MELIORA

Decentralized, Trustless and Anonymous Swaps

WHITEPAPER

Table of Contents

1. INTRODUCTION

Crypto market continues to experience rapid growth and ever since Bitcoin birth many new successful projects were developed like Ethereum, Ripple, Monero, Zcash, Uniswap, EOS, stable coins and new ones are started every day. The first aim of digital blockchain assets and cryptocurrencies is to achieve a truly open and free financial system accessible for everyone. This system will fulfill the idea of having programmable money to help people exchange assets digitally over the blockchain.

The first major attempt was made by the centralized exchanges. The problem here is that they use the blockchain technology but keep the well-known trusted third-party model where users must give up the custody of their own assets to third-party providers such as Bifinex or Coinbase. When you register on a CEX, you must fully trust these systems not to get hacked, expose your assets to risk.

Further steps in achieving an open financial system were taken by the Decentralized Exchanges (DEXs). Unlike the centralized platforms above, the DEXs operate in a decentralized way, meaning that users keep the control of your own assets; you do not provide personal information or perform any KYC/AML procedures; the trading is happening peer-to-peer without any interference of third party. Although, theoretically DEXs sound flawless, they have some bottlenecks as well. Decentralized exchange platforms can only do single chain swaps such as ETH Token-to-ETH Token Transactions.

However, what happens if you want to exchange a DOT token for a ERC20 token? Here comes the latest innovation in Decentralized Finance - cross-chain transactions executed by Atomic Swaps without any intermediaries. We are at Meliora creating Decentralized P2P Cross-chain Swap Governance Protocol which will become an ultimate solution for cross-chain crypto trading solving most problems by providing flexibility in using our protocol, which other trading platforms will be able to integrate and their users will be handed full shared ownership over their crypto assets. Around it we will build Meliora ecosystem that ensures trust, security and transparency which will greatly boost crypto adoption.

2. Meliora Vision

At present, blockchain assets are trapped in their own systems without inter-chain division of work and collaboration. BTC has established the most widely accepted consensus, but its transaction efficiency is low; ZEC has enabled privacy protection, but smart contract is not incorporated; ETH has smart contract, but it cannot migrate to the PoS system. In terms of trading those digital assets, traders should use CEXs and they can trade them against BTC or stable coins. Usage of CEXs brings too many risks and problems together. Moreover, CEXs are against the decentralized nature of blockchain. On the other hand, DEXs are currently only operating on a single chain which prevents traders from exchanging their different blockchain assets against another chain assets. At Meliora, we envision a fully decentralized future where chains are interoperable, and traders can swap all kind of digital assets against each other. Meliora aims to build a multi-chain cross swap protocol that will truly decentralize how people trade.

3. Meliora Mission

The blockchain industry, originating from Bitcoin, is essentially part of the financial industry, and the exchange serves as its fundamental infrastructure providing functions of liquidity and value discovery. After more than ten years of development, the blockchain industry has embraced explosive growth in both the type and quantity of digital assets: from Bitcoin in the beginning to tens of thousands of digital assets including digital currencies, stable coins, and application tokens. This exponential growth of cryptocurrencies makes the need of cross chain swap more vital day by day. At Meliora, we aim to eliminate the current risks and problems that DEXs and CEXs create. Meliora will be one stop trading solution for multichain swaps. Meliora will support Bitcoin, Ethereum, Binance Smart Chain, Cosmos, Polkadot, Tron, Avalanche blockchains. Traders will be able to exchange any of those blockchain assets against any each other from wallet to wallet in a trust-less, permission-less and secure way.

4. Atomic Swaps

At its most basic, a blockchain is a computer file used for storing data – information. Like any computer file, it exists on a digital storage medium, such as a computer hard drive. And it takes the form of a string of binary "bits", ones and zeros, which can be processed by computers to be made readable by humans. One can also say, a blockchain is a distributed service that allows clients to publish transactions to a publicly readable, tamper-proof distributed ledger. Blockchains, however, have three properties which, while not unique individually, when put together mean they function very differently than other types of computer files. They are distributed, encoded and open. Current state of technology doesn't support interoperation among blockchains, which means one cannot transfer his assets from one blockchain to another without the need of a trusted third party. Here comes the need of atomic swaps. An atomic swap can be defined as a transaction between two parties that does not depend on a third party, for instance a centralized exchange, and either happens in full, or not at all. The reason this is preferred is the fact that trusting a third party is a risk, as that third party could, accidentally or on purpose, leave one or both clients without their funds. What makes the Ethereum blockchain particularly suitable for atomic swaps is the fact that it supports smart contracts, and the fact that Solidity, a language used to write these contracts, is Turing-complete. A method used to implement cross-chain atomic swaps with contracts has existed for a few years already. These contracts are known as Hashed TimeLock Contracts (HTLC). The idea is that both contracts (one on each blockchain) store the hash of a secret key initially only known to one of the two parties (client A from now on). Both parties publish a swap contract on the two different blockchains and commit to locking their funds in the contracts for a predefined amount of time. The time limit on the contract on which client A deposits their funds is longer than that of the contract deployed by client B. Only after this time limit has passed can they request a refund. Client A deposits their funds first, and only after client B has checked that this deposit is right, they deposit as well. Only when a claim-request specifying the secret key is sent to the contract, will the contract transfer the funds. The swap contract verifies the key by generating a SHA-256 hash from it and checking whether it equals the hash that is hard-coded into the contract. Client A can now use the key to claim the funds put up by client B, an action that makes the key publicly available. Client B can then claim the funds from the swap contract put up by client A, using the secret key. Because the time limit on the contract that client A published is longer than that on the contract client B published, there is no chance that client A can simultaneously claim client B's funds and refund their own (because a refund can only be requested when the time limit has passed). If client A does not claim their funds, client B cannot either and if client A does claim their funds, client B automatically can as well. This is exactly what an atomic swap entail.

4.1. Single Chain

The single-chain swaps are done by having both parties send their funds to a single swap contract. The single-chain implementations depicted in Figures 1 and 2 have a certain amount of commonality at the start of the process. Therefore, these steps will be explained first, after which the subsections dedicated to the specific swaps explain the remainder of the steps for each case. Please note that in the single-chain atomic swap process, we found there to be no direct need to use a HTLC.

1. The clients first need to agree on the number of tokens each party sends to the other, a time limit, and which party will deploy the contract necessary for the swap (client A from now on). This all happens off-chain, and the way this communication happens is not in the scope of this project. The time limit is used as a safety measure. The whole transaction must happen within this time limit for the transaction to succeed. Only after the time limit has passed are the parties allowed to request a refund. This measure is necessary from a user experience perspective, as it makes transactions much swifter, because neither party can keep the other party waiting indefinitely.

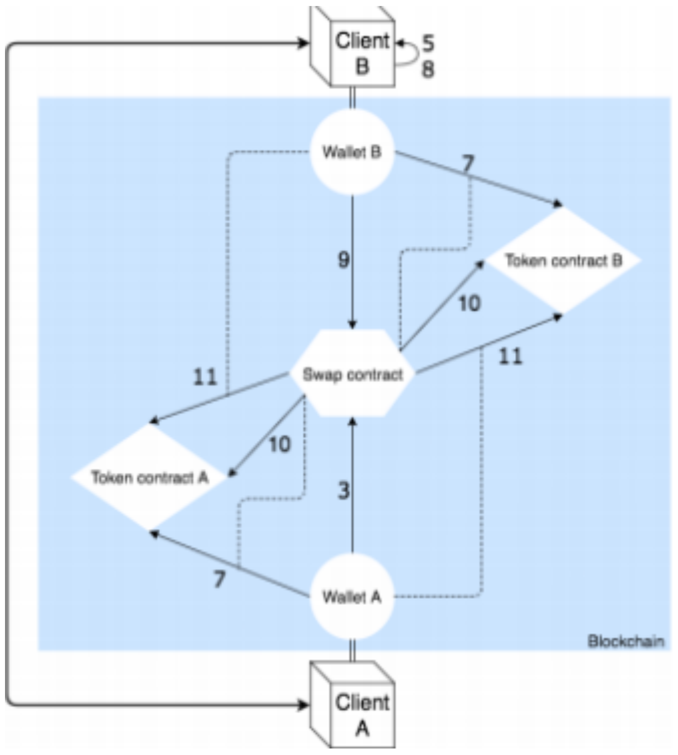
2. When the details from the first step have been agreed upon, the parties exchange their Ethereum addresses. This happens off-chain as well.

3. Both parties now have all the information to create and deploy the contract. While only one of the clients must actually deploy the contract, it is important that both are able to compile it, for reasons that become apparent in steps 4 and 5. In our case, client A generates and deploys the contract. This contract is slightly different for the two single-chain cases, but the general setup is the same.

4. Client A sends the address of the swap contract it published, the source code for the contract, the compiler used, and the arguments used to client B.

5. Client B compiles the contract themselves using the available information and compares it to the contract on the chain. This is an easy way to make sure that the contract that was deployed was exactly what both parties agreed upon.

6. Client B confirms to client A that they accept the contract as it was deployed. If the contract was not as expected, the swap can either be stopped completely, or re-tried. This is more related to the user experience than to atomic swap process, and happens completely off-chain, thus it does not fall within the scope of the project.



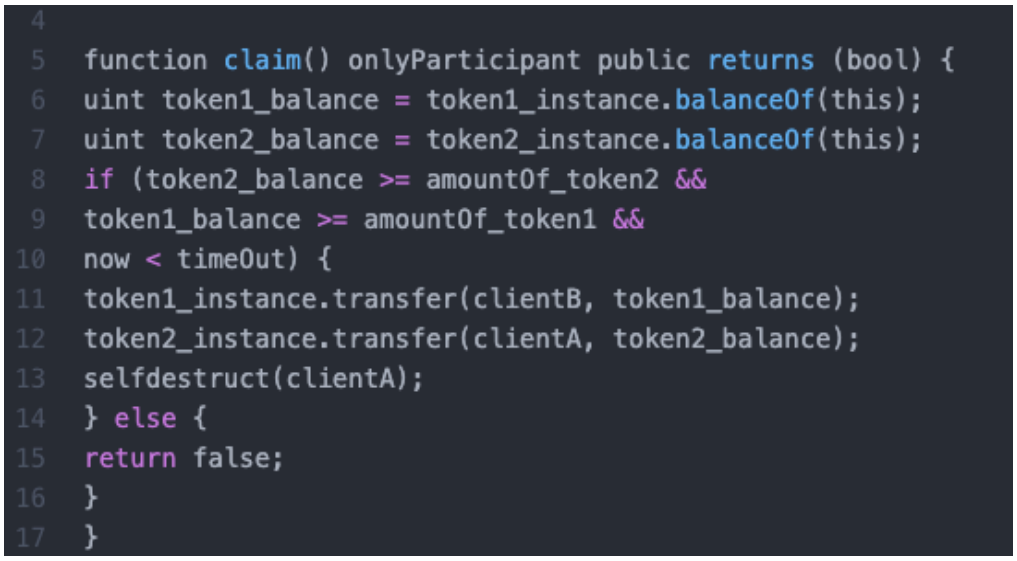
7. At this point it is safe for both parties to send their tokens to the contract. This happens in the following way: client A executes transfer () on token contract A, in which they request the contract to move tokens from their own wallet to the swap contract. Client B does the same with token contract B. The order in which this happens does not matter, as the contract does not execute any transaction until all funds are on the contract.

8. One of the clients checks whether the funds have been transferred to the swap contract. This happens by indexing the blockchain and checking the transactions. Which party does this can be decided off-chain, for the purposes of this explanation client B will do this.

9. Once client B sees the right amounts on the swap contract, they send a confirmation of this to the swap contract (by calling claim ()).

10. The swap contract checks if it has received the agreed upon funds from both clients. It must request its balance at both of the token contracts, as it does not have that information itself.

11. If the contract concludes the funds have both been received, it will proceed to transfer the funds to the clients. It does this by executing transfer() on each token contract, requesting a token transfer to the corresponding wallets. Validation This process should be atomic due to multiple factors. Firstly, once the funds are on the swap contract, only the contract itself can move the funds. Executions of Ethereum smart contract functions are atomic as a whole, thus we can be confident that any function is either completely executed or not at all [15, 14]. The two clients can call two functions on the contract to move the funds, claim() and refund(). At no point is either client capable of stealing the other client's funds. The following snippet of code demonstrates this:

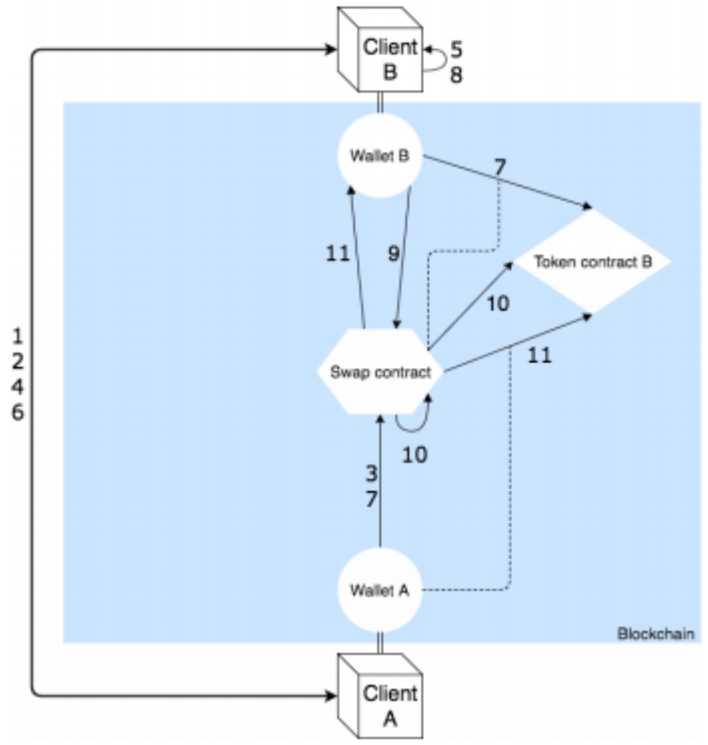


```
function claim() onlyParticipant public returns (bool) {
uint token1_balance = token1_instance.balanceOf(this);
uint token2_balance = token2_instance.balanceOf(this);
if (token2_balance >= amountOf_token2 &&
token1_balance >= amountOf_token1 &&
now < timeOut) {
token1_instance.transfer(clientB, token1_balance);
token2_instance.transfer(clientA, token2_balance);
selfdestruct(clientA);
} else {
return false;
}
}
```

This function checks the amount of each token it owns with the respective token contracts (line 2 and 3). If these amounts are at least the amount of tokens that was agreed upon, the transfer happens. As only this contract can transfer these tokens, there is no risk of either person not receiving their tokens: the transfer only happens when both parties sent (at least) the correct amount of tokens, and then one of them calls claim(), which sends the funds to both parties at once. The reason we chose to send all tokens (even if either party has sent more than was agreed upon) is the fact that otherwise those tokens will get lost. As all addresses involved in the transaction are hardcoded in the contract, outsiders are not be able to re-route the funds when claimed or refunded. The contract self-destructs after the transfer has happened (line 9), meaning that from that point on interaction with the contract is not possible anymore. The single-chain swaps do not require a secret key (as used with HTLCs) because there is only one contract. The cross-chain swaps have two contracts that should only allow certain functions to be executed once certain steps have been executed on the other contract. Because the chains cannot see what happens on the other chain, the key is necessary to make sure that clients do not even have the option of executing functions out of order, which might result in stolen funds. Aside from the atomicity, the most important part of the whole process from a security-perspective is the client-side contract confirmation. This part of the process makes sure that neither party can deploy a contract different from what the other party expects. This is very similar to the contract confirmation that Etherscan has in place. The individual that deployed the contract can send the source code and other relevant information to Etherscan via their website, so that Etherscan can then compile the contract under the exact same circumstances. If their compile is the same that is found on the chain, the contract is verified. This recompile should happen locally.

4.2 Single-chain Coin-Token Swap

While the differences between the tokens swap and the coin-token swap are minor, they are significant enough to cause confusion if not highlighted. The main difference is the fact that the swap contract does not have to do an external call with a token contract whether the funds have been deposited (for the Ether).



7. Client A sends coins to the swap contract. Client B executes **transfer()** on token contract B, in which they request the contract to move tokens from their own wallet to the swap contract. The order in which this happens does not matter.

8. Client B checks whether the funds have been transferred to the swap contract.

9. Once they see the right amounts on the swap contract, they send a confirmation of this to the swap contract (by calling **claim()**).

10. The swap contract checks that it has received the agreed upon funds from both clients. It must ask token contract B what its balance is, as it does not have that information itself. For the coin it can just check its own balance without making any further requests to outside sources.

11. If the contract concludes the funds have both been received, it will proceed to transfer the funds to the clients. For the token transfer, it does this by executing transfer() on the token contract that is involved. The coins can just be sent in the regular way.

5. Meliora Cross-chain Swap Platform

Meliora uses the Hashed TimeLock Contract for the cross-chain atomic swaps. In a simple two-party swap, each party publishes a contract that assumes temporary control of that party's asset. This hashed timelock contract stores a pair **(h,t)**, and ensures that if the contract receives the matching secret s, h = H(s), before time t has elapsed, then the contract is triggered, irrevocably transferring ownership of the asset to the counterparty. If the contract does not receive the matching secret before time t has elapsed, then the asset is refunded to the original owner. For multi-party cross-chain swaps, we will need to extend these notions in several ways.

The cross-chain atomic swap is possible, because of these key technologies:

- **Cryptographic Hash Function**
  A cryptographic hash function (CHF) is a hash function that is suitable for use in cryptography. It is a mathematical algorithm that maps data of arbitrary size (often called the "message") to a bit string of a fixed size (the "hash value", "hash", or "message digest") and is a one-way function, that is, a function which is practically infeasible to invert. Ideally, the only way to find a message that produces a given

hash is to attempt a brute-force search of possible inputs to see if they produce a match or use a rainbow table of matched hashes. Cryptographic hash functions are a basic tool of modern cryptography.

- **Time Locked Transactions**
  The receiver has a limited amount of time to confirm on the blockchain that he has received the funds. It guarantees that the blockchain will reverse the transaction if the receiver never confirms it and the assets will be sent back to the sender.

- **Hash Locked Transaction**
  A special secret key (different from the private key) used to unlock the transaction and get the funds. the hash is derived from the secret key, but it is a one-way process - there is no way to go back from the hash and reveal the secret key. While the hash is visible for the receiver, the secret key is the only piece of information which can unlock the transaction.

- **Smart Contracts**
  A smart contract is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts allow the performance of credible transactions without third parties. These transactions are trackable and irreversible.

- **Bitcoin Script**
  Bitcoin uses a scripting system for transactions. Script is simple, stack-based, and processed from left to right. It is intentionally not Turing-complete, with no loops. A script is essentially a list of instructions recorded with each transaction that describe how the next person wanting to spend the Bitcoins being transferred can gain access to them. The script for a typical Bitcoin transfer to destination Bitcoin address D simply encumbers future spending of the bitcoins with two things: the spender must provide:

1. A public key that, when hashed, yields destination address D embedded in the script.
2. A signature to prove ownership of the private key corresponding to the public key just provided.
3.

Algorithm in example:

1) Alice has BTC, Bob has ETH
2) Alice creates a secret key consisting of random letters, words or numbers.
3) A hash is created from Alice's secret key
4) Alice submits a transaction on the Bitcoin blockchain which is sent to Bob's address. It is locked in both of terms of time (where Bob has a certain amount of time to confirm it) and in terms of security (with the hash).
5) Alice sends the hash to Bob
6) Now it is time for Bob to create his Ethereum transaction to Alice (again it is both time and hash locked, where the hash is the same as Alice's). As a result, we have a transaction on both the Ethereum and the Bitcoin blockchains which is locked with the same hash.
7) Alice unlocks Bob's Ethereum transaction (because she holds the secret key). By unlocking his transaction, Alice reveals the secret key, and it is publicly recorded on the blockchain.
8) Bob can now see the secret key and unlock the Bitcoin transaction which was aimed at him.
9) They have exchanged assets without the need to trust or know each other.

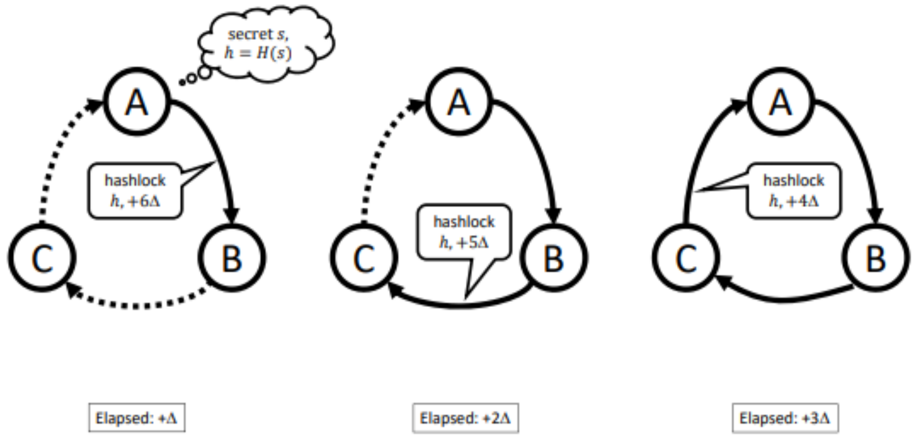


Figure 1: Atomic cross-chain swap: deploying contracts

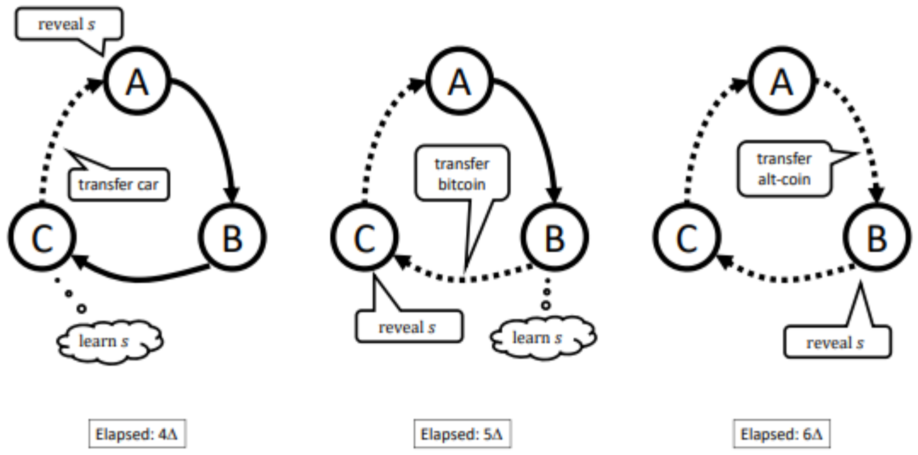


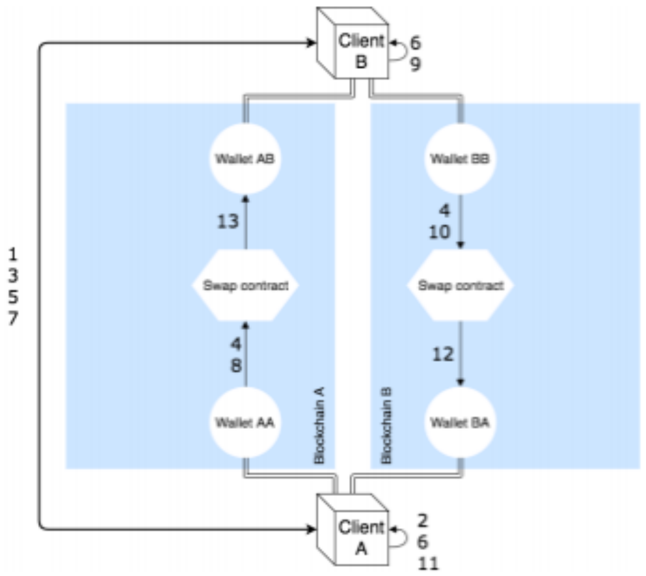
Figure 2: Atomic cross-chain swap: triggering arcs

5.1 Meliora Cross-chain Coin Swap

1. Client A sends their funds to swap contract A. It is important for client A to be the first to deposit their funds, for reasons that will become apparent in the next steps.
2. Client B has access to the blockchain, and can thus see when and how much client A has sent to swap contract A. This happens by indexing the blockchain and

checking the transactions. If this is the amount the parties agreed upon, the process can continue.

3. Client B transfers their funds to swap contract B.
4. Client A checks whether the transaction has succeeded and whether it was the right amount. This happens locally.
5. Client A now calls the claim() on contract B using the secret key to claim the funds. In the process of claiming the funds, the secret key is sent in plaintext to contract B, and from then on is available on the blockchain for anyone to see, including client B.
6. Client B uses the secret key to claim the funds on swap contract A. The destination address in the swap contract is hard-coded, which means that it does not matter that the secret key is no longer secret; even if someone else would use it to call the claim(), the funds would still be sent to client B.

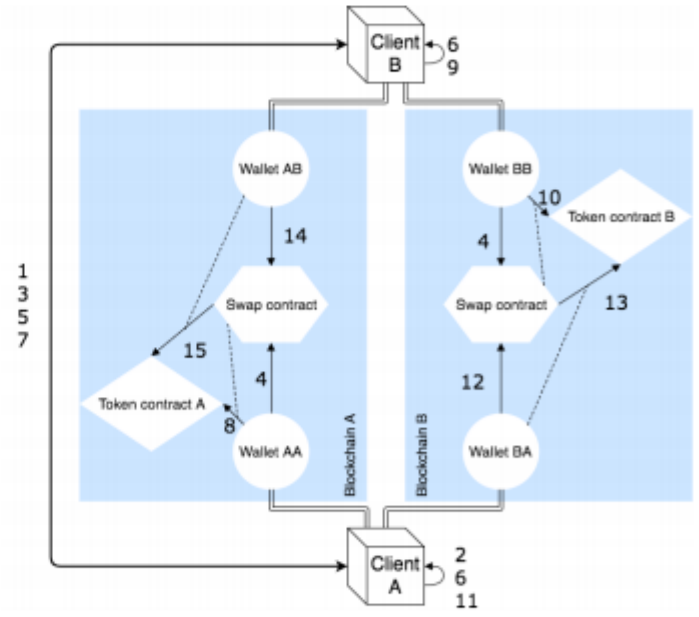


Validation: The validity of the cross-chain swap contracts is again confirmed by recompiling the contracts off-chain and comparing them to the versions that were published on-chain. The main difference with respect to the single-chain swaps is the fact that it requires client-side input during the swap, retrieving the key from blockchain

B and then using it to claim the funds on blockchain A. This is because the two chains cannot execute things on the other chain. This client-side input can happen anywhere between the moment client A claims their funds and the moment swap contract A expires, which should result in no loss of funds. However, client-side input halfway through the swap does leave open the possibility of non-atomicity. Alternatives to the HTLC should be considered to solve this problem, as this is not a consequence of the implementation, but of the theory of the HTLC, which depends on user input during the swap. Assuming client B is able to execute claim() on the contract on blockchain A, the method using the secret key appears to be secure, as the secret key stays locally on the machine of client A until they claim their funds.

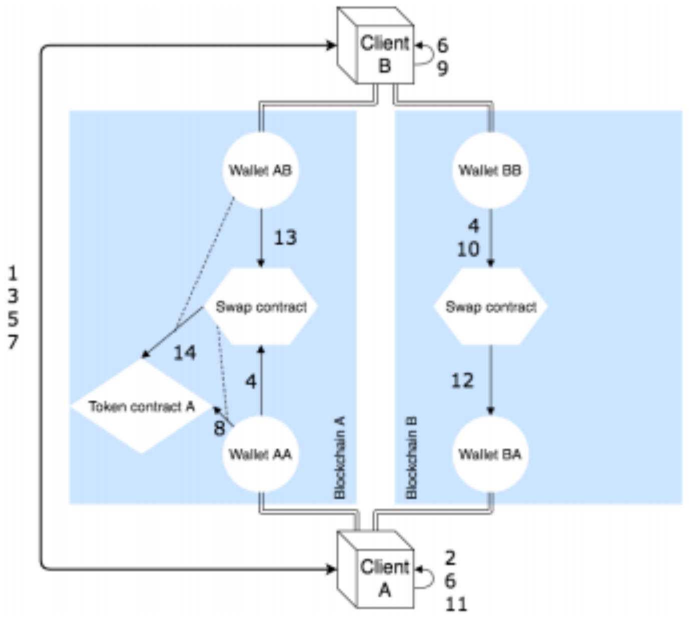
5.2 Meliora Cross-chain Token Swap

1. Client A calls transfer() on token contract A such that their tokens are transferred to swap contract A. It is important for client A to be the first to deposit their funds.
2. Client B has access to the blockchain, and can thus see when and how many tokens client A has deposited to swap contract A. This happens by indexing the blockchain and checking the transactions. If this is the amount the parties agreed upon, the process can continue.
3. Client B transfers their funds to swap contract B by calling transfer() on token contract B.
4. Client A checks whether the transaction has succeeded and whether it was the right amount. This happens locally.
5. Client A now calls the claim() on contract B using the secret key to claim the funds.
6. Swap contract B calls transfer() on token contract B to move its tokens to the wallet of client A.
7. The key is now freely available on the blockchain. Client B uses the key to claim the funds from swap contract A.
8. Swap contract A now calls the transfer() on token contract A to move its tokens to client A's wallet.



5.3 Meliora Cross-chain Coin-Token Swap

1. Client A calls transfer() on token contract A such that their tokens are transferred to swap contract A. It is important for client A to be the first to deposit their funds.
2. Client B has access to the blockchain, and can thus see when and how much client A has sent to swap contract A. This happens by indexing the blockchain and checking the transactions. If this is the amount the parties agreed upon, the process can continue.
3. Client B transfers their funds to swap contract B.
4. Client A checks whether the transaction has succeeded and whether it was the right amount. This happens locally.
5. Client A now calls claim() on contract B using the secret key to claim the funds.
6. The key is now freely available on the blockchain. Client B now uses that same key to call the claim function on swap contract A.
7. Swap contract A now calls transfer() on token contract A to move its tokens to client A's wallet.



6. Meli-Wallet

Meli-Wallet, a convertible multi-chain wallet, will be developed to address the issue of interoperability between different blockchains and to achieve value exchange between different blockchains. The wallet uses hash timelock technology to allow the exchange of various digital currency assets. Every blockchain in conventional blockchain projects is a closed island. Users must exchange various digital currencies through CEXs, which runs counter to the concept of decentralization and non-tamperability of blockchains. Furthermore, the security and reliability of blockchains are entirely dependent on Exchanges. Meli-Wallet which uses hash timelock technology can exchange digital assets across multiple blockchains without the use of a third-party intermediary, ensuring decentralization, security, and reliability. Users will be able to hold, swap, transfer coins in multiple chains. Some key features of Meli-Wallet are; no need to account sign up, anonymous, non-custodial, trust-less security and interoperable.

7. Token Utility

The Meliora Utility Token (MORA) is a native utility token for our platform and itself has multiple forms of utility, essentially being the cornerstone of our present and future ecosystem. MORA is an ERC20 token and it has four main utilities which are fee discount, governance, revenue generating and passive income.

- **Fee Discount**

On Meliora wallet-to-wallet cross-chain platform, traders who hold more than 5000 MORA tokens are rewarded with 50% fee discount. Meliora charges the Taker 0.6% of the transaction amount, while the rate is 0.2% for the Maker.

- **Governance**

Meliora is truly decentralized and gives right of speech to its community, which makes Meliora a community driven decentralized platform. Users can initiate proposals and vote on the proposals in the community as a way of on-chain governance. Those who hold more than 1% of the total tokens can initiate proposals, and any user holding MORA can vote for or against a proposal. The voting lasts for two days, with one token for one vote. After the voting, the proposals receiving more positive votes than the negative are passed. The voting process is uniformly managed by the governance contract. For proposals passed by voting, the governance contract automatically performs corresponding operations on the chain.

- **Revenue Generating**

Meliora distributes %20 of monthly DEX profits to MORA holders. Thus, holding Mora tokens will provide monthly revenue. To be eligible to share the monthly fees revenue, holders must be keeping more than 5000 MORA tokens in their noncustodial wallets for the last 15 days before the monthly snapshot. Snapshot will be taken on the first day of each month.

- **Passive Income**

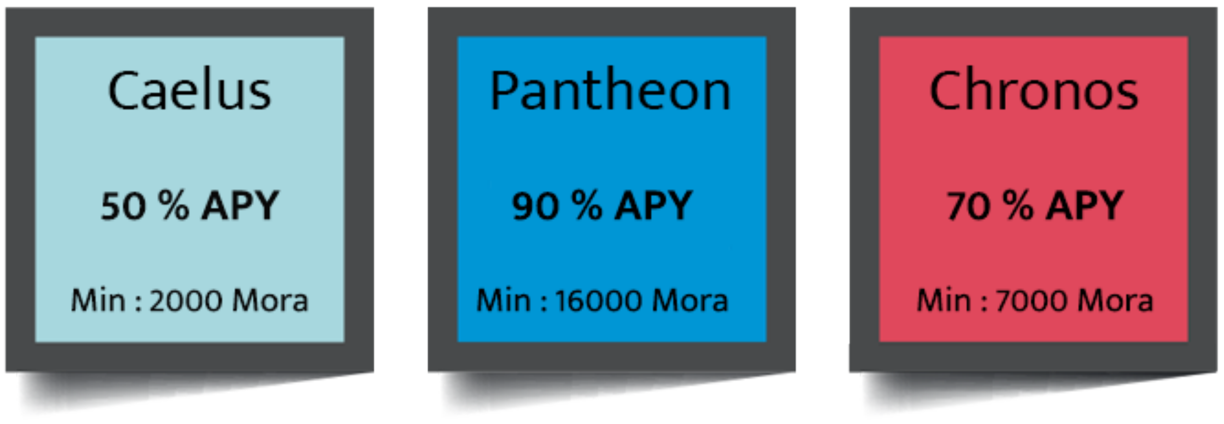
Mora holders enjoy passive income by staking their tokens. Meliora offers high yield staking opportunities to its community. Holders will have a chance to get up to 90% APY.

8. Staking

Staking is the process of holding tokens in a smart contract to support the operations of a network. Participants are rewarded for depositing and holding coins, with constant guaranteed time-based returns. Rewards are calculated based on the staked amount and duration of staking. With the advent of DeFi, staking has been recording phenomenal growth and sparking interest. As it stands, there is more than $10 billion in USD value locked via staking in the Tezos, Cosmos, and Polkadot protocols. On the other hand, total value locked in DeFi platforms has exceeded $40B.

MORA token holders will have the chance to increase their tokens just by holding them. Token holders will be able to lock their MORA tokens on our staking platform. Staking system will incentivize our community members to hold more tokens. Staking rewards are calculated based on the token amount and duration that user stakes. We will offer three different staking tiers which will be based on the locked token amount. Minimum amount to stake is 2000 MORA tokens and maximum amount to stake depends on the user’s call. 1000000 MORA tokens have been allocated as staking rewards and every month 100000 tokens will be unlocked from the vested contract. In the case of non-distributed rewards, remaining extra tokens will be automatically burnt by the contract every month. Thus, MORA will have a deflationary feature. The graph below shows the staking rewards for each tier.

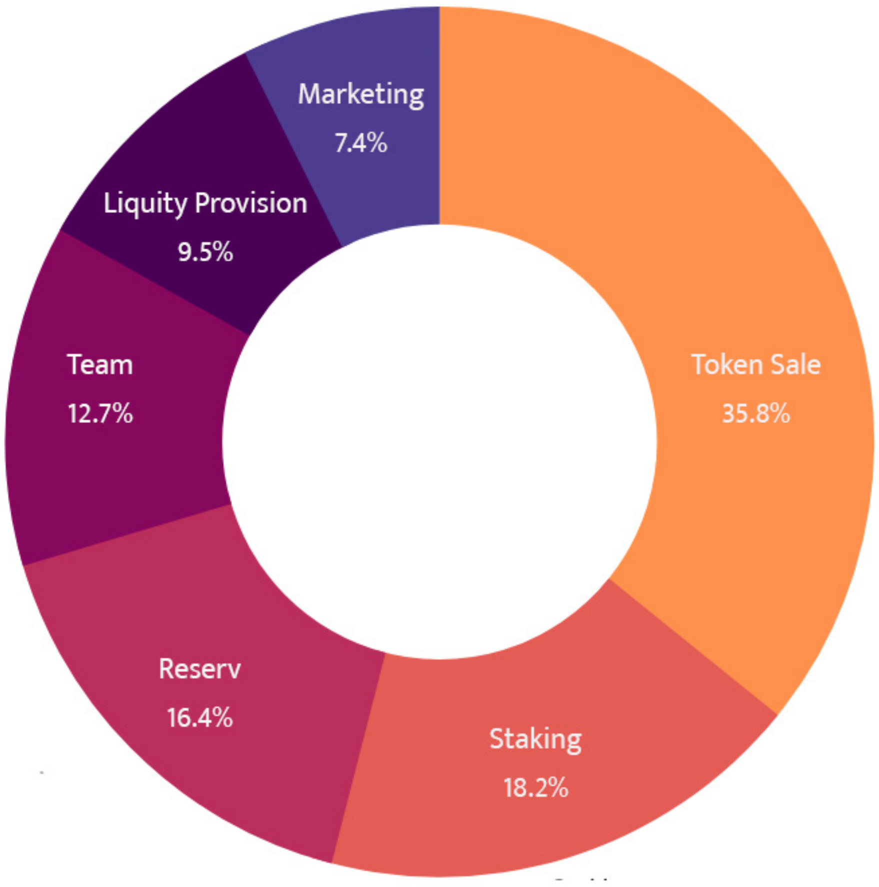
Staking Tiers



Caelus	Pantheon	Chronos
50 % APY	90 % APY	70 % APY
Min : 2000 Mora	Min : 16000 Mora	Min : 7000 Mora

9. Tokenomics

MORA is the governance and utility token of Meliora platform. MORA is a standard ERC20 token and there will ever be maximum of 5500000 tokens. Thus, maximum total supply will never be more than 5500000. On the contrary, MORA will have deflationary features as the surplus staking rewards each month will be burnt forever. Staking contract is designed to distribute 100000 MORA tokens every month and undistributed tokens will be burnt by the contract every month. The chart below shows the token distribution of **MORA Tokens.**



Allocation	**Percentage**	**# of Tokens**
Token Sale	35.80 %	1.970.000
Staking	18.20 %	1.000.000
Reserve	16.40 %	900.000
Team	12.70 %	700.000
Liquidity Provision	9.50 %	525.000
Marketing	7.40 %	405.000

10. Roadmap

2020 Q4

- Idea & Concept
- Team Assemble
- Strategic Plan

2021 Q1

- White Paper 1.0
- Website Launch
- Token Generation Event – Token Sale
- Meliora Staking Platform Launch

2021 Q2

- Release of Meli-Wallet
- Meliora Platform Testnet
- Meliora Crosschain Platform Launch
- BSC Integration

2021 Q3

- Polkadot Integration
- Cosmos Integration
- Avalanche Integration

10. Legal Disclaimer

This white paper does not constitute advice nor a recommendation by the Meliora Finance, its officers, directors, managers, employees or any other person to any recipient of this document on the merits of the participation in the TGE sale. Participation in the TGE carries substantial risk and may involve special risks that could lead to a loss of all or a substantial portion of such an investment. Do not participate in the token sale unless you are prepared to lose the entire amount you allocated to purchasing MORA tokens. MORA tokens should not be acquired for speculative or investment purposes with the expectation of making a profit or immediate re-sale. No promises of future performance or value are or will be made with respect to MORA tokens, including no promise of inherent value, no promise of continuing payments, and no guarantee that MORA tokens will hold any value. This white paper is not a prospectus or disclosure document and is not an offer to sell, nor the solicitation of any offer to buy any investment or financial instrument in any jurisdiction and should not be treated or relied upon as one. This white paper is for information only. All information here that is forward looking is speculative in nature and may change in response to numerous outside forces, including technological innovations, regulatory factors, and/or currency fluctuations, including but not limited to the market value of cryptocurrencies. This white paper is for information purposes only and is subject to change.

11. References

Bitcoin - Monero Cross-chain Atomic Swap: https://eprint.iacr.org/2020/1126.pdf

Bowe, S., Hopwood, D.: Hashed time-locked contract transactions https://github.com/bitcoin/bips/blob/master/bip-0199.mediawiki

M. Herlihy, "Atomic cross-chain swaps," in Proceedings of the 2018ACM Symposium on Principles of Distributed Computing

Bitcoinwiki, "Atomic swap": https://en.bitcoin.it/wiki/Atomic swap

https://www.researchgate.net/publication/223217050_An_improved_algorithm_for_multi-way_trading_for_exchange_and_barter

https://www.researchgate.net/publication/341297894_Cross-chain_Transactions

https://www.researchgate.net/publication/335937528_Extending_Atomic_Cross-Chain_Swaps