. centralized exchange platforms). This is the result of the genesis of open blockchain networks : most of them were developed by open source communities that wanted to take the lead as a blockchain network. Therefore interoperability "by design" was not always part of the specifications. As a result of this, most of inter-chain exchange are executed trough the help of a custodian trusted third party i.e exchanges.

As stated in [16], all benefits of blockchain technologies are lost if one has to delegate trust to an intermediary. Indeed, delegating your trust to an exchange platform or any trusted intermediary results in several flaws : 1)SPOF : At some point of the exchange, both assets are stored inside the exchange wallet, making it a prime target for attackers. Most of the cryptocurrencies that have been stolen the past few years have been stolen from exchanges [7]. 2) Increased fees : When exchanging an asset trough an exchange platform one must deposit the assets inside the exchange's wallet, open a trade, perform the exchange and finally withdraw the coins from the exchange. Not only this increases the number of transactions required to perform an exchange but also at each stage of this process, the exchange platform is likely to take a fee. 3) Governance : Because of the way they operate, exchange platforms end up having tremendous amount of cryptocurrencies in their wallets. This can be used as tool to manipulate the market [20]. They have also been reported faking the volume of their market [17].

The flaws of centralized notary schemes to achieve inter-chain exchanges have been observed by the scientific community and led us to the conclusion that we need peer to peer, trust-less protocols to swap assets across chains. Interoperability is now part of many blockchain network project's whitepapers [14] [19] [18] but swapping assets across "legacy" chains (i.e all blockchains that are not inter-operable by design) remains a challenge.

In order to face this challenge, two main tracks are being explored; sidechains/relays and hash time locking [9]. Each of those two tracks presenting some pros and cons.

On the one hand, Hash time locking as described in [11] requires the user to be online during most of the swap and only fulfills the atomicity property with the assumption that the user's blockchain client does not crash. This is not sufficient for real life application where crash and packet lost is common. On the other hand, developing a side chain just for interoperability comes with great complexity, risks of mining centralisation and risks of forks [8].

In this paper we present Relay Swap, a protocol that takes the best out of both tracks in order to fix the flaws of hash time locked contract. To do so, it makes use of blockchain *relays* and blockchain

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

adapters to automate audit and transaction broadcast as well as preventing risks of atomicity violation.

2 MOTIVATION

Although hash time locking seems like the go to solution for inter-chain exchanges, it comes with several flaws. As explained earlier we will overcome those flaws thanks to relays and adapters. In this section we will recall hash time lock based atomic cross chain swaps scheme and discuss their flaws. Then we will detail a recent proposal called AC3WN which tries to overcome the same flaws and discuss its drawbacks.

2.1 ACCS scheme

2.1.1 *ACCS scheme Overview.*

An atomic cross-chain swap (or ACCS) [11] is a distributed coordination task where multiple parties exchange assets across multiple blockchains.

To be considered so, an atomic swap protocol must ensure the all or nothing property, also known as *atomicity*. In other word, if all parties are honest and conform to the protocol then the exchange takes place (1) and if any party deviates from the protocol, then no honest party can end up worse of (2).

One way of achieving ACCS is to make use of Hash Time Locked Contracts. Those contracts make use of time locking and pre-image revelation. They allow a party A to lock some assets (or coins) on a blockchain such that they can be unlocked in two manners: by party A after a period of time Δ or by party B right away but only if party B is able to provide a secret s.

By setting two similar contracts on two different blockchains, A and B can safely exchange their coins without having to trust each other or any other third party.

A hashed time locked contract (or just HTLC) can be defined by parameters h, A, B, v, t. Parameter h is the computed image of a secret $h = H(s)$, H being a cryptographic hash function. Parameter A represents the blockchain address of the swap initiator and B the one of the recipient (or participant). Parameter v is the amount of coins transferred and t is the period of time during which the assets cannot be unlocked by A.

The scenario of an Atomic Cross Chain Swap between two parties, Alice and Bob, exchanging one Ether for one Bitcoin is as follows:

- (1)Alice creates a secret s and computes its image $h = H(s)$. Then, Alice locks her Bitcoins on a hashed time-lock contract with parameters h, Alice's Bitcoin address, Bob's Bitcoin address, $v = 1$, $t = 2\Delta$.
- (2)Bob looks up Alice's contract and checks that the correct recipient, amount and time-lock have been set.
- (3)Bob creates a similar HTLC on the Ethereum blockchain with parameters h, Bob's Ethereum address, Alice's Ethereum address, $v = 1$, $t = \Delta$.
- (4)Alice unlocks the ether that has been locked by Bob. Doing so, she reveals to Bob the secret s that he did not know.
- (5)With the secret Bob just unveils, he can now withdraw the Bitcoin that has been locked by Alice.

The one that has generated the secret and hence the hash lock is called the initiator. Similarly, the one that doesn't know the secret while contracts are being set is called the participant (we may refer at both of them as "participants").

The time-lock value matters. That is, the time-lock of the party that has generated the secret s (or the initiator) has to be higher than the other one (or the participant). Indeed if the Δ has been set for the contracts to elapse at the same time, then Alice could wait for the time to elapse, withdraw the Ethers when $t \rightarrow \Delta$ and then withdraw the Bitcoins at $t > \Delta$.

If the parameters have been correctly set, at no stage of this scenario any malicious party is able to withdraw both coins. If any party halts at any stage of the protocol, the secret s will not be revealed and eventually all contracts will time out and both parties will be able to withdraw their coins.

2.1.2 *ACCS Drawbacks.*

Liveness violation. One property of an ACCS [11] states that *no honest party can end up worse of*, but this is said assuming that the participant's blockchain client doesn't crash. Indeed between stage (4) and stage (5) of the scenario of section 2.1, if we consider that Bob's blockchain client has crashed, Alice can wait for the timelock to expire and retrieve both assets.

This is a violation of the atomicity because Bob is an honest participant. In real world situation, where node crashes and packet loss in common, we cannot admit such assumptions. This is a violation of liveness property [22].

During an ACCS, the participant's blockchain client is a single point of failure and hence an attractive target for the initiator. *In order to overcome this problem we propose that instead of being a single instance, the participant's blockchain clients should be decentralized and replicated trough blockchain adapters.* Indeed, besides the fact that there is less chance of crash, it is far more difficult to attack several instances than a single one. Blockchain Adapters will be discussed in details in Section 3.

Poor ergonomics. A HTLC based ACCS [11] requires each participant to perform at least two transactions on two different blockchains. They must also perform at least one smart contract audit. During this whole process, participants have to stay online and regularly check if the contract's states have changed.

This is very impractical and error prone, both in terms of user experience and in infrastructure, as it requires the participants to run at least two different blockchain clients.

Our approach proposes to automate smart contract auditing trough blockchain relays. Blockchain relays will be discussed in Section 3 but to sum up they are smart contracts on chain A that have "light client like" verification capabilities on chain B. They allow one to verify transaction of chain B from a smart contract on chain A.

Adversarial commerce and out of band communication. Until now, we've only discussed the flaws of an ACCS once the participants have agreed to exchange at a certain rate. Indeed at stage (1) of the protocol presented in Section 2.1, Alice sets Bob's bitcoin address as receiver.

This means two things. First, Alice and Bob have to agree on the exchange and the exchange rate on out of band communication channels. This communication challenge might be a vector of attack. Secondly, once Alice has set up the contract for Bob, at stage (1), Bob can deliberately chose not to continue the protocol and hence,

having Alice's coin be locked until the timelock expires. Besides the fact that it has costs Alice a transaction fees, it might also mean that Alice's locked coin will lose value. Indeed, in an Adversarial environment such as an exchange market, where prices are very volatile and where your capital is your tool, having your funds locked might benefit your adversaries [12].

In this paper, we propose an enhanced version of HTLC contracts that allows participants to withdraw their assets anytime before both contracts are ready to be redeemed. Moreover, this enhanced version of HTLC's will allow for decentralized order matching, which means that Alice and Bob will no longer have to communicate trough out of band channels.

Details of the enhanced HTLC will be discussed in Section 3.

2.2 AC3WN scheme

Several other research teams have made the same observations concerning the flaws of HTLC based ACCS [12] [22].

One paper that has particularly drawn our attention proposes AC3WN (Atomic Cross Chain Commitment Witness Network). This proposal overcomes the shortcomings of HTLC based ACCS by setting up a decentralized network of witness as well as blockchain relays[21]. *Our proposal is inspired by their work, but we improve AC3WN by removing the need for a witness network. To do so we will make use of blockchain adapters.* Our proposal will be discussed in Section 3. In the following we will detail their proposal and explain the improvement we propose.

2.2.1 *AC3WN overview.*

The Atomic Cross Chain Commitment Witness network (or AC3WN), proposed in [21] is a permissionless network of witness that allows to execute atomic cross chain transaction (or AC2T). Their architecture is made out of the following components:

- BCa and BCb are the two blockchains hosting the assets that will be exchanged.
- BCw is the witness blockchain. As a regular blockchain, it stores consecutive blocks that are validated by minors.
- SCw is a contract hosted on BCw. This smart contract can be in three different states : *Published, RedeemAuth, and RefundAuth.*
- $SC1$ and $SC2$ are two smart contracts hosted respectively on BCa and BCb. In those smart contracts, Alice and Bob have committed to exchange their assets and they have linked the Redeem and Refund events of their commitment to the state of SCw

An atomic cross chain swap can be defined as digraph $D = (V, A)$. Each vertex V represents a participant and each arc A represent a proposed asset transfer from the arc's head to its tail via a shared blockchain. [11]

So the scenario of an atomic cross chain transaction using AC3WN is as follows; Alice and Bob are willing to exchange some assets between chain BCa and chain BCb. They construct the digraph D and multisign it generating $ms(D)$ (1). Then one of them registers $ms(D)$ to the smart contract SCw on the witness network (2).

At this stage SCw is in state *Published*. Once the digraph is published, both Alice and Bob publish concurrently their smart contracts $SC1$ and $SC2$ on their respective chains (3). They agree to link the *refund* and *redeem* conditions to the state of SCw. At this point, one of the participants can request a *RedeemAuth* state change to the contract SCw on the witness network(4). Alongside with this request, they provide evidence that their contracts $SC1$ or $SC2$ has been published. Next the smart contract SCw will check if the evidence is valid and if so, it will change its state to *RedeemAuth* (5).

Once SCw is in state *RedeemAuth*, the participant will be able to redeem their funds by providing to $SC1$ or $SC2$ the proof that SCw is in state *RedeemAuth*.

If one participant stops reacting during the swap, then the other party just has to wait for the timelock to expire and can request a *RefundAuth*. Similarly, once the state of SCw is *RefundAuth*, the participant can be refunded by providing to the $SC1$ or $SC2$ the proof that SCw is in state *RefundAuth*.

One very important aspect to understand this protocol is that each "asset blockchain" i.e the chains that host the exchanged assets must have "light client like" verification capabilities over the witness network blockchain.

In parallel, the witness network chain must have "light client like" verification capabilities over all the asset chains. The component (in general a smart contract) that allows chain to have verification capabilities over other chain is called a Relay. This component will be explained with more details in Section 3.

2.2.2 *AC3WN Drawbacks.*

Out of band communication. As well as the ACCS, the AC3WN suffers from the out of band communication problem. Indeed, Alice and Bob have, before stage (1) of the scenario in section 2.2.1, to communicate and build the digraph trough out of band communication channels.

Not only this channel might be used as a vector of attack but it also means that a decentralized order book and order matching engine would not be possible. Indeed, since Alice and Bob have to publish their digraph to SCw before they actually lock their assets, we cannot build a decentralized market from asset chains data.

Trough a modified version of HTLC based ACCS, we can remove the need for out of band communication. Moreover, we can build a decentralized market and matching engine from asset chain's data. Details of our proposal will be discussed in Section 3.

Sidechains complexity. AC3WN requires a witness network blockchain. This can be assimilated to a sidechain. But as stated in [8] sidechains come with several drawbacks: soft forks, risks of mining centralization but the one that we will treat here is complexity. Indeed, sidechains induce complexity and particularly in terms of infrastructure : building and maintaining a sidechains represents high costs.

We remove the need for a witness network sidechain by modifying the HTLC based ACCS protocol. Our proposal also makes use of cross chain verification, but instead of having cross chain verification capabilities of the witness chain, asset chain will have verification capabilities over other asset chains. This will be achieved with relays, further discussed in Section 3.

3 RELAY SWAP

In this section we present Relay Swap, a protocol that makes use of blockchain relays and blockchains adapters to execute an enhanced version of the HTLC based ACCS.

3.1 Building blocks

In this section we'll present the different building blocks of our solution. Then in the next section we will explain in details how those building blocks interact to achieve an ACCS.

3.1.1 *Smart Contracts*.

Smart contracts are programs deployed on a blockchain and executed by mining nodes. They are written in a scripting language such as Solidity for Ethereum [5].

As they exist in the same environment, smart contracts are capable of receiving, storing and sending assets on the blockchain. A smart contract byte code is visible to anyone as well as each function call or any write operation to the smart contract's memory. Any call to a smart contract's function or procedure is signed by the caller's key, and each interaction is permanently logged in the blockchain.

From a more general point of view, smart contracts allows one to automate tasks on the blockchain, tasks that can involve assets, in a distributed and auditable environment.

3.1.2 *Blockchain Relays*.

A blockchain relay is a smart contract hosted on a blockchain *BCa* that stores block headers of a blockchain *BCb*. This smart contract has "light client verification" capabilities of chain *BCb* on chain *BCa*[9].

Relayers are miners that publish block headers of chain *BCb* to the relay smart contract on chain *BCa*. Then the smart contract uses the standard verification procedure for chain *BCb*'s consensus algorithm to verify this block header[13].

Once a block header has been published, verified and buried under a satisfying number of new blocks (to avoid risk of fork), it is possible to verify any transaction of chain *BCb* from *BCa*.

Such implementations have already been launched in production, for example BTCrelay that relays bitcoin blocks on the ethereum blockchain. [10].

It is possible to build a high level interface that hides the complexity of the cryptography verification to developers. A possible interface inspired from BTCrelay is presented in Figure 1. Code available on our github repository [15].

3.1.3 *Blockchain Adapters*.

Blockchain Adapters are stand-alone, self-hosted blockchain APIs that allow to send transactions and query different types of blockchain protocols and networks [2]. They can be seen as an enhanced client that allow to automate certain tasks. For example, in the case of an HTLC based ACCS, an adapter would be responsible for generating a secret, hash it and store it. The adapter would also be responsible for broadcasting transactions automatically if some conditions are fulfilled.

One flaw of the ACCS is that the blockchain client of the participants is a single point of failure. A participant, even if he tries to comply with the protocol, is subject to DDOS attacks or any type of attacks delaying the propagation of a transaction. The blockchain

```
Abstract class Relay{
    String [] blocks

    function storeBlockHeader(String blockHeader) do
        //Verifies and stores a new block header.
        //This function is for relayers.
        requires verifyBlock(blockHeader)
        this.blocks.append(blockHeader)
    end

    function verifyTransaction( string rawTransaction,
                                int transactionIndex,
                                bytes merkleSibling,
                                bytes blockHash) do
        /*
        Verifies the presence of a transaction on the relayed blockchain,
        primarily that the transaction is on main chain
        and is buried under a sufficient amount of new blocks
        */
        if the transaction is valid
            return true
        else
            return false
        end if
    end

    function verifyBlock(blockHeader) do
        //Computes validity of the block with standard verification procedure
        if the block is valid
            return true
        else
            return false
        end if
    end
}
```

Figure 1: Blockchain Relay Smart contract

adapters provide a solution to this flaw because it can be decentralized and distributed over several nodes. Recent projects have shown the ability to store private information in decentralized environment in order to query protected services. For example, Chainlink has achieved to query Paypal's API and execute Paypal transactions from a blockchain adapter [6] [4].

In Figure 2, we present the interface of a blockchain adapter that could achieve Relay Swap protocol. This is a minimal interface, for full algorithm please refer to our github repository [15].

3.1.4 *Atomic Swap Smart Contract*.

The relay swap protocol uses a smart contract inspired by standard HTLC contracts. This contract is used to store swap's parameters as well as swap's states. The data structure of a swap is equivalent to the digraph $D = (V, A)$ [11].

The contract can be in five different states : *Invalid, PreCommited, Commited, Redeemed,* and *Refunded*. For each state (except *Invalid*) there is a function that triggers state changes and execute the corresponding actions. The function also verifies that the requested state change is legal i.e that the atomicity of the swap is not violated. To do those verification, the smart contracts relies both on its memory and on calls to the Relay contract. Indeed, as they are hosted on the same chain, the Atomic Swap contract and the Relay contract can communicate. This is a key element to understand the relay swap protocol : given that the Relay hosted on *BCa* can verify the state of an Atomic Swap contract *SC*2 on chain *BCb* and given that an Atomic Swap contract *SC*1 hosted on *BCa* can communicate with the Relay on *BCa*, we can set the conditions of state changes of *SC*1 to *SC*2's state and vice versa.

In Figure 3, we present a minimal interface of an Relay Swap Smart contract (see more details in github repository [15]).

```
class Adapter{
    struct Swap{
        bytes secret, bytes hashedSecret, address receiver, int value, float exchangeRatio
    }
    enum States {INVALID, PRECOMMITED, COMMITED, REDEEMED, REFUNDED}
    function preCommit(Swap swap) do
        //If swap's hashed secret == null then
            //Generate random secret
            //Compute SHA256(secret)
        //Call precommit function of our HTLC
        //Store the swap
        //Set swap's state to PRECOMMITED
        //If the swap is already in preCommit state on the other side then
            //Call reqCommit function of the other party's HTLC
            //Call commit function  of our HTLC
            //Set swap's state to COMMITED
            //Run cronJobParticipant(swap)
        //Else
            //Run cronJobInitiator(swap)
    end
    function refund(Swap swap) do
        //Call refund function of our HTLC
        //Set swap's state to REFUNDED
    end
    function cronJobParticipant(Swap swap) do
        //While the swap is not redeemed nor refunded
            //For every new incoming and validated new blocks
                //If the secret key has been revelated then
                    //Call the redeem function of our HTLC with secret
                    //Set swap's state to REDEEMED
                //If the timelock has expired then
                    //Call the refund function of our HTLC
                    //Set swap's state to REFUNDED
    end
    function cronJobInitiator(Swap swap) do
        //While the swap is not redeemed or refunded
            //For every new incoming and validated new blocks
                //If the other party's HTLC is in COMMITED state then
                    //If timelock < now
                        //Call redeem function of the other party's HTLC
                    //Else if timelock > now
                        //Call refund function of our HTLC
    end
    function reqPreCommit(Transaction tx) do
        //Call data = audit(tx)
        //data.exchangeRatio = 1/data.exchangeRatio
        //Call preCommit(data)
    end
    function audit(Transaction tx) do
        //Calls the relay's verifyTransaction function
        //Decodes the raw TX
        //Returns an object containing tx data
    end
}
```

Figure 2: Adapter

```
class AtomicSwap{
    struct Swap{
        int timelock, int value, address sender, address receiver,
        bytes secret,bytes hashedSecret
    }
    enum States {INVALID, PRECOMMITED,COMMITED,REDEEMED,REFUNDED}

    public function preCommit(Swap swap) do
        //Set swap's timelock to null
        //Set swap's receiver's to function caller's address
        //Store the swap
        //Set swap's state to PRECOMMITED;
        //return true
    end
    public function reqCommit(Transaction tx) do
        //Call relay to audit given tx
        //If transaction valid and burried under sufficiant then
            //If transaction's value, receiver, secret and timelock correspond then
                //Call the commit function with parameters decoded from the transaction tx
    end
    internal function commit (Swap swap, address receiver) do
        //Checks swap is in state PRECOMMITED
        //Set swap's receiver to receiver address
        //Set timelock to now + timeDelta
        //Set swap's state to COMMITED
    end
    public function commit (Swap swap, address receiver) do
        //Checks that swap is in state PRECOMMITED
        //Checks that caller is swap's sender
        //Set swap's receiver to receiver address
        //Set timelock to now + (timeDelta/2)
        //Set swap's state to COMMITED
    end
    public function redeem(Swap swap, bytes secret) do
        //Checks that swap is in state COMMITED
        //Checks that caller is swap's sender
        //Checks that timelock < now
        //Checks that SHA256(secret) == swap's hashed secret
        //Set swap's secret to secret
        //Set swap's state to REDEEMED
        //Transfer swap's value to receiver
    end
    public function refund(Swap swap) do
        //Checks that swap is in state OPEN
        //Checks that caller is swap's sender
        //Checks that timelock > now
        //Set swap's state to EXPIRED
        //Transfer swap's value to sender
    end
    //Returns the state of the provided swap ID
    public function checkState(Swap swap)
    //Returns the timelock of the provided swap ID
    puublic function checkTimeLock(Swap swap)
    //Returns the secret of the provided swap ID if it has been revelated
    public function checkSecretKey(Swap swap)
}
```

Figure 3: Relay Swap Smart Contract

3.2 Relay Swap protocol

The full Relay swap protocol, whatever the output is, consists of three state changes that triggers asset movement or a state change in another contract.

The first state change from *Invalid* to *PreCommit* is triggered by the initiator as shown in Figure 4.

As we can see, the initiator, Alice requests a pre-commit to her adapter. She chooses the parameters of the swap (value and exchange ratio) but the secret and its hash will be generated and stored by the adapter. Then the adapter calls the $SC1$ to pre-commit.

Once the swap is in *PreCommit* state, it will appear on a decentralized order book.

Bob, the participant will retrieve the parameters of the swap from the decentralized order book and will, in turn, request a state change to its adapter. Then the adapter will call a state change on $SC2$ and will return the transaction hash to the adapter. Once receiving the transaction hash Bob's adapter will request a commit to Alice's swap contract $SC1$. It is at this moment that we achieve cross chain communication, indeed, $SC1$ will verify, thanks to the adapter, that Bob has indeed set up a smart contract $SC2$ with corresponding parameters. This whole process is described in Figure 5. On receiving the confirmation that Alice's smart contract $SC1$ is

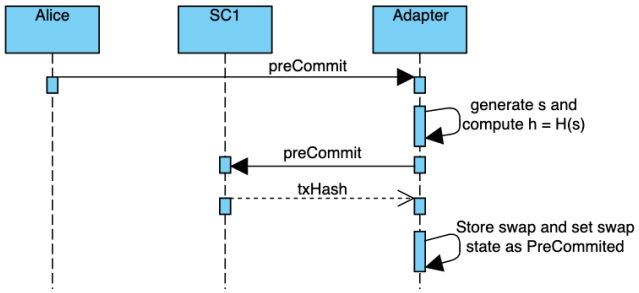

Figure 4: Pre Commit State Change

in *Commit* state, Bob's adapter will trigger the commit event on its own contract $SC2$.

As you can see at the end of each process presented in Figure 4 and Figure 5, a cron job has been launched. Indeed, once the *PreCommit* state has been reached for the initiator resp. *Commit* for the participant, the only way to know if a new state change has been done is to check the contracts at every new valid block relayed by the relay.

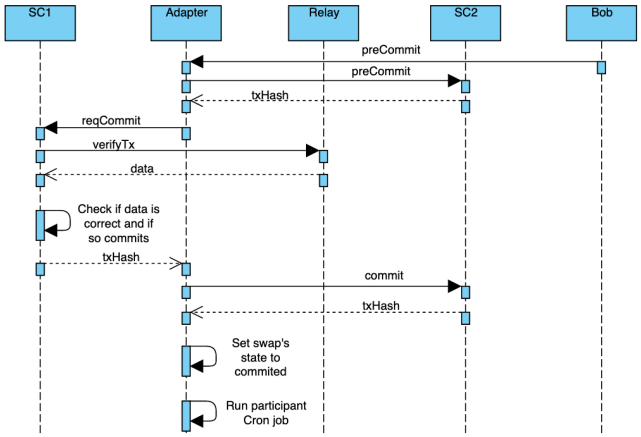

Figure 5: Pre Commit and Commit State Change

So, for every new block the initiator's cron job will check *SC2* state, and if the state is *Commited*, then it will request a redeem.

When calling the redeem function, the adapter will have to provide the secret it has generated at the beginning of the swap. This will trigger Bob's (the participant) cron job. See Figure 7.

If the state is not *Commited* then it will check if the timelock is expired and if so request a refund. See Figure 6.

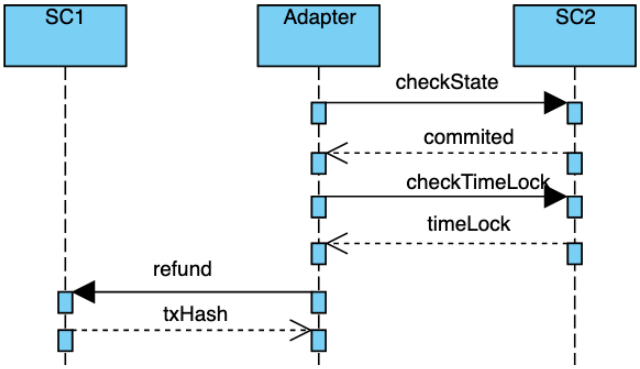

Figure 6: Initiator Cron Job refund

Bob's (the participant) cron job does similar actions but instead of checking for state, it is checking if the secret key has been revealed. See Figure 7. If the secret key is never revealed, then Bob's cron job will request a refund similarly as in Figure 6.

4 CONCLUSION

While HTLC based atomic swaps is a promising protocol to achieve inter-chain exchanges, it comes with some drawbacks when it comes to real world applications. Indeed, in an environment where packet loss, client crashes and network failure is common, the atomicity of the ACCS protocol cannot be guaranteed. In this paper, we presented Relay Swap which uses blockchain adapters. Blockchain adapters remove the need for a witness chain used by AC3WN [21] and hence reduces the complexity of the protocol. Moreover, our protocol allows for decentralized order book.

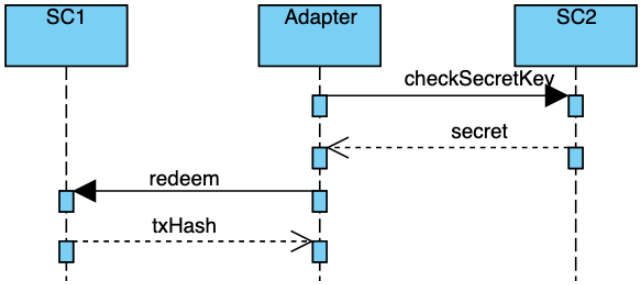

Figure 7: Participant Cron Job redeem

ACKNOWLEDGMENTS

We thank Viktor Zakhary for detailing AC3WN functioning.

REFERENCES

[1] [n.d.]. Argent – The best Ethereum wallet for DeFi. https://www.argent.xyz/
[2] [n.d.]. Blockchain Adapters. https://support.unchain.io/category/3-blockchain-adapters
[3] [n.d.]. The Monolith DeFi account. https://monolith.xyz/
[4] [n.d.]. PayPal External Adapter. https://chainlinkadapters.com/details/smartcontractkit/paypal-adapter
[5] [n.d.]. Solidity. https://solidity.readthedocs.io/en/v0.6.8/
[6] 2020. chainlink external adapters explained. https://blog.chain.link/chainlink-external-adapters-explained/
[7] 2020. the largest cryptocurrency hacks so far. https://www.investopedia.com/news/largest-cryptocurrency-hacks-so-far-year/
[8] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. [n.d.]. Enabling blockchain innovations with pegged sidechains. ([n. d.]).
[9] Vitalik Buterin. 2016. Chain interoperability. *R3 Research Paper* (2016).
[10] Ethereum. 2017. ethereum/btcrelay. https://github.com/ethereum/btcrelay
[11] Maurice Herlihy. 2018. Atomic Cross-Chain Swaps. arXiv:cs.DC/1801.09515
[12] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. 2019. Cross-chain Deals and Adversarial Commerce. arXiv:cs.DC/1905.09743
[13] Aggelos Kiayias and Dionysis Zindros. 2019. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*. Springer, 21–34.
[14] Jae Kwon and Ethan Buchman. 2016. Cosmos: a network of distributed ledgers (2016). *URL https://cosmos. network/whitepaper* (2016).
[15] Léonard Lys. 2020. Two Phase Commit Atomic Cross Chain Swap. https://github.com/leoloco/Two-Phase-Commit-Atomic-Cross-Chain-Swap
[16] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008).
[17] Helen Partz. 2019. Few Big Exchanges Continue to Report Fake Volumes in 2019: Chainalysis. https://cointelegraph.com/news/few-big-exchanges-continue-to-report-fake-volumes-in-2019-chainalysis
[18] Komodo Platform. 2018. Komodo, An Advanced Blockchain Technology, Focused on Freedom. https://static2.coinpaprika.com/storage/cdn/whitepapers/140811.pdf
[19] Gavin Wood. [n.d.]. Polkadot: Vision for a heterogeneous multi-chain framework. ([n. d.]).
[20] Jiahua Xu and Benjamin Livshits. 2018. The Anatomy of a Cryptocurrency Pump-and-Dump Scheme. arXiv:q-fin.TR/1811.10109
[21] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. 2019. Atomic commitment across blockchains. *arXiv preprint arXiv:1905.02847* (2019).
[22] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. 2019. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 193–210.