Unlocking Hash Time Locked deposits on Smart contract chains with XMR payment

https://github.com/StrawberryChocolateFudge

April 2025

Abstract

Hash time lock contracts are traditional methods to implement atomic swaps between smart contract chains or chains with scripting capabilities. Monero is not one of these chains and because of this traditional HTLC Atomic swaps are not possible and other solutions are developed in the ecosystem for it.

This paper proposes to use XMR to unlock hash locked deposits on smart contract chains with a workaround that allows trading XMR to ETH in a trustless DEX. The implementation is not an Atomic Swap but it is a method of trading XMR on a Decentralized Exchange with existing non-custodial wallets.

1 Background

There are currently 3 ways to trade XMR, a Multisig Escrow (Haveno), Atomic Swap implementations or a centralized exchange. They each have different benefits and trade offs and different use-cases.

A multisig escrow requires a desktop wallet, and 3 parties to make a trade, a maker, taker and arbitrator. This method of trading is most efficient when exchanging to cash or bank transfer and serves an important niche. The downsides are user-experience, speed and has a medium barrier to entry.

Atomic Swaps are a way to exchange 2 blockchain currencies with each other in a trustless way that makes the exchange atomic, it either succeeds or it's automatically refunded. It has an important place in the ecosystem as trustless swaps. The downsides are implementation complexity, slow speed and high barrier to entry.

Centralized Exchanges are a way to exchange XMR with a central exchange entity that holds custody of the funds and will make trades in the database without making on-chain transactions. The downsides are KYC requirements, delisting, withdraw suspensions, frozen balances and trust issues but nonetheless it's an important part of the ecosystem because it provides good UX in the browser and new user onboarding.

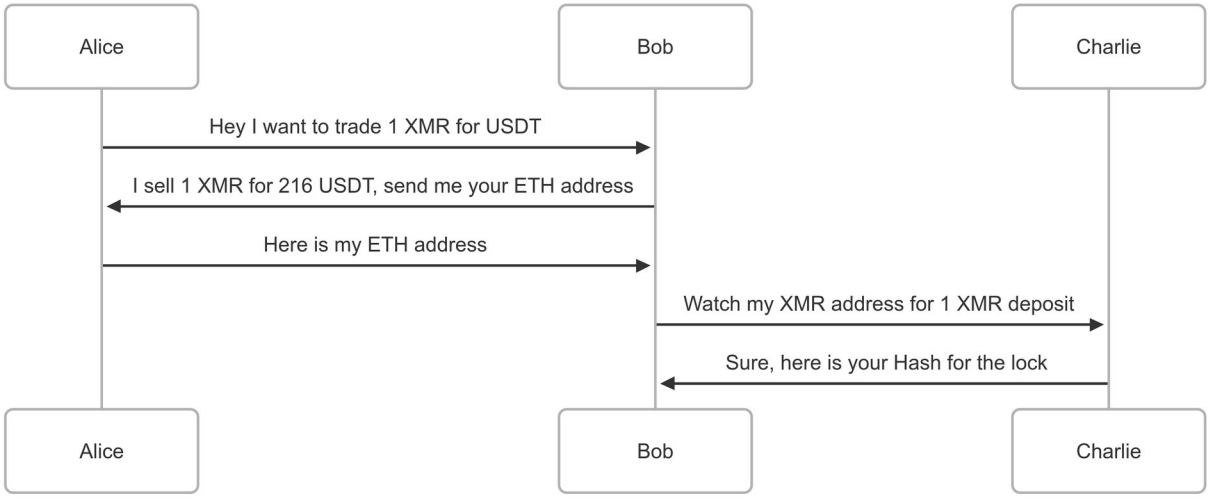
The proposed option outlined in this paper is a middle ground between the above 3 solutions where the trading happens on-chain between 2 blockchains, it's non-custodial and it sits somewhere between the multisig escrow and atomic swaps and the UI works in the browser. It requires an infrastructure of nodes that watch the monero blockchain with view only wallets and after observing valid transactions they unlock hash locks.
The payments are always P2P transferred between 2 parties, the xmr is directly sent while the other currency that is traded for is behind a Hash and time lock.

2. **Trading flow**

The untrusted parties will be referred as:
Alice -- The party that sends the XMR
Bob -- The party that sends the ETH
Charlie – The commitment service which is an untrusted node participating for an incentive

So Alice and Bob they want to trade.
Alice sends XMR and Bob will send ETH.

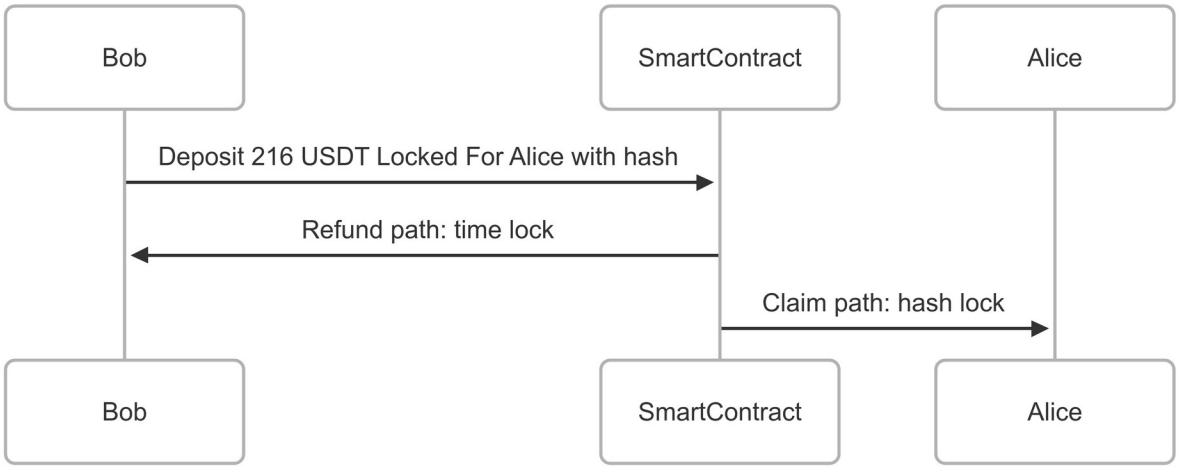
The service that observes payments made to Bob is Charlie. Charlie can be a decentralized network made up of N nodes, but for simplicity there is a single one for now.



Alice and Bob agree on the Rate of the trade and Alice sends her address to Bob.

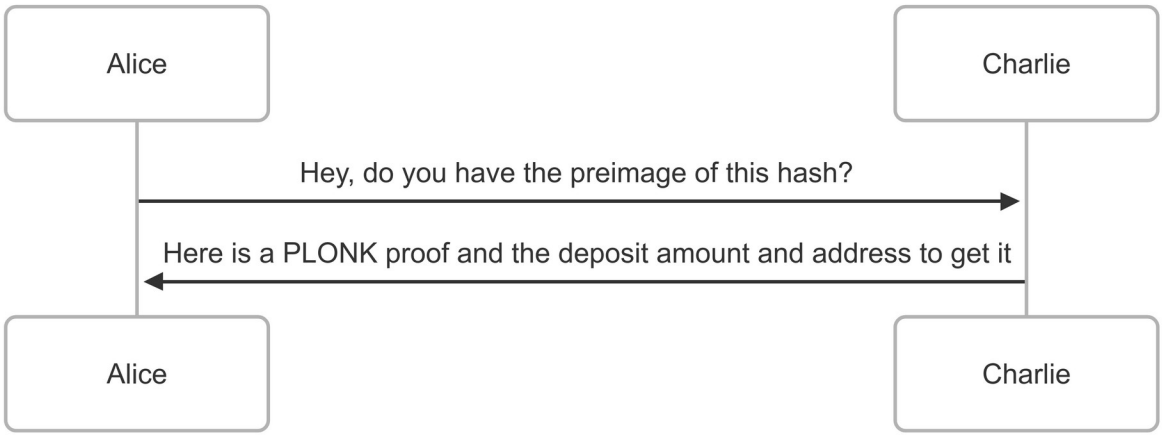
Now Bob contacts Charlie and asks Charlie to give him a hash.
Bob needs to tell his secret view key to Charlie so he can create a view only wallet to watch for transactions.
Charlie will tell Alice the pre-image of the hash if Alice makes a successful payment to Bob.

Bob now takes the hash from Charlie and deposits into a Hash Time Locked Contract. The contracts have the following withdraw paths:



1. Unlock via hash lock which is only available to Alice's address
2. Unlock via time lock which is only available to Bob, as a refund mechanism

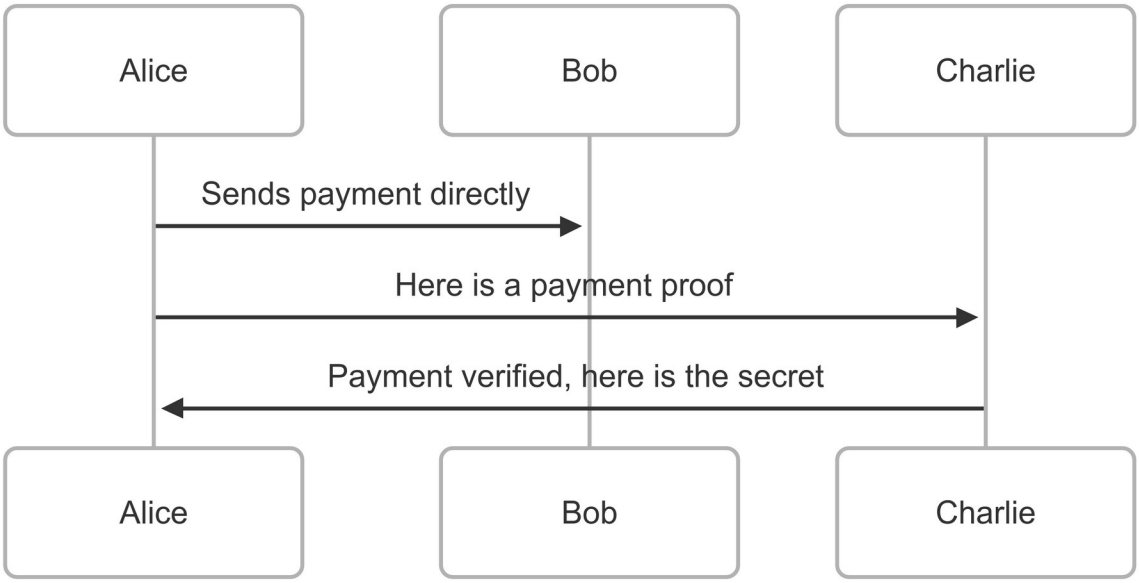
Alice now contacts Charlie and verifies that Charlie possesses the pre-image of the hash stored on-chain. The verification uses a Zero-knowledge proof for proving to Alice that Charlie does possess the hash pre-image and Alice confirms the deposit address and amount with Charlie.



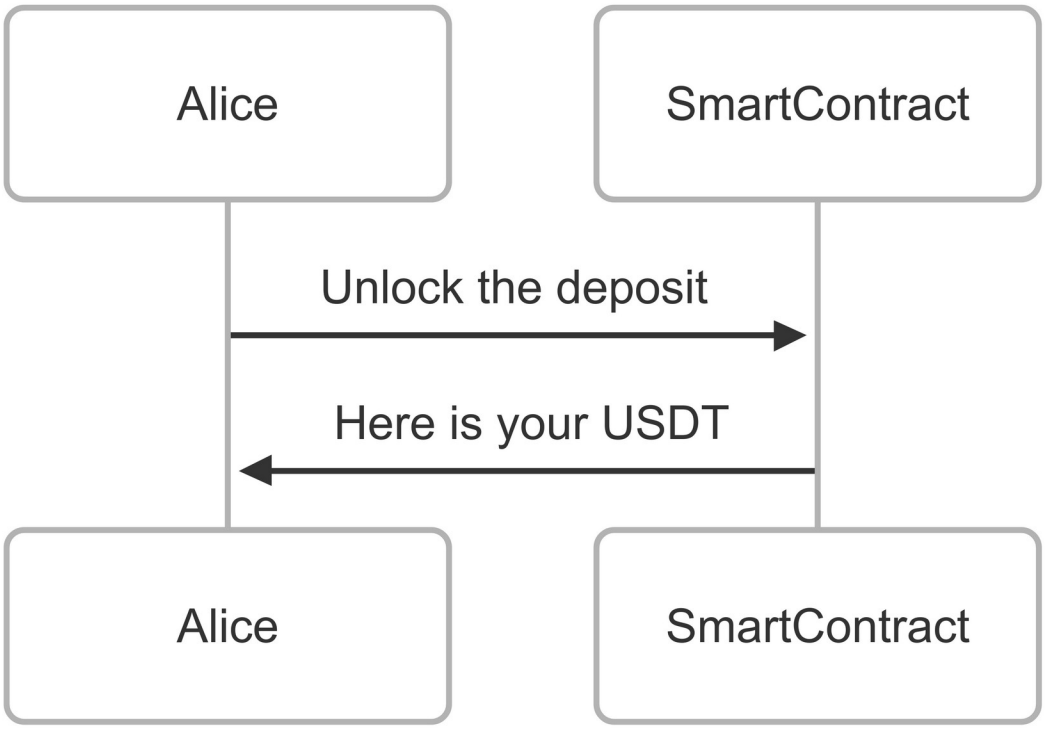
Now Alice makes a payment to Bob's address after verifying that all the details are correct.

After the payment is made, Alice contacts Charlie to redeem the hash pre-image. It requires a Payment Proof containing the TXID TXKEY ADDRESS

and Charlie will run check_tx_key RPC call and verify the the payment while checking the view only wallet to make sure the payment is spendable.

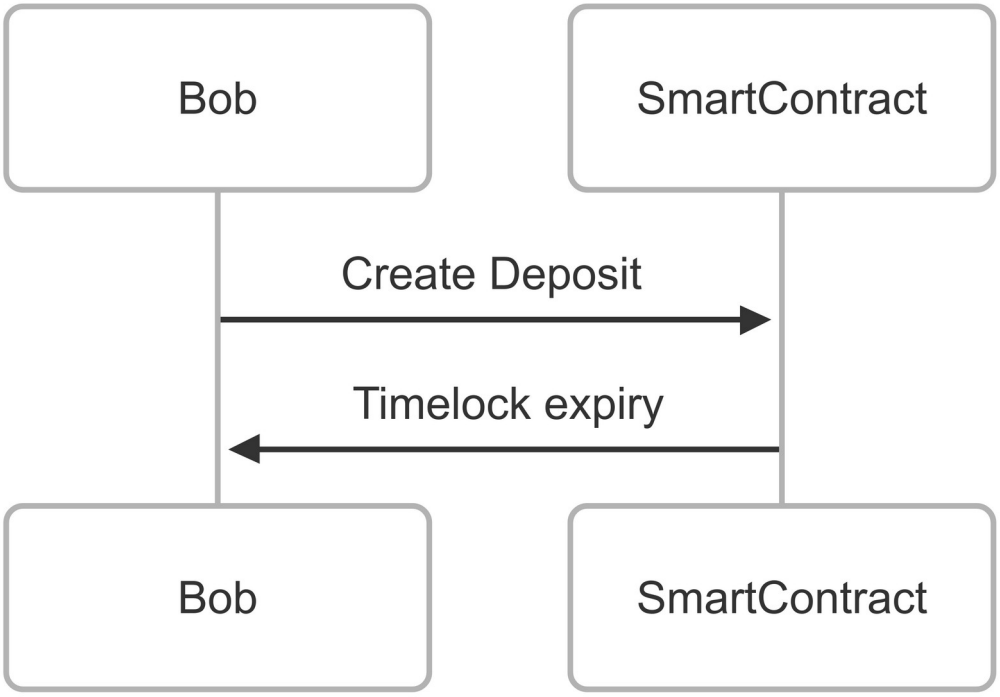


Alice now uses the pre-image(secret) for the hash unlocks and withdraws the deposit.
The trade is complete.

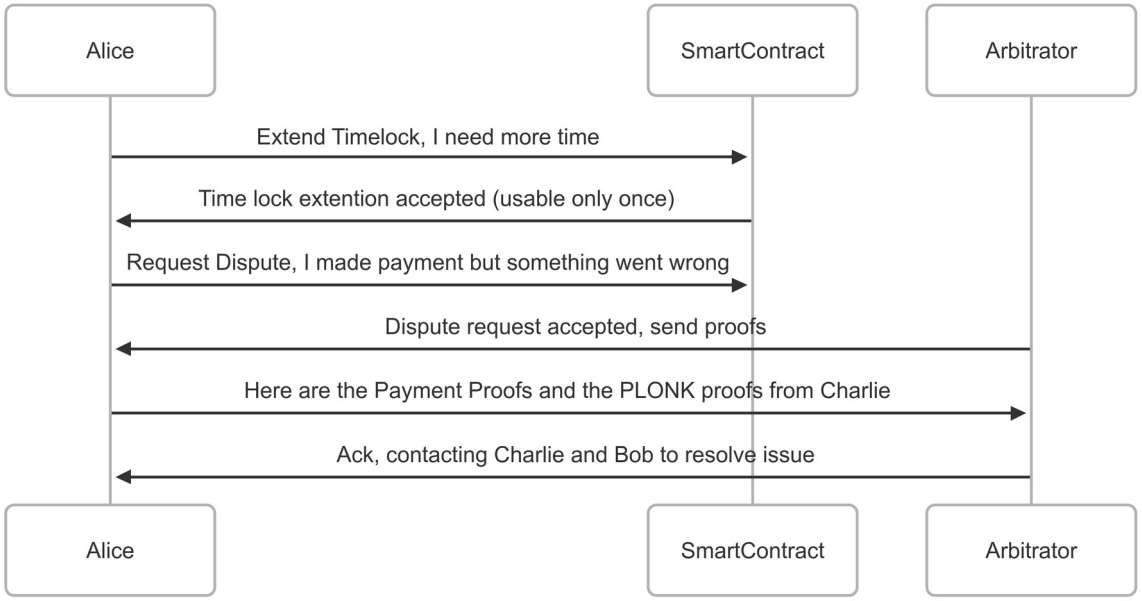


Unhappy Paths
Given that things can go wrong, there are a few things to consider

- Bob makes a deposit but Alice doesn't make XMR payment, in this case Bob is refunded via the expiring timelock.

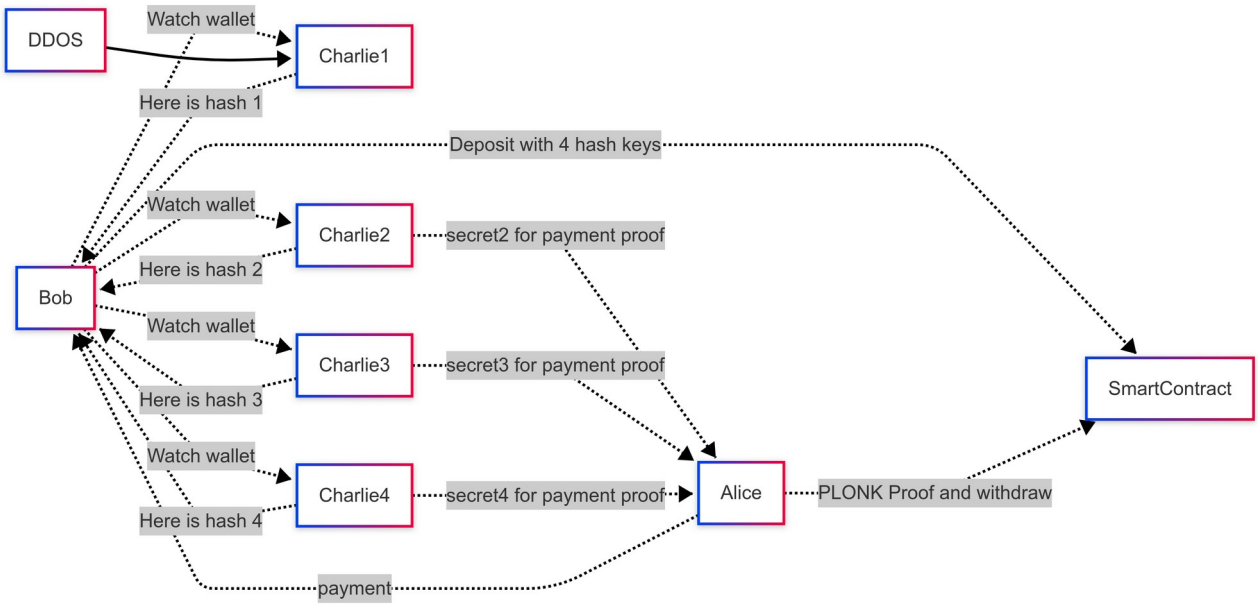


- Alice makes a payment but transfers insufficient funds, in this case a extendTimelock() function should be available for Alice to use, to give time for Alice to make a payment again with the missing funds.



- Because this is not an Atomic Swap, insufficient transfers are not automatically refunded and the deposit risks time lock expiry. If Alice is incapable of transferring the remaining amount to unlock the deposit, the trade fails. Risk can be averted by implementing escrow like mechanism inside a deposit locking contract that would allow an arbitrator to participate in deciding the reimbursing process as a last resort.

- Charlie is under heavy ddos attack
  The system mitigates issues with Charlie with decentralization and multi-hash locks. If Charlie is N amount of services, if a single one is taken down via heavy ddos the remaining ones can still participate in the hash lock.
  A multi-hash lock offers similarities with normal hash locks but it uses N of M amount of hashes to open the lock. Similar to threshold multisig, these are threshold hash-locks.



- The same logic applies if Charlie and Alice collude to steal locked deposits from Bob, with enough parties participating, this attack can be mitigated

- Charlie's server is compromised and the secrets are leaked. Because the hash lock can be only used by Alice it doesn't matter if it leaks to other parties, they will be unable to access funds with it. The purpose of Charlie is to hide the secret from Alice and then reveal it when Alice makes a payment. If Alice hacks Charlie, this issue is mitigated again by running multiple instances and a hacker would need to compromise N amount of servers to do a successful attack.

- Further ddos mitigation: security deposit or Api key with limit. Charlie should scale to create thousands of view only wallets and an attacker could attempt to overwhelm the backend by creating wallets that are not used at all. To mitigate this issue, either a security deposit or a rate limited API key is used that can be purchased. In a DEX, the security deposit is the more probable path.

Technical implementation

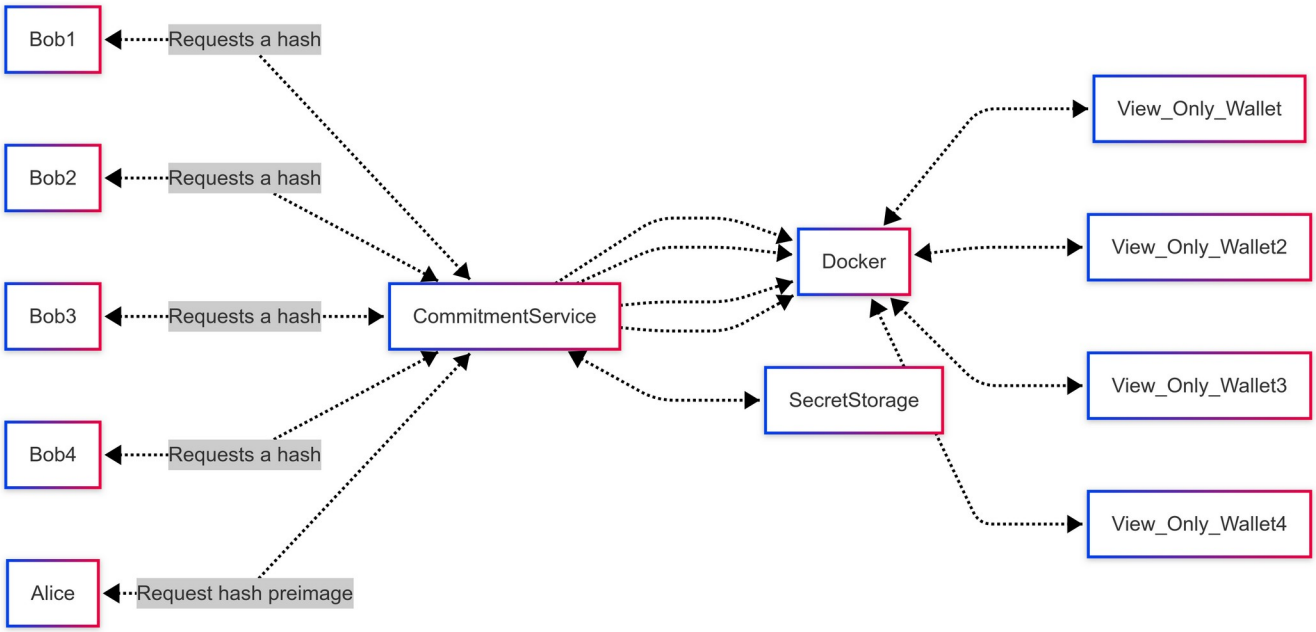
The traders use monero wallet gui and metamask. It assumes the users already possess the required wallets. It doesn't have to implement wallet

functionality at all and will just make rpc calls to monero-wallet-rpc to interact with monero blockchain in the background at the service level. Users are expected to interact with Ethereum smart contracts manually.

 # Commitment Service (Charlie in the above situation)

This backend service has multiple purposes:

- Provision view only wallets using docker
- Watches transactions made to view only wallets and verifies payment proofs
- Store and manage secrets used for the hashes
- Reveal the secrets for valid payment proofs
- Provide a ZkSnark proof to prove the possession of the hash secrets (pre-image). PLONK could be used for verification because it requires no phase-2 trusted setup.
- Provide an html interface and a Json Api so it’s easy to build into services
- It should be universal and easy to use for multiple exchanges



 # Smart Contracts

The smart contracts in this case are Solidity contracts on Ethereum. The protocol would support other languages too, Rust, Move, FunC even Bitcoin

Scripts using Tapscripts but the implementation is a little different for each. For now, the focus is Solidity.

The smart contracts:

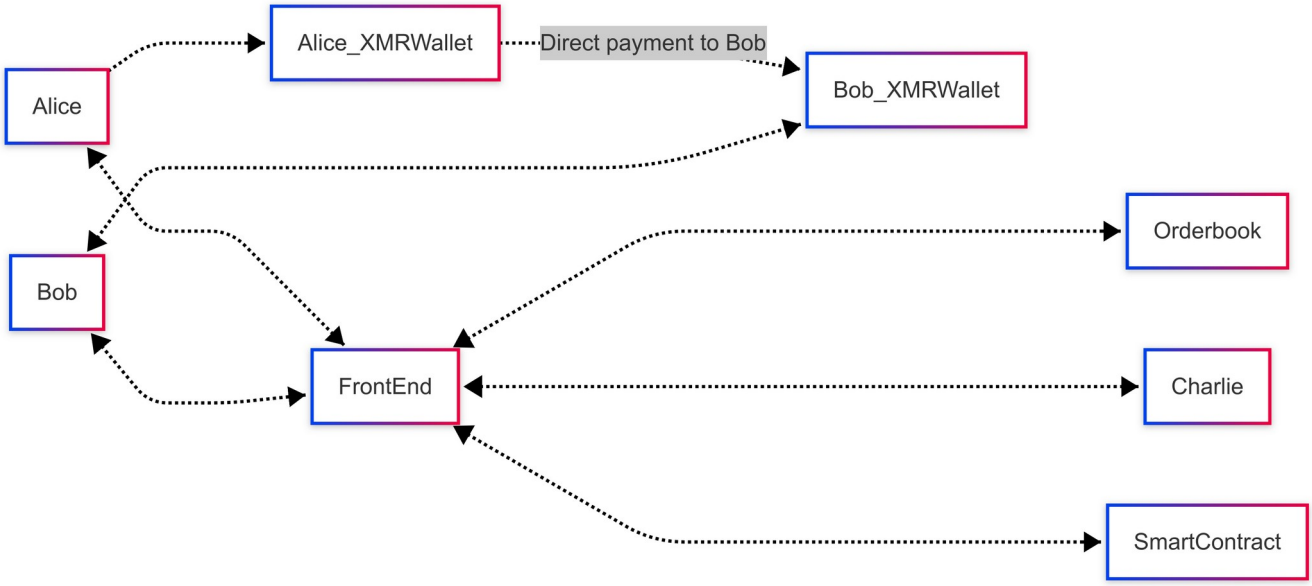
- Should support ETH and ERC-20 tokens
- Implement an interface that describes deposit, claim and a refund functions
- The deposit path should be called by Bob with a deposit and hash lock parameters
- The claim path should be only available to Alice
- The refund path should be only available to Bob
- For scalability, the claim path should verify a PLONK proof for withdrawing, that would allow it to scale to large amount of hash locks without running out of gas so the verification scales with the amount of commitment services available and allows for account abstraction later too.
- Should provide incentive to run commitment services by distributing fees to providers of valid hashes, when the claim function is called.
- It should allow for extending the timelock for a period given a failed payment
- It can optionally offer an escrow fallback for when insufficient xmr payments were made.

 # Front end

The front end should be available in the browser and it should be an SPA, that allows using the application in the browser and provide a similar UX to existing DeFi applications on Ethereum. It should allow existing Ethereum users to buy XMR as comfortably as possible.

- React SPA with a IPFS/centrally hosted front end
- Familiar UI displaying Chart and Maker/Taker positions
- A wallet signature required to advertise trading intent
- Verify PLONK proofs used when validating the commitments services knowledge of secrets

- Compute PLONK proofs when claiming deposits. Using a zksnark when pulling payment also allows to introduce a relayer later that can withdraw on behalf of the user, if they don’t have ETH balance for gas.



 # Order book backend

The order book is a separate backend given that it can’t be stored on-chain for free. Currently the proposal is to use a centralized webrtc server that will store the orders and match orders to prompt users to trade. It’s possible to use LibP2P with WebRTC transport for this to decentralize the order book.

The trading flow requires the users to interact with each other and accept the trades manually by signing transcitions. Solutions to create ***Automated Market Makers*** *can be added later, but that would require running local desktop client with built in wallet to sign transactions.*

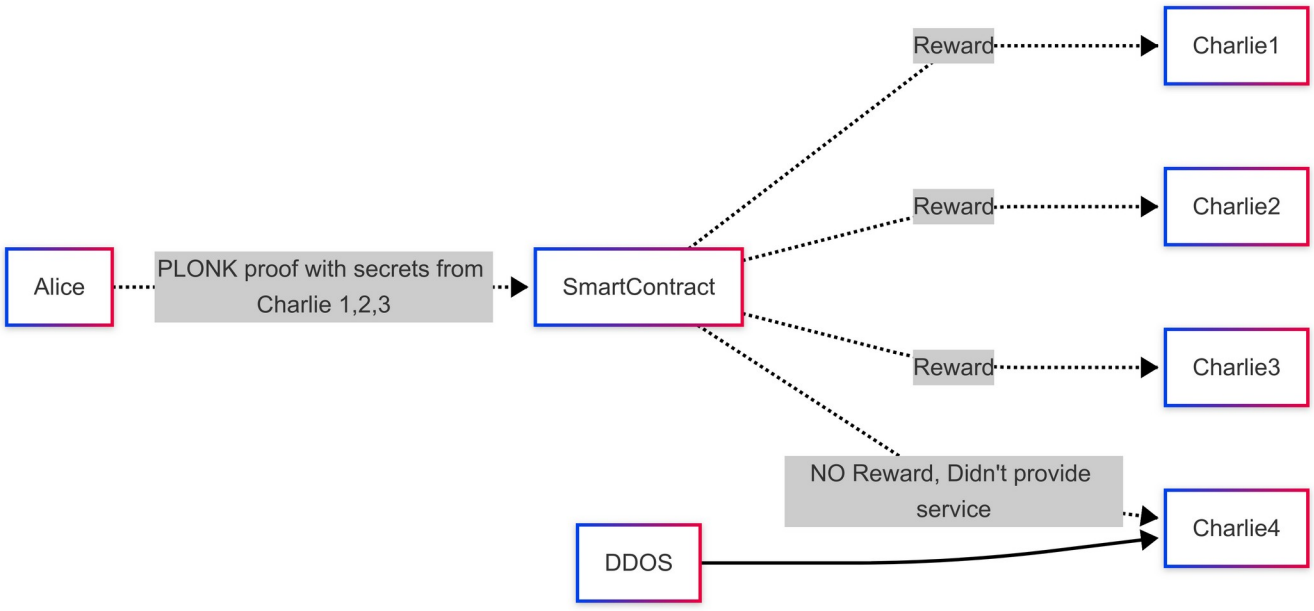
Zero-knowledge proofs

There is a need for 2 zkp. Once the Commitment Service needs to prove to the monero sender that they possess the pre-image of a hash and that the pre-image is related in some way to the address to deposit to and the amount to send, so they are all verifiable. This proof could be later used for slashing the security deposit of a commitment service if it misbehaves.

The other proof is for unlocking the hash lock. This allows for a scalable multi-part hashlock that could scale to hundreds of locks (but that would be overkill)
The chosen proving system is PLONK on bn254 curve. It requires no phase-2 trusted setup and the ptau files for the phase-1 trusted setup can be used from another ceremony (Polygon Hermez ceremony has phase-1 trusted setup files already)
PLONK can be verified on-chain and in the client effectively and the verification constraints can be defined in Circom.

Incentives

For Commitment Services (Charlie) to participate in the protocol a smart contract level reward system can be implemented that would allow distribution of a protocol fee (exchange fee) to the services that participated in unlocking watching wallets and unlocking hashlocks.



Penalties

The commitment service (Charlie) is an untrusted party with economic interest to participate in the protocol. Charlie is expected to reliably watch wallets and provide a service. If there is a threshold of hashes needed to unlock a deposit and too many hashes are missing, the option to penalize misbehaving commitment services could be available. Penalizing the commitment service would be possible if:

- They make a security deposit to join so there is something to slash
- They provided a PLONK proof of knowledge to Alice about their knowledge of the secret before Alice makes the payments
- They were unable to reveal the hash secret
- Other commitment services were able to verify the payment and reveal the hash secret but didn't reach high enough threshold.
- An arbitrator was invoked to resolve the problem

  Then the Arbitrator could use the PLONK proof given to Alice by Charlie to slash Charlie's deposit, to reimburse the party that can't pull payment because of Charlie's downtime.

In case a Commitment service just reveal secrets without verifying the monero payments, a penalty could be given after timelock expiry.

This is still experimental and more thought required into how penalties should work.

Limitations

The approach is the mix of escrow vs atomic swaps. The options to refund directly sent transactions are limited without implementing a specialized wallet for it, so falling back to escrow on the ETH side as a final absolute last resort is the only solution for now for a refund.

51% attack

If more than half of the commitment services are malicious they can collude with Alice to drain a deposit. This kind of attack when the network is overtaken by malicious nodes exists in every blockchain network. The attack

surface however is low and not all deposits would be effected, if the commitment services used for the locks are matched dynamically and have no knowledge about each other, then when the protocol participation is sufficiently large it can be mitigated.

Griefing attacks
The protocol requires the ETH holder to move first, so the implementation is vulnerable to Griefing attack. When the attacker initiates trades but never transfers XMR, the ETH will be refunded but the gas fees will be consumed. This could potentially result in abuse. The only solution for this currently is a private chat using the order book server where the traders could send each other messages and discuss the trades. The application could provide encrypted chat and rank traders.

Denial of service attacks

The commitment services are vulnerable to targeted ddos that could take them down for a sufficient amount of time to let the time lock expire. The only solution to this is decentralization and using multiple services with a threshold of N required to unlock a hash lock.

The owners of the view only wallets would need to do a security deposit, for sybil resistance.

Privacy concerns
While it can be used from behind Tor, the front end requires javascript to run locally and it’s not designed to run from Tor browser. The trading flow requires that one of the parties publishes their secret view key to multiple untrusted services that will gain insight into it’s transaction and trading history and the IP address will be shared when connecting to these servers. So essentially the trading history of users will be public.
Using VPN and Tor can hide the IP.

Why Hash locks and not multisig?

It’s for simplicity and removing the reliance on private keys. If multisig is used for XMR deposits, that requires a desktop client and will make the implementation more complex and requires multisig messaging.
If multisig is used on the Ethereum side, (using signatures instead of hash lock) there are issues with scaling, storing private keys and it’s not compatible with all chains.
For good measure, hash locks with sha256 and poseidon hash should be sufficient for all future implementations. Poseidon is zk compatible and sha256 is backwards compatible with all chains where only scripting is available.

Extending the protocol with a custom XMR wallet client
If the trading parties are comfortable downloading a custom wallet, more features can be added using monero multisig for payment address instead of direct payment, like conditional refunds which would allow creating a better failed payment refund , monero collateralized loans by implementing a liquidator service with multisig or to implement liquidity pool.
Custom wallet would also be needed for an Automated Market Maker because otherwise the payments and smart contract interaction relies on manual sending.