Decentralized Cryptocurrency Exchange

A Proof-of-Concept based on Hashed Timelock Contracts

Bachelor of Science Thesis in Computer Science and Engineering

Magnus Andersson
Kevin Chen Trieu
Petros Debesay
Jesper Persson
Jacob Torrång
Samuel Utbult

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY   UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Bachelor's thesis 2018:05

Decentralized Cryptocurrency Exchange

A Proof-of-Concept based on Hashed Timelock Contracts

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Decentralized Cryptocurreny Exchange
A Proof-of-Concept based on Hashed Timelock Contracts
Magnus Andersson
Kevin Chen Trieu
Petros Debesay
Jesper Persson
Jacob Torrång
Samuel Utbult

Supervisor: Magnus Almgren, Computer Science and Engineering, Chalmers University of Technology
Examiner: Arne Linde, Department of Computer Science and Engineering

Bachelor of Science Thesis 2018:05
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: David McBee. Licensed under CC0 1.0.

Gothenburg, Sweden 2018

Abstract

In recent years, cryptocurrencies have grown rapidly to a market capitalization of 400 billion US dollars. As the value of cryptocurrencies has increased, several exchanges have emerged. Some of these exchanges have been targets of attacks leading to losses of cryptocurrency worth hundreds of millions of dollars. The attacks have been made possible as a result of exchanges keeping track of the cryptocurrency in circulation on the exchange by storing their users' funds. A successful attack puts not only the exchange, but all its users at risk.

This bachelor's thesis explores an alternative to a centralized cryptocurrency exchange by trading directly on the blockchain using peer-to-peer technology. This decentralized approach eliminates the risk for attacks with far-reaching consequences as users' funds are not stored in a central location. A protocol for exchange between cryptocurrencies is proposed and a prototype based on this protocol is implemented. An evaluation is done to ensure the quality of the prototype and protocol based on three parameters - delay, cost and trading pairs.

The protocol is proven to be a possible alternative to the solutions offered today. The prototype displays a proof-of-concept of a decentralized platform that implements this protocol.

Keywords: Blockchain, Decentralized, Cryptocurrency, Bitcoin, Ethereum, OrbitDB, IPFS, Atomic Swap, Smart Contract

Sammandrag

De senaste åren har kryptovalutor ökat i popularitatet till ett sammanlagt börsvärde på 400 miljarder amerikanska dollar. Allt eftersom värdet har ökat har ett flertal handelsplatser för kryptovalutor utsatts för cyberattacker, där kryptovalutor till ett värde av flera hundra miljoner dollar gått förlorade. Attackerna har varit möjliga till följd av att handelsplatserna hanterar allt kapital i cirkulation genom att förvara användarnas tillgångar i ett centraliserat system. En lyckad attack utsätter både handelsplatsen och dess användare för risker.

Den här kandidatuppsatsen utforskar ett alternativ till centraliserade handelsplatser för växling av kryptovalutor genom att genomföra byten direkt på blockkedjor med hjälp av P2P-teknologi. Detta tillvägagångssätt minskar incitament för att genomföra hackerattacker eftersom tillgångarna inte lagras på en central plats. Ett protokoll för växling mellan kryptovalutor föreslås och en prototyp baserat på detta protokoll implementeras. Därefter utvärderas prototypen för att säkerställa kvaliteten på både prototypen och protokollet efter tre parametrar: fördröjning, kostnad samt antalet möjliga växlingspar.

Protokollet visar sig vara ett möjligt alternativ till de lösningar som erbjuds idag. Prototypen visar ett konceptbevis på en decentraliserad plattform som implementerar detta protokoll.

General terms and abbreviations

Atomic swap/Atomic exchange An exchange between two parties that either succeeds or reverts to its original state. It cannot stop in an intermediate state.

Bitcoin Bitcoin with a lowercase *b* refers to the currency in the Bitcoin network.

Blockchain A public digital ledger in which transactions are recorded chronologically in cryptographically hashed blocks. A blockchain in this text has a cryptocurrency associated with it.

BTC An abbreviation for Bitcoin, which is a blockchain.

Centralized (system) A system where instructions are issued and delegated by a top-level entity.

Cryptographic hash function A one-way function used to map data of arbitrary length to data of fixed length and is suitable to use in cryptography.

Decentralized (system) A system where complex behaviour emerges from local computations and consensus by several low-level entities.

Ether Refers to the currency of the Ethereum network.

ETC An abbreviation for Ethereum Classic, which is a blockchain.

ETH An abbreviation for Ethereum, which is a blockchain branched off from Ethereum Classic.

HTLC An abbreviation for "Hashed Timelock Contract", and it is a way of executing secure exchanges.

P2PKH Pay To Public Key Hash is the standard transaction in the Bitcoin network.

P2SH Pay To Script Hash is a transaction in the Bitcoin network that allows arbitrary code for spending and verifying a transaction.

Peer-to-peer A system of nodes where clients communicate directly with each other without the need of a server.

Pre-image The original value used to create a hash.

Proof-of-concept An implementation that demonstrates that a theoretical concept has practical potential.

Proof-of-work A piece of data that is difficult to acquire but easy to verify.

Smart contract A piece of deterministic code, run on the blockchain, with which various entities can interact.

Transpiling The process of converting source code written in a certain programming language into the equivalent source code in another programming language.

UTXO Unspent Transaction Ouput in the Bitcoin network.

Contents

1
Introduction

In recent years, the increased popularity of cryptocurrencies has been hard to ignore. The total market capitalization of all cryptocurrencies is, as of May 2018, 400 billion US dollars. Also, between 2016 and 2018, the increase in the number of cryptocurrencies was roughly threefold, growing from 551 to 1551 cryptocurrencies [1, 2].

Blockchains and cryptocurrencies are rapidly gaining adoption in mainstream society, with the Bitcoin network leading the way. Cryptocurrencies are often touted as an alternative to government issued currencies, however they are not limited to that field of application. For example the cryptocurrency ether, in the Ethereum network, can be used in combination with programmable contracts deployed on the blockchain. This has gained a lot of traction over the years as can be seen by the increasing value of ether [3].

As cryptocurrencies with different fields of application increase in popularity, so has the demand to exchange between them. It is not surprising that such a demand has lead to the creation of several cryptocurrency exchanges.

1.1 Outline

Chapter 1, *Introduction*, explains the problems that currency exchanges are currently facing and what the goals of the thesis is.
Chapter 2, *Technical background*, briefly summarizes the technical concepts that are necessary to understand the rest of the thesis.
Chapter 3, *Methods*, describes the requirements, design, and implementation of the prototype.
Chapter 4, *Evaluation and Results*, describes how the prototype is evaluated and then presents the results.
Chapter 5, *Discussion*, gives a qualitative discussion of the project.
Chapter 6, *Conclusion*, concludes the report with a brief summary.

1.2 Background

Cryptocurrency is a new technology that is quickly gaining popularity as an alternative to currencies issued by central banks, commonly referred to as *fiat money* [4]. Two examples of fiat currencies are the *Swedish krona*, which is issued by *Sveriges*

Riksbank [5], and the *United States dollar*, which is issued by the *Federal Reserve* [6].

At any given point in time, there exists a limited number of cryptocurrency coins whose owners are specified in a decentralized blockchain. The cryptocurrency coins have intrinsic value as one has to spend physical resources to acquire them [7]. The concept of cryptocurrencies was coined in 1998 and the first full specification of a cryptocurrency was published in 2009 when Satoshi Nakamoto released his white paper for Bitcoin [8]. Bitcoin is not the only cryptocurrency. A large number of cryptocurrencies exist and each currency is used in different markets. An actor in a specific market has to exchange cryptocurrencies to purchase goods or services in another market where the cryptocurrency that they are in possession of is unusable.

Exchange of currencies is essentially trading one currency for another. An individual wishing to make an exchange has two options. The first is to ask a centralized financial institution that offers the service of exchanging a currency for another, i.e. a currency exchange [9]. The other option is to trade directly with another individual wishing to make the opposite trade, thereby bypassing the centralized exchange and making a decentralized exchange.

In today's cryptocurrency world, centralized exchanges are prevalent [10, 11]. In a centralized exchange, the user typically relinquishes control over the cryptocurrency by placing it in the internal wallet of the centralized exchange. The exchange essentially acts as a bank. This has advantages as it allows for fast trades within the exchange. The exchange can also assure that both parties receive their part of the trade. Once the user is satisfied with the trades made, the money can be extracted, and control returns to the user.

However, there are also disadvantages with centralized exchanges. It is easier for attackers to steal large amount of funds stored in an exchange. An exchange might also fold before you get a chance to withdraw the currency [12]. Additionally, nothing prevents a centralized exchange from abusing its powers by blocking specific users or even halting trades arbitrarily [13]. Yet another factor is that users of centralized exchanges cannot be fully anonymous as information about them has to be stored in a centralized system. Clearly, an alternative approach is needed for users looking to trade cryptocurrencies, and a decentralized exchange platform might be the answer to the aforementioned issues.

1.3 Purpose

The purpose of this project is to investigate the possibility of performing a cryptocurrency exchange without the need of a middleman, specify a number of requirements that a decentralized exchange should fulfill and develop a prototype based on these requirements. The prototype should have comparable features to its centralized counterparts. One of the main focus points will be to eliminate the risk of users getting scammed by the second part in the exchange.

1.4 Problem

To develop a functional decentralized prototype, three main problems need to be addressed:

- How to execute a secure, as defined in section 1.4.1, decentralized cryptocurrency exchange between two users?
- How to facilitate decentralized communication between users?
- How to decentralize the data storage?

These problems come with challenges that will be further discussed below.

1.4.1 Secure cryptocurrency exchange

The measure of security is defined in this thesis as the risk of getting scammed when using the prototype. Two existing solutions to ensure a secure exchange exist today. The first solution is to use a middleman, usually a trading platform that both users can trust. However, such a solution leads to centralization, which is not desirable in an otherwise decentralized platform. The second solution is to have the users meet physically to conduct an exchange. The problem with this solution is that the users are physically restricted, and intercontinental transfers would be difficult to carry out.

For these reasons, a solution that facilitates exchanges without the need of middlemen or physical presence is needed. For such a solution, it is required that an exchange is done under an agreement without the possibility of this agreement changing during or after the exchange. Such an exchange needs to be atomic, which means that an exchange is indivisible, irreducible, and either happens or reverts to its original state [14].

1.4.2 Decentralized communication

In a centralized platform, communication between users occurs through a communication channel that goes through a central server. A solution that utilizes this method to manage communication is certainly not decentralized. Nevertheless, decentralized communication is a mature area and there exists a large number of methods to communicate in a decentralized manner. BitTorrent is an example of a protocol that achieves this [15].

A decentralized cryptocurrency exchange should use a decentralized protocol to create communication channels between users so they can find each other and exchange information. This protocol has to address these issues:

- **Connectivity** – The nodes need to be able to find and connect to the network.
- **Routing** – The nodes need to be able to find each other in the network.
- **Security** – Malicious nodes should not be able to jeopardize the transaction.

1.4.3 Decentralized storage

In order to retrieve user bids and other data stored in the platform, the information needs to be stored in a physical location. Two common ways of storing data today is either through the traditional solution, where the data is stored in one central location [16], or through a distributed system, where the data is stored in several locations and linked through a data communication network. The first solution is clearly not suitable for a decentralized platform, but the second solution is. Before such a system can be considered fully decentralized, the following issues need to be addressed:

- **Independency** – The nodes storing data should not be reliant on each other.
- **Security** – The data should only be manipulated by the owner.
- **Reliability** - The data should immediately be replicated when added.

1.5 Scope

The scope of this thesis is to make a prototype that will facilitate decentralized exchanges between a subset of cryptocurrencies. This includes bitcoin on the *Bitcoin* (BTC) network, and ether on the *Ethereum* (ETH) and *Ethereum Classic* (ETC) networks. In terms of security, the focus will be on making a protocol executing atomic exchanges making it difficult to scam the counterpart on their funds.

The project will run on test networks as well as private networks. Using private networks allows for the prototype to be tested in a controlled environment. This controlled environment allows for the accumulation of an unlimited amount of cryptocurrency units. Transactions can also be performed almost instantly, which removes monetary and time-related constraints on the test runs. By using test networks, which have the same functionality as the main networks, it allows the prototype to be tested in an environment that is as close to the main network as possible without the risk of losing funds.

2

Technical background

This chapter aims to introduce the reader to the technical knowledge needed to understand the following chapters. The chapter begins by describing blockchains in general and features in the two blockchains used in the project. Thereafter, the current state of cryptocurrency trading will be summarized and following that is a description how network communication works today. The chapter concludes by introducing the concept of decentralized storage.

2.1 Blockchains explained

Blockchains are the most fundamental parts of this project. A blockchain is a public ledger, or more simply a database, of all transactions stored on a decentralized network of nodes [17]. This database is called a blockchain because the transactions included in the database are stored in linearly consecutive blocks where all blocks have a reference to the earlier block as can be seen in figure 2.1. To create new blocks, nodes gather up a set of transactions not included in any previous block and apply a cryptographic hashing function to solve a challenging cryptographic puzzle. This process is called mining [18] and when a new block has been created, the node that was able to first mine and distribute the new block into the network is rewarded with new coins. A fee has to be paid for every transaction and this is transferred to the miner of the block when the transaction is included in a new block. This fee and the rewards gained through mining a new block give extra incentives for miners of cryptocurrencies to continue with their activity [19].

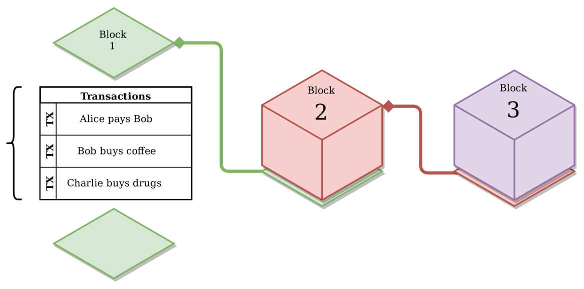


Figure 2.1: Visualization of a blockchain

2.2 The Bitcoin blockchain

Bitcoin is a peer-to-peer electronic cash system [20] and it currently has the largest market cap among all cryptocurrencies [21]. It was created by Satoshi Nakamoto in 2009 and has since paved the way for many other blockchain technologies and cryptocurrencies, especially a set of derivative cryptocurrencies that are commonly known as *altcoins* [22]. Some of these altcoins include Litecoin, Dogecoin and Peercoin [23].

Every Bitcoin transaction is processed through a simple scripting system called *Script* [24]. Script is based on a stack-like machine without any loops, intentionally making it *not* Turing complete. A language that is Turing complete is capable of performing any possible calculation or computer program [25]. Every Bitcoin transaction is performed with a script that validates the transaction. The transactions consist of one or more inputs and one or more outputs. For every transaction input, the bitcoin that will be sent must be taken from one or more existing outputs that have not been spent. These unspent outputs are called *Unspent Transaction Outputs* (UTXO).

The most common Bitcoin transactions are called *Pay to Public Key Hash* (P2PKH) [26]. In a P2PKH transaction, as seen in figure 2.2, if Alice wants to receive bitcoin from Bob, Alice hashes one of her private keys into a public key hash, creating an address. Bob then combines one or more of his available UTXOs as an input for a new transaction, which restricts further spending to Alice. The new transaction is locked by a script that states that the transaction can only be spent further if provided a key that when hashed will result in the public key hash that Alice provided. This gives Alice the bitcoins, as only she knows the key. When the transaction is confirmed, the input transactions are considered spent, and cannot be included in other transactions. Alice can later on spend the transaction in the same fashion.

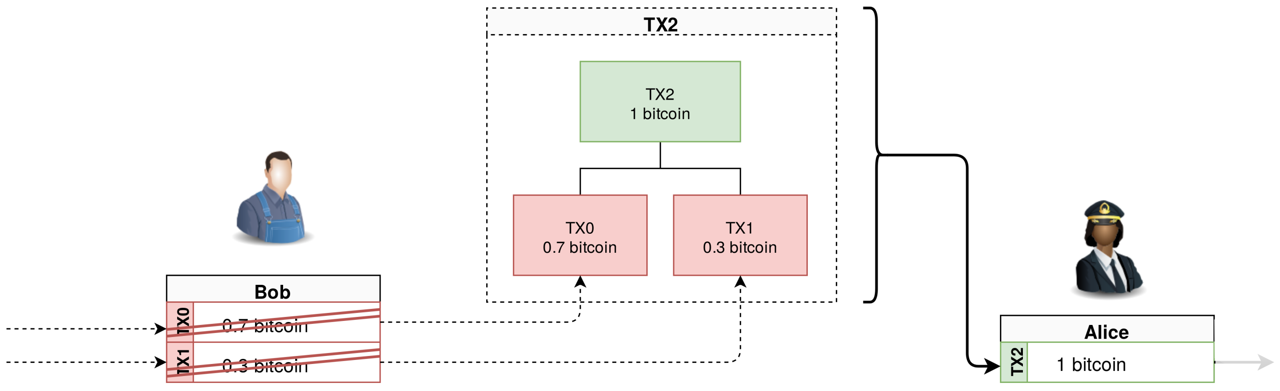


Figure 2.2: Visualization of a P2PKH

Another kind of transaction is called *Pay To Script Hash* (P2SH) [26]. Unlike P2PKH that creates a public key hash address that the bitcoin is transferred to, P2SH generates a script hash address that the bitcoin is instead transferred to. This script hash address contains a set of rules allowing anyone who can fulfill these rules and also give a proof of how to the script was created, through what is called a redeem script, possibility to spend from this transaction.

2.3 The Ethereum blockchain

The decentralized network Ethereum and its cryptocurrency ether is Bitcoin's main competitor. Ethereum was released in 2014 [27] and ether now has a market cap that is only surpassed by Bitcoin [21]. While ether is the native currency of Ethereum, the network also allows for deployment of so-called tokens [28]. Tokens can be compared to casino tokens in that they can be utilized as currency in a specific domain. Tokens do not replace ether, but can allow for additional functionality and provide added value. Ethereum is also a general purpose blockchain meaning that it provides a Turing-complete language [29] for programming smart contracts (for explanation of smart contracts see section 2.5) on the blockchain. Ethereum enables developed applications to run on the network with no downtime. Once a smart contract is executed by a node in the Ethereum network, the result is validated by the other nodes in the network [30].

2.3.1 Transaction costs

Since Ethereum is a public blockchain, measures have to be taken to prevent malicious users from slowing down the whole network by overflowing it. Ethereum solves this by having a set cost for each computational step in a transaction. A transaction could be simply sending ether to another account, deploying a contract or running a function on a contract. The cost for a computational step is measured in the unit gas, used internally in the Ethereum network [31]. A user proposes, in their transaction, a price for how much ether they are willing to pay for the unit of gas, and each miner decides if they are willing to accept that price. The user also sets a maximum limit on the amount of ether they are willing to pay. If the gas used by the transaction exceeds that limit, the executed code automatically fails and any changes made by the transaction are rolled back. However, the miner still gets all the ether provided for computing the transaction. If the gas amount used does not consume all the ether provided, the remaining ether is returned to the user. This effectively stops spam on the network as it would be too costly.

2.3.2 Ethereum and Ethereum Classic

Ethereum is a hard fork of Ethereum Classic, meaning they have the same genesis (first) block and partial history as can be seen in figure 2.3. But at one point there were changes to the chain leading to a fork of it. Nodes running the old version, now named Ethereum Classic, were no longer accepted [32]. The Ethereum hard fork was done as a response to a big attack where one person or organization got hold of 15% of the existing ether at the time [33]. Most nodes agreed that this should be reversed, so a hard fork was performed which took all the involved money and allowed users to withdraw them [34]. So today both Ethereum and Ethereum classic exists and they still share many similarities.

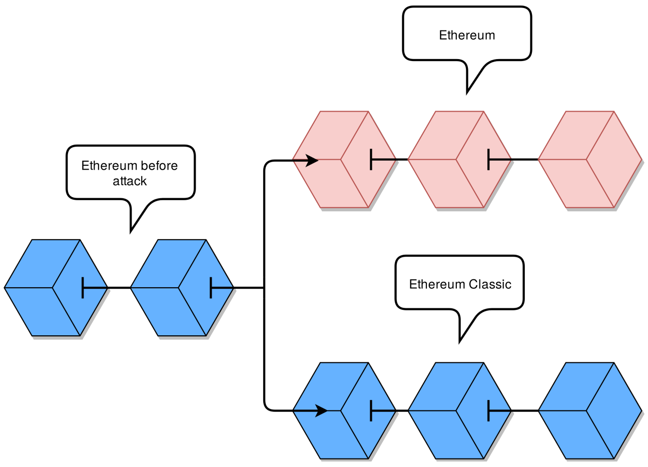


Figure 2.3: Visualization of a hard fork

2.4 Security of blockchains

Since blockchains are distributed among a set of peers, ensuring the security of the chain is of great significance. A particularly big issue that had to be tackled was the problem of double spending, which is when an actor is able to spend the same coins several times [20]. Blockchains solve this problem by a cryptograhic hashing method called proof-of-work, which requires the solving of cryptographic puzzles as outlined in section 2.1. These puzzles in practice combine all transactions to be included in a block, add a cryptographic nonce (arbitrary number used once [35]) to it and calculate a hash that start with a certain number of zeros, usually referred to as the difficulty of the block. The difficulty can be raised or lowered to make sure only a small subset of all possible values generated by the hash function fulfill the requirements. Since this subset can be quite small, in comparison to all possible hash values, lots of nonces need to be hashed before finding one that succeeds, requiring large amounts of computing power. However, trying to recreate the hash when verifying the new block can be done by already knowing the nonce, so verification is much faster. This results in that the creation of new blocks require a lot of computing power in comparison to the verification, and this should theoretically ensure a network without double spending [36].

However, the security of a blockchain cannot be deterministically guaranteed. If a single miner, or group of miners, control 51% of the computing power in the network, they could theoretically double spend in the network and cause other users to lose money. This has come to be known as a 51% attack [37]. After having sent funds to a different account, in order to accomplish a double spend, the miners could fork off the blockchain at a previous point and recreate one or several of the blocks by replacing the transaction to an address owned by someone else, by a transaction sending the same amount to an address owned by the attacker. When eventually the length of the blockchain in the forked network surpasses the length of the main network, the malicious miners could then merge their network with the main network and have their blockchain be the consensus blockchain. In theory, it is also possible to rewrite the entire blockchain history, through an attack called a Genesis Attack. It is also worth noting that an attacker having control of a proportion of the network that is lower than 51%, could perform a 51% attack. The risk of this happening gets lower the smaller the portion of the network that the attacker controls is [20, page 6].

Forking the blockchain and then trying to catch up with the length of the main blockchain have very high demands on computing power. And for each additional block, that demand increases. Therefore, after a block containing a transaction is mined, one would usually wait a certain amount of blocks until that transaction is deemed secure. For example, on the Bitcoin blockchain this number is 6 [38].

2.5 Smart contracts

Smart contracts, as outlined by Szabo [39], is an electronically executed contract with rules, requirements, and execution described in code. Szabo makes the comparison to a vending machine. A client inputs coins and receives goods according to the vending machine's programmed logic. Smart contracts are often an integral part of a the logic of a Blockchain and allows for programming involving currency, without the involvement of a central third party. The functionality of smart contracts widely differs depending on the blockchain they are deployed on. ETH offers programming smart contracts in a Turing complete language called Solidity while BTC offers much lower functionality in their Turing incomplete language Bitcoin Script.

2.6 Atomic swaps

An atomic swap is an exchange method, using smart contracts, that removes the need for third parties. This method also removes the need to trust the other part. The only part requiring trust is the decentralized network used to execute the smart contract.

This type of solution that uses smart contracts can also be useful in other cases where trust to one or more parts is required to make a system work as planned. As an example, this can be used to trade virtual goods safely if these goods are connected to some kind of token that can be managed like a currency. Another example could be a charity that can create a smart contract with a specification of what the collected funds should be used for. Using this, donators can be ensured that the collected funds is used for its intended purpose.

Another advantage of managing money with smart contracts is that a third party cannot steal money from or shut down the service as simply as a centralized service.

Malicious individuals or groups do not have a single point they can attack; they must attack the entire network to affect the service.

2.7 Cryptocurrency exchanges

There are a number of services that exchange cryptocurrencies for fiat money and vice versa. BTCX is an example of this. However, this particular service only sells bitcoin and ether [40]. This limitation where a user is only able to exchange fiat currencies for a limited set of cryptocurrencies is common for many existing exchanges.

Another type of exchange allows for exchange between different cryptocurrencies. One such exchange is *ShapeShift* [41]. To use ShapeShift, or any similar service, the users need to trust the service itself. However, it is easy for criminals to scam customers by pretending to run a similar service and not return any cryptocurrency to the customers that have already transferred their part of the transaction. Nevertheless, if an exchange service would start a contract of sale by first sending cryptocurrency to the customers, it is simple for a malicious customer to not return any cryptocurrency and thereby committing fraud.

2.8 Network communication

In the late 1960s, the creation of the *Advanced Research Projects Agency Network* (ARPANET) [42] became the first working prototype of the internet. Later in 1990, the first web browser was made which led to the modern World Wide Web as we know today.

The protocol known as *Hypertext Transfer Protocol* (HTTP) [43] quickly became the foundation of communication between clients and web servers on the World Wide Web. HTTP works by having the client, i.e. a web browser send requests to web servers that in turn returns information back to the client. This client-server mentality means that the clients must rely on the servers to get the information it wants.

Today, with the evolution of the World Wide Web, development of new technologies such as WebSocket has allowed for two-way communication between clients and the server. The traditional client-server mentality is no longer required, making it possible for clients to communicate directly with each other. This made it possible for new ways of distribution with interesting technologies such as IPFS (see section 2.8.3).

2.8.1 WebSocket

WebSocket is a communication protocol that was standardized in 2011 [44]. The WebSocket protocol enables a two-way communication between a client and a server.

This is done by keeping a persistent connection where the client and server can facilitate real-time data transfer between each other essentially making it a conversation. Both sides can close the connection when wanted [44]. Figure 2.4 illustrates a websocket session from start to finish.

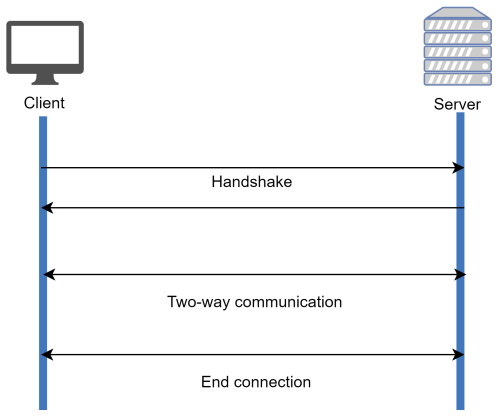


Figure 2.4: The flow of a communication channel using websocket.

2.8.2 NAT and relaying

The explosion of the internet led to a depletion of public IP addresses. A short-term solution to tackle this problem was the introduction of *Network Address Translation* (NAT). NAT works by having a router act as a middleman between the internet and a private network. It works by assigning the clients in a private network a unique private IP address which is used inside the private network. When a client wants to talk to the internet, it will go through the router, which uses a public IP address to communicate with the internet [45]. Therefore, the use of NAT leads to a slower depletion of public IP addresses as clients in a private network can use the same public IP address to communicate with the internet. Figure 2.5 illustrates the usage of NAT.

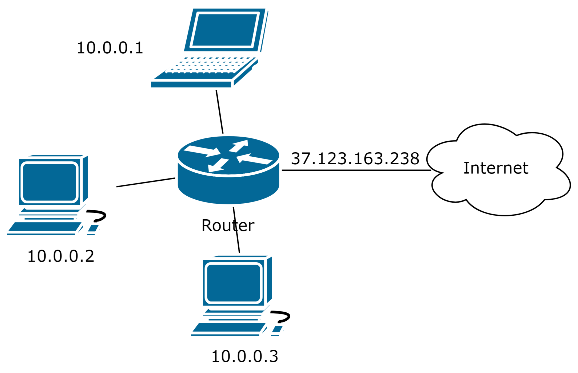


Figure 2.5: Example usage of NAT

The use of NAT has led to problems when two peers behind different poorly behaved NATs decide to establish a direct communication channel [46]. Due to various multimedia and peer-to-peer applications requiring a direct communication channel a solution had to be found.

A solution to this problem is using relaying and a protocol made for this is *Traversal Using Relays around NAT* (TURN). When a direct communication path cannot be found between two clients they can opt to use a TURN server as a relay. The TURN server will then forward the data sent from one client to the other as seen in figure 2.6 [46].

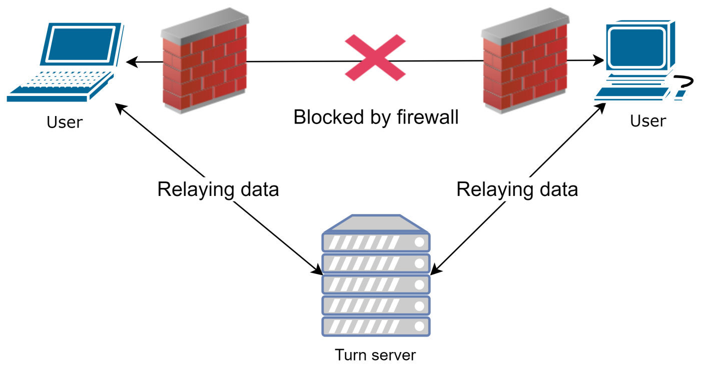


Figure 2.6: Two computers connecting through a TURN server

2.8.3 IPFS

IPFS is a new technology that provides a solution for information access where servers do not have to distribute content to clients. Instead, it makes it possible for the content to directly be stored in clients and allows clients to retrieve content from each other. In other words, IPFS is a peer-to-peer distributed file sharing system that combines popular concepts, such as distributed hash tables, from other peer-to-peer systems like BitTorrent and Git [47].

Because IPFS is a peer-to-peer network, it consists of clients also known as the nodes of the network. These nodes need to be identifiable by each other. IPFS solves this by generating a key pair consisting of a public and private key. The public key is hashed and used to identify the node. When two nodes want to communicate with each other, they exchange their public keys, which are then hashed and checked with the node [47].

Nodes in the IPFS network can communicate with any transport protocol available [47]. I.e. it can use WebSockets to connect to other peers. For nodes to be able to find other nodes and share data, IPFS uses a *Distributed Hash Table* (DHT). DHT is essentially a key-value store that is distributed in a network [48]. It allows nodes to find other nodes through the hash table. If a direct connection cannot be established between two nodes, IPFS will use a third node to establish a connection. The third node will act as a TURN server [49].

2.9 Distributed database

With the evolution of the internet, organizations have an easier time growing internationally, which means that there are benefits to have data stored in multiple locations, providing a good alternative to the traditional database solution with a central server [50]. The solution to this problem is to make a *Distributed Database System* (DDBS) where the database is stored on multiple servers that can be located in different places and then connected together through a network [51]. To ensure that a database is up to date, it can rely on a technology called replication. It works by having every server replicate every change that happens in another server. By doing this, the distributed database can ensure that it is consistent throughout all servers.

Today, there are many advanced solutions for distributed databases making it possible for the DDBS to work in an independent network. An example of this is the blockchain technology as explained above.

2.10 Test automation

Test automation is a tool that can be used to reduce the introduction of new bugs. As soon as new functionality is added, the new code is automatically run against a set of tests to make sure that it does not break already functioning parts. However, this tool is difficult to use with APIs as their dependency on outside actors makes it impossible to predict the exact results given certain parameters.

A common metric for test automation is code coverage. It is a measure for the amount of program code that is executed by the tests. The optimal code coverage is when all code is executed during the automatic tests. It should be noted that having optimal code coverage does not guarantee that the code is bug-free; it simply expresses that all code has been tested. The code might still fail on some input that is not tested.

3 Methods

This chapter details the methods used for creating the prototype. The chapter is divided into four sections: development, requirements, design, and implementation. The development section covers the development process and rest of the sections roughly cover the same main topics, though the implementation will go more in-depth in what methods were used. The choices made in the implementation section will be motivated by what is mentioned in the requirements section, whereas the design section lays an outline of how the requirements are to be met.

3.1 Development

The development of the project consists of two parts, deciding a protocol and implementing a prototype using the protocol. Requirements are first set and explained further in section 3.2.

Based on research in the blockchain domain, a choice is made for a protocol that complies with the requirements. Then a minimal-viable-product is built to ensure that the protocol works and meets the requirements.

When a working implementation of the protocol is complete, the development of the prototype begins. The development of the prototype is divided into four categories: database, communication, blockchain and user interface.

3.2 Requirements

The platform is the proof-of-concept prototype that is developed during the course of the project. The platform is divided into two different areas: the user interface and the actual exchange that allows for placing and accepting bids as well as executing exchanges.

3.2.1 Exchange method

The platform will support a method of exchanging cryptocurrencies between two parties. The exchange method should at a minimum fulfill the following requirements:

- It should be atomic.
- It should be decentralized.

- It should work for exchanges between different cryptocurrencies.

It should be atomic, meaning that neither party should be at risk of losing their funds. In any realized exchange, both parties should receive their respective funds. If the exchange is not realized, the funds should instead be returned to their respective owner.

For a secure exchange to take place, the transactions need to occur simultaneously or rely on a third party ensuring the validity of the exchange. A third party is typically a person or a company that both parties trust. Since the premise of the platform is that it should be decentralized, the exchange method needs to fulfill that requirement as well.

Lastly, the method will need to work across blockchains. It is not required to work across the set of all blockchains, it should, however, be possible to make an exchange across at least two unconnected blockchains.

3.2.2 Bid database

The platform will support a way of publicizing bids as a mechanism to find potential trading partners in the form of a publicly available database. The bid database should at minimum fulfill the following requirements:

- It should allow users to post bids.
- It should allow users to accept bids.
- It should be decentralized.

Bids that are posted to the database should be accessible by all users using the prototype. When a bid gets accepted it should no longer be accessible by other users. Ensuring that the database is decentralized is required for building a decentralized prototype.

3.2.3 Message passing

To support the exchange method, users need some way to communicate with each other to exchange the data necessary to complete a transaction. The requirements for message passing that need to be fulfilled are:

- It should be automatic.
- It should not be dependent on any blockchain network.
- It should be decentralized.

That the message passing should be automatic means that a user should only have to accept or post a bid and type in their private key to complete an exchange. The user should not have to care about technical exchange details such as creating and validating the smart contracts. To allow for the extension of support for other cryptocurrencies, the message passing should not depend on any blockchain network.

The same implementation of message passing should work for all implemented cryptocurrencies. The message passing also needs to fulfill the requirement of being decentralized as not to compromise the decentralization of the prototype.

3.2.4 User interface

The user interface should be designed in a way that makes it user-friendly and fault tolerant. The user interface should at a minimum fulfill the following requirements:

- It should use real world concepts.
- It should prevent the user from performing common errors.
- It should display errors in a clear manner and the user has to be able to recover from them easily.

These requirements are based on *Nielsen and Molich's 10 User Interface Design Guidelines* [52]. It makes it easier to interact with the prototype if the user can apply real world experiences. Therefore, the user interface should incorporate real world concepts into the design.

If problems that are easy to predict are prevented from happening, the user will not have to deal with these problems. Therefore, the user interface should prevent the user from performing common errors. Nevertheless, if any errors occur, they should be displayed in a clear manner so that it is easier for the user to find out what went wrong. The user should also be able to recover from the errors easily, so that a single mistake will not have any far-reaching consequences.

3.3 Design

In the immediately preceding section 3.2, the requirements of the platform are described. The approach to fulfill these requirements are explained in this section. The structure follows a similar pattern as section 3.2, and for each requirement, a design is outlined and explained.

The general design of an exchange in the prototype is visualized in figure 3.1. This design is based on using the exchange method outlined in 3.2.1. The users should only have to be concerned about posting, finding and accepting bids while the actual exchange is done automatically by the client.

3.3.1 Exchange method

The implementation of the exchange method is based on atomic swaps. To facilitate atomic swaps, it is possible to use *Hashed Timelock Contracts* (HTLC) [53]. Such a contract needs to be able to specify a destination, a hashed password, a time limit and hold funds. It also needs to have two functions: 'claim' and 'refund'. The funds in the contract will be sent to its destination if 'claim' is called with a password, that when hashed, matches the hashed password in the contract. If the time limit passes, the creator of the contract may call 'refund' to return the funds. These contracts

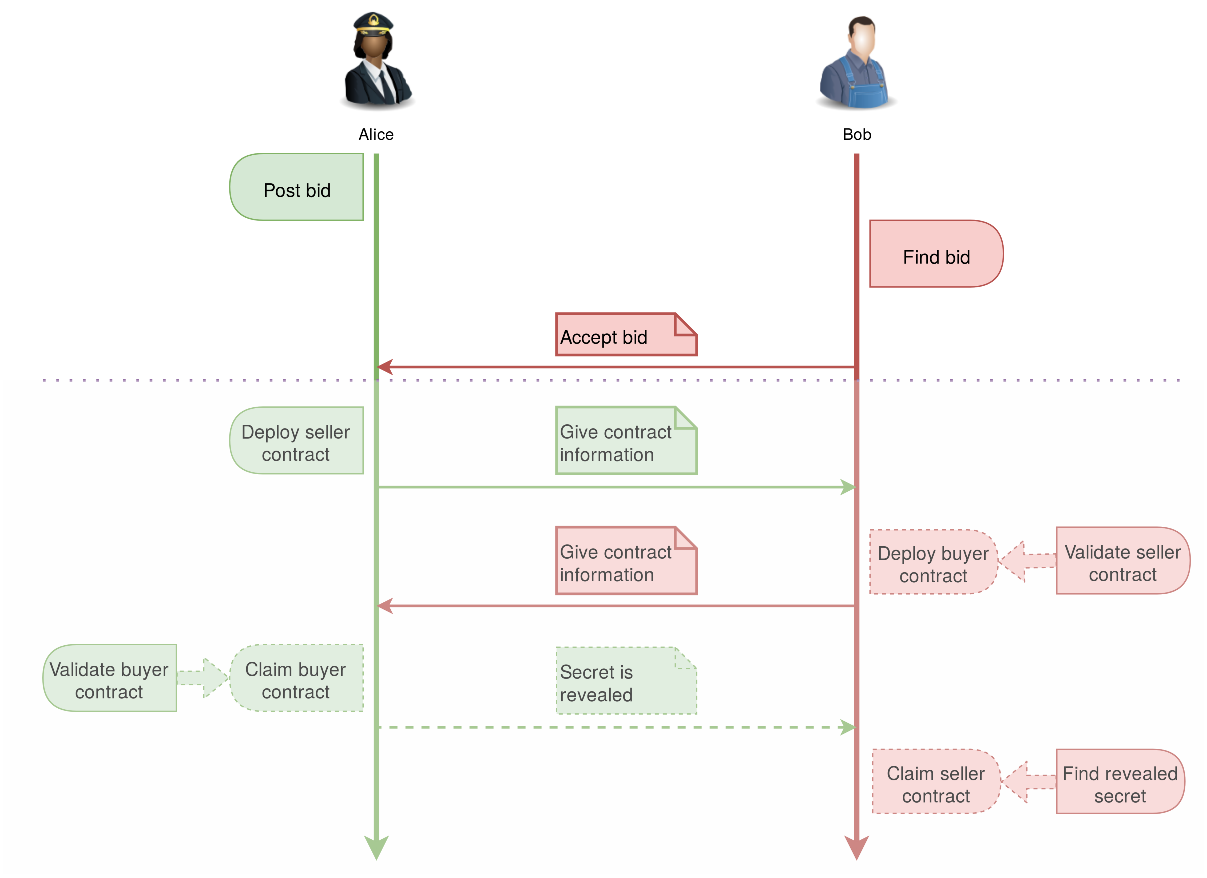


Figure 3.1: Visualization of an exchange using the prototype

can be used for atomic swaps, as outlined in the example below.

HTLC example

1. Alice decides on a password and hashes it
2. Alice creates a HTLC with Bob as its destination
   (a) With the hashed password
   (b) With the specified amount Bob wants
   (c) With a reclaim time limit
3. Bob validates Alice's HTLC
4. Bob creates a HTLC with Alice as its destination
   (a) With the same hashed password as on Alice's contract
   (b) With the specified amount Alice wants
   (c) With a reclaim time limit
5. Alice validates Bob's HTLC
6. Alice claims Bob's funds from his HTLC by entering her password
7. Bob now knows Alice's password and claims the funds from her HTLC
8. Alternatively enough time passes and they reclaim their money

If Bob is able to guess Alice's hashed password then Bob can claim Alice's funds without putting up his own contract. As such this type of exchange relies on the difficulty of the hash function, since the hash is visible for anyone to try and brute force. Furthermore, if Bob and Alice cannot trust each other, they will need a third party to mediate the exchange. The third party in this instance is the respective blockchains of the cryptocurrencies being exchanged. Thus, it also fulfills the requirement of being decentralized. HTLC does not work for all cryptocurrencies, it is however proven for both Bitcoin [54] and Ethereum [55], which is enough to fulfill the last requirement. The last requirement is covered in more detail in the following section 3.3.2.

3.3.2 Smart contracts

To be able to exchange cryptocurrencies between two blockchains, using the method described in the previous section 3.3.1, it requires the blockchains to support smart contracts. An HTLC has very basic functionality that most smart contract implementations support. The most important function is the ability to hash, which must be the same between the two blockchains where cryptocurrency is being exchanged. While Ethereum supports several hashing algorithms, Bitcoin's Script Opcodes only supports SHA-256, RIPEMD160 and SHA-1 [56], which makes it necessary to use one of those three to hash the password when performing exchanges between the blockchains. SHA-256 is the stronger algorithm of the three [57] and therefore it is the one used in the prototype.

3.3.3 Contract validation

Because smart contracts are public information, it is only necessary to know the address of the contract or the transaction holding the contract to get enough information to validate it. By creating the same contract locally, in the shape that you expect it to be, it is possible to compare it to the contract deployed on the blockchain and thereby validate it.

3.3.4 Design of bid database

The database system in the prototype consists of three different databases: **GlobalDB**, **StatusDB** and **LocalDB**. The GlobalDB is used to store bids from the users and it can be accessed by anyone using the prototype. The bids stored in GlobalDB include data about the currencies involved as well as the proposed exchange rate, along with the address to the communication channel over which the message passing will take place, as detailed in section 3.3.5. StatusDB is another globally accessible database used to store the state of a bid. There are three states a bid can have, active, pending, and finished. The third database, LocalDB, is a locally accessible database used to store the user's own bids and bids the user has accepted. The reasoning behind using three databases is discussed in section 3.4. Figure 3.2 shows how the database is designed.

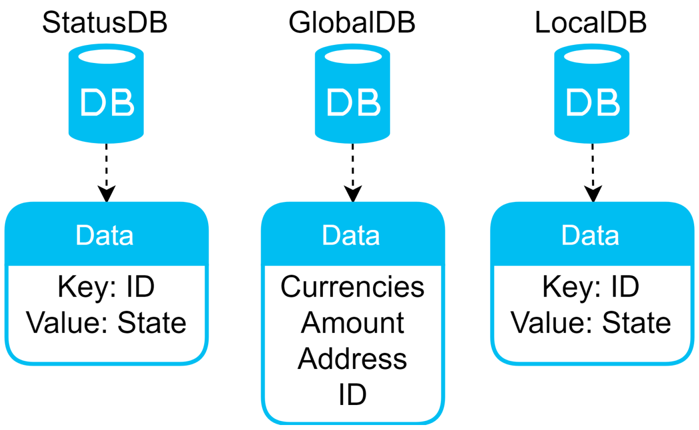


Figure 3.2: Design of the database including data stored in each of them.

3.3.5 Design of message passing

As a user publicizes a bid, a new communication channel is created. The communication channel is essentially a new database that both users involved in the exchange listen to. This means that any message written in the channel is read by both users. The second party in the exchange starts listening to the new channel as soon as the user accepts the bid. The motivation is to ensure that the exchange is automatic, as laid out by the requirements. As soon as both users listen to the same channel, information necessary to complete the exchange is passed through the channel as JSON objects mapping the values so they can be parsed by the second part. The message passing can be generalized as the only difference between different cryptocurrencies is the information sent out via these JSON objects, but the general steps are the same.

3.3.6 User interface

The user interface uses real world concepts by displaying bids in a way that corresponds to how a bid is traditionally represented. It will prevent the user from performing common errors by enforcing form validation. The user interface will manage errors by displaying a description of them in a clearly visible error message box. It will also roll back any progress that has been made in the action that caused the error so that the user can try to perform the same action again or perform another action in the user interface.

A number of views should be included in the user interface. Each view should correspond to a specific functionality that the user can use. These should include a view for adding new bids, a view to browse among other users' bids and views for displaying bids that have been added and accepted by the user. It should also contain a view that displays the current balance of the wallets contained within the user's blockchain software. Additionally, there should be a *settings* view to specify the paths to the blockchain software that the prototype supports.

3.3.7 Test automation

Test automation is used to ensure that the compilation works after an update. The automatic tests run after a new feature is added into the program to make sure that it does not break any existing code. Due to the limitations that test automation has on testing code with external APIs, the tests will run only in the user interface and other parts of the code that do not use external APIs.

3.4 Implementation

This section gives a detailed view of the implementation choices made based on the requirements in section 3.2 as well as the design detailed in section 3.3. The development starts on a private Ethereum network and as the prototype develops, it is also deployed on the public test networks of Ethereum. Priority lies in having exchanges work within a single network and then make an exchange across different networks. The exchange across different networks will also start as an exchange between two Ethereum networks and then expand into cross-chain swaps between the Ethereum and Ethereum Classic test networks.

Although steps are taken towards implementing support for exchange with Bitcoin, no testing is done since the integration with the rest of the platform is incomplete at the time of writing.

3.4.1 Exchange method and atomic swaps

The exchange method, as mentioned in section 3.3.1, is based on HTLC. Due to the several different types of cryptocurrencies, a unique implementation is possibly required for each and every cryptocurrency. However, a central trade handler, to ease the integration, is implemented. In the following two sections, the implementation for Ethereum and Bitcoin is explained.

3.4.2 Atomic swaps on the Ethereum platform

The HTLC in Ethereum is implemented using smart contracts written in Solidity, which is the only stable option [58]. A contract is deployed on the blockchain with ether, or tokens, and has the two functions: 'claim' and 'refund'. To successfully run the 'claim' function, the user has to provide the secret that the contract is locked with. The 'refund' function cannot be executed until the timeout has expired. If a contract is claimed or refunded, the contract self-destructs and is removed from the blockchain and sends the funds to its respective destination targets. The destinations are predetermined and recorded on the contract.

If the contract is claimed, an event with the secret is also emitted and recorded on the blockchain. To access the event, blocks from the point of deployment forward are searched looking for the event. An execution of HTLC on Ethereum can be seen in figure 3.3 and figure 3.4.

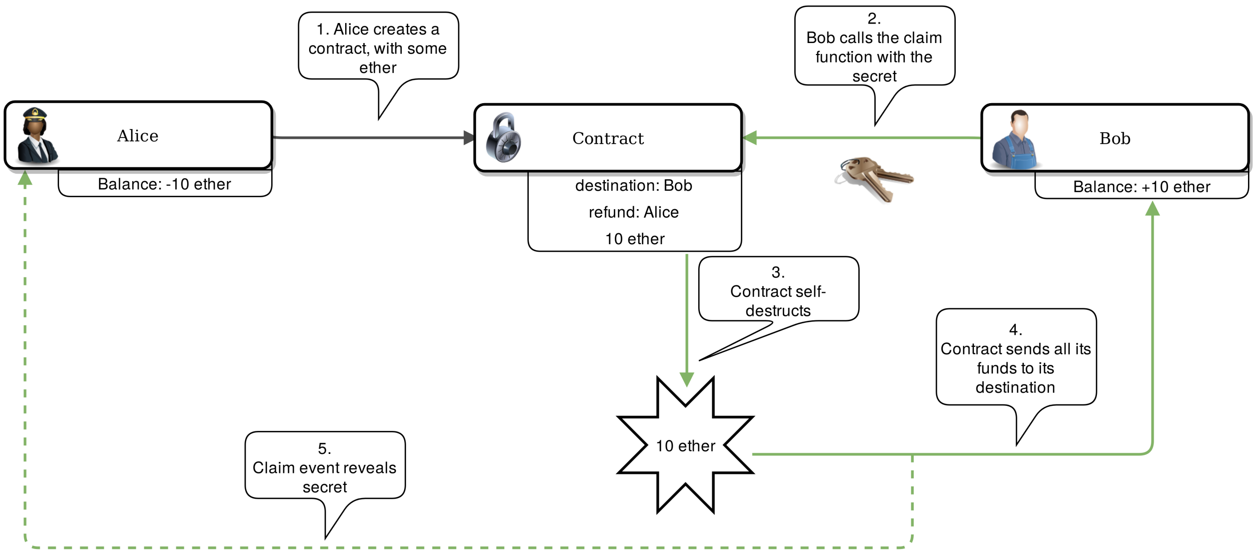

1. Alice creates a contract, with some ether
2. Bob calls the claim function with the secret
Alice
Balance: -10 ether
Contract
destination: Bob
refund: Alice
10 ether
Bob
Balance: +10 ether
3. Contract self-destructs
4. Contract sends all its funds to its destination
10 ether
5. Claim event reveals secret

Figure 3.3: Example of a successful HTLC in Ethereum

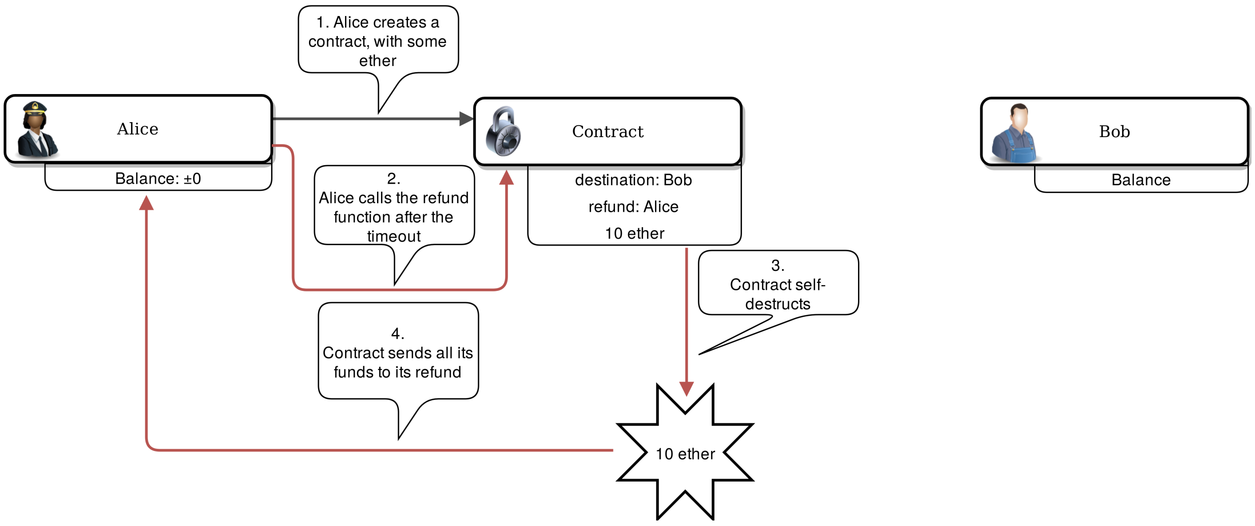

1. Alice creates a contract, with some ether
Alice
Balance: ±0
Contract
destination: Bob
refund: Alice
10 ether
Bob
Balance
2. Alice calls the refund function after the timeout
3. Contract self-destructs
4. Contract sends all its funds to its refund
10 ether

Figure 3.4: Example of a refunded HTLC in Ethereum

The HTLC works slightly different when handling tokens of the Ethereum network as it requires an intermediate step. The tokens have to be sent to the contract after its creation and instead of the self-destruct sequence, where the tokens on the contract is sent afterwards, the tokens must be sent before self-destructing. The reason why the contract still self-destructs is that it makes the function less costly, as some ether is refunded when the contract self-destructs. This is to incentivize users to self-destruct contracts they do not use, as otherwise they take up needless space on the blockchain [59].

Validation of contracts deployed on the Ethereum network is done to deny malicious users. The validation checks that the runtime bytecode of the deployed contract matches a previously saved runtime bytecode. This ensures that the deployed contract is indeed a HTLC. The next step of the validation is to check that all the variables of the contract are assigned with correct data. These variables include the destination, amount of funds, and the hashed secret. This is to ensure that the correct user receives their entitled part of the transaction. In the case of a token contract, it will also validate that the contract is of the predetermined token type.

3.4.3 Atomic swaps on the Bitcoin network

The HTLC is implemented in Bitcoin using Script as it is the only option for creating smart contracts on Bitcoin. Every transaction has a script embedded into it that dictates the conditions for spending the funds further. The scripts are written using Script. For more complex transactions, it is possible to output transactions to a P2SH. A transaction outputted to a P2SH can only be spent further by a user if provided a script that has a hash value equal to the P2SH as well as a stack that resolves the script to true as seen in figure 3.5 in the case of a user claiming the funds, and figure 3.6 in the case of a refund.

As the spending of a transaction is up to the spender, the contract needs to ensure that the claiming and refunding operations are exclusive to the respective parties. This is done similarly to how standard transactions are done in Bitcoin, by requiring a signature that has a hash value that evaluates to a public key recorded in the script. The signature needed is different depending on if one is claiming or refunding, split by an if-statement in the script. If a claim is issued, the provided secret will be put in the transaction. Starting from the block where the HTLC transaction was included, every transaction is searched to find the secret.

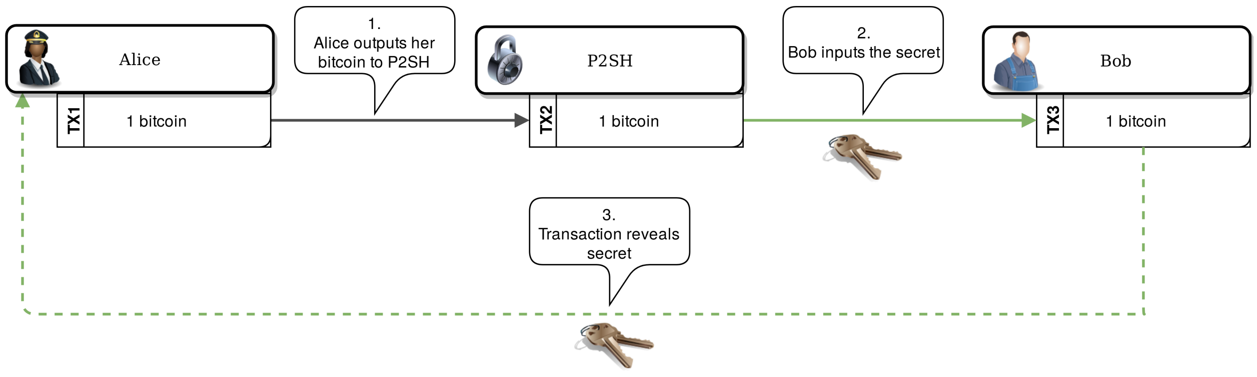


Figure 3.5: Example of a successful HTLC in Bitcoin

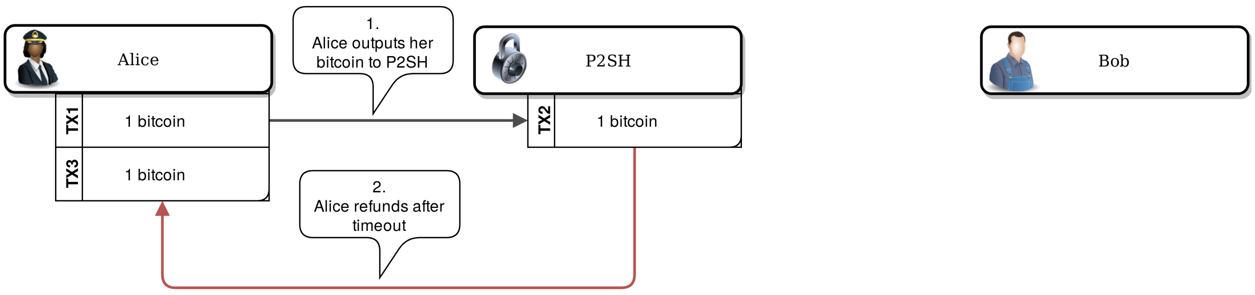

Alice
1.
Alice outputs her bitcoin to P2SH
P2SH
Bob
TX1
1 bitcoin
TX2
1 bitcoin
TX3
1 bitcoin
2.
Alice refunds after timeout

Figure 3.6: Example of a refunded HTLC in Bitcoin

Validation of the contracts is done by sharing the transaction id, and other information necessary to recreate the P2SH, to the other party. The transaction id can be used to request the transaction information from the blockchain, which can then be used to validate that the value and the destination address is correct, and that it is currently unspent. The party wanting to validate the contract recreates the HTLC by the specifications provided in the prototype and compares the P2SH to confirm that the contract is valid.

3.4.4 Bid database

The implementation of the database system is based on the framework OrbitDB. It is a distributed, peer-to-peer database that is built on top of IPFS [60]. OrbitDB supports various kinds of databases including key-value and log databases. This makes OrbitDB an excellent choice for the decentralized prototype.

The bids are stored in GlobalDB as JSON objects for simple parsing to extract the data. The databases also have listeners implemented that triggers when the databases are replicating. Thereafter, the listeners trigger the user interface to update. This ensures that the users will always have the most recent bids available. In figure 3.7 the database architecture is shown.

While implementing OrbitDB, two critical issues were noted and had to be dealt with. The first one is that OrbitDB does not completely support Windows leading to an error when initializing a database. The solution to resolve this was to implement a database creator that runs on a Google Chrome instance. The database creator initializes the database before being used in the prototype for Windows machines.

The second issue is that write access to a database cannot be restricted to a specific set of data in OrbitDB. If all users were given write access, it would result in malicious users being able to manipulate other users' bids. Therefore the database system consists of three databases as told in section 3.3.

To solve the issue it was decided to make the database GlobalDB an append-only log. By having GlobalDB as append-only, data cannot be removed or changed which means that a separate database is needed for keeping track of the state of a bid and this is why the key-value database StatusDB was made. When a user accepts a bid,

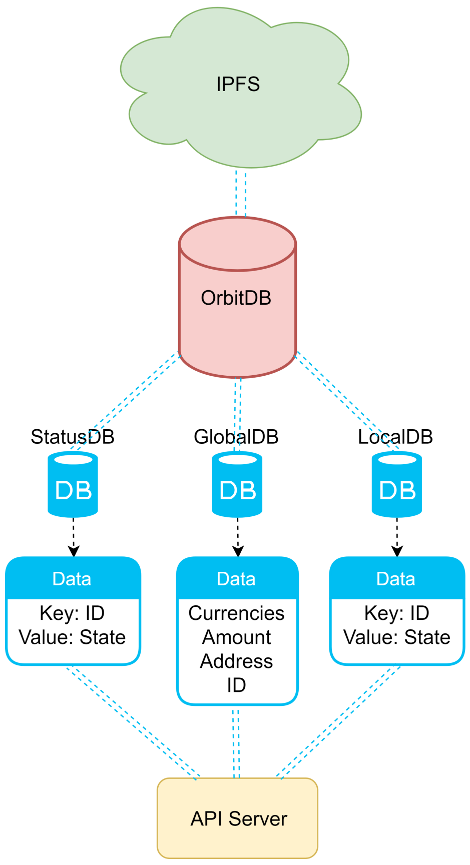


Figure 3.7: The architecture of the database using OrbitDB.

it also needs to be stored to ensure that the exchange works between different sessions in case an exchange cannot happen immediately. It also does not make sense to share this data with uninvolved peers, which is why the local database LocalDB was made.

3.4.5 Message passing

As a bid has been accepted by a user, the two parties involved in the exchange need to be able to communicate with each other to complete the steps defined in section 3.3.1. For this, OrbitDB is also used as it is judged to be an effective enough tool and would work well as it has already been implemented for the bid database.

As explained in section 3.3.5, as soon as a user publicizes a bid, a new communication channel is created. The communication channel has been implemented as a new database that both users involved in the exchange listen to. As soon as both users listen to the same channel, the information necessary to complete the exchange is passed through the channel as JSON objects. The values are mapped so that they can be parsed by the second party in the exchange.

There is a three step process involved to complete the message passing, as shown in figure 3.8. In the first step Bob, who accepts the bid, sends the address of where he wants to receive the funds. Alice then receives Bob's address and sends over her address, the hash as well as the contract address to the smart contract she set up on her network. Bob completes the last step by sending the contract address to the smart contract on his network.

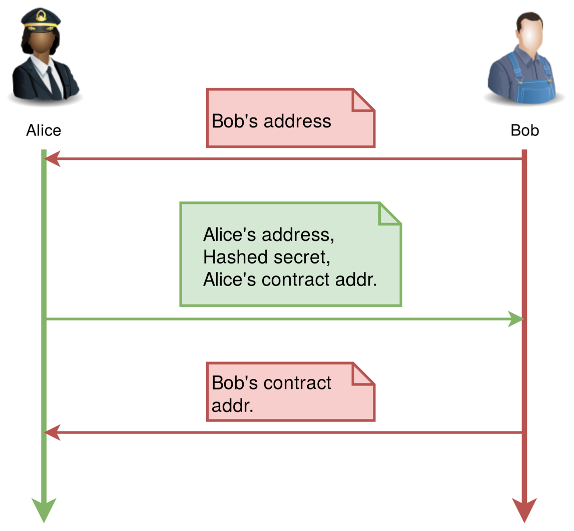


Figure 3.8: Message passing to complete an exchange.

3.4.6 Blockchain software

To participate in, send and receive coins in a blockchain network, a blockchain software tool needs to be used. To connect to the Ethereum network there are several alternatives. The command line interface Geth [61] is used in our prototype, and the users of our prototype are required to do the same. Bitcoin Core is an example of such a software in the Bitcoin network and it also is the reference implementation [62]. Bitcoin Core consists of both node software that validates the blockchain and a wallet software which allows a user to keep track of their transactions.

Because of the cost of downloading and maintaining a node, in terms of memory size and bandwidth, some networks have clients that allow users to download a lightweight node. Geth provides three options for synchronizing with the blockchain: full (default), fast and light. In terms of size these correspond to a local database of about 300 GB, 30 GB, and 0.5 GB respectively. While a light node is more lenient on the computer, there are several drawbacks. As it does not store all information locally, it has to query other full nodes for information. It is therefore much less secure, as there is a risk of getting information from malicious nodes. It is up to the user to contemplate the risks. Also, it is not possible to solo mine using a light node, as it does not have enough information to do so. To speed up the development, light and fast nodes where used when connected to the test network and full nodes where used on private networks.

3.4.7 User interface

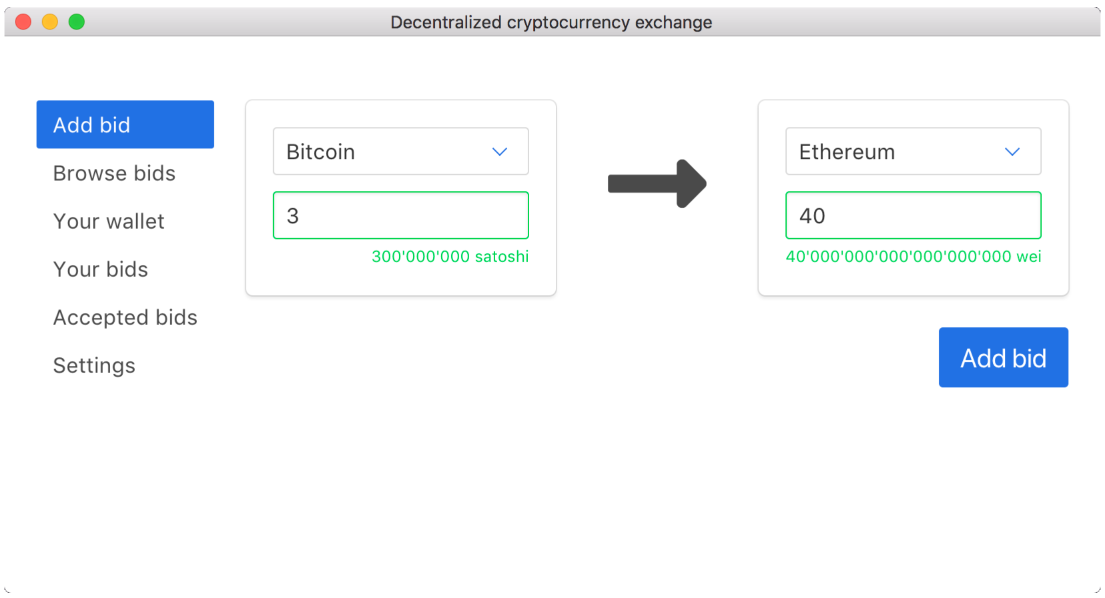
The user interface is implemented as a desktop application as seen in figure 3.9. It is developed as an HTML5 web application and executed as an Electron application. It would be possible to run the user interface through a browser but Electron is used as it makes the user interface behave like a desktop application.

Most parts of the client are implemented in Elm, which is a purely functional, type safe, and exception free programming language with a syntax similar to Haskell. The Elm code is run by first being transpiled into Javascript before Electron executes it. Elm was chosen instead of plain Javascript as its features help to easier fulfill the requirements of the user interface regarding error handling [63].

Javascript running through Node.js has been chosen as implementation language for the communication between the user interface and the decentralized networks as it has a large ecosystem of libraries for communication with different blockchains [64].

Electron executes on a legacy version of Node.js. This means that certain language features available in recent versions of Node.js are unavailable for Javascript programs executing through Electron. This issue led to the separation of the prototype into a user interface process and a server process that manages communication to the decentralized networks. This server will henceforth be called the *API server*.

Communication from the user interface to the API server is modeled as a set of actions where each action corresponds to a HTTP request that the user interface can perform. This means that it is possible to test the API server without the user interface by using an arbitrary HTTP client, such as a web browser.



Bitcoin
3
300'000'000 satoshi

Ethereum
40
40'000'000'000'000'000'000 wei

Figure 3.9: The implemented user interface.

3.4.8 Test automation

For the user interface, the automatic tests first transpiles the elm code to Javascript. Elm's transpiler is relatively strict, so it catches a larger portion of errors compared to most other compilers [65]. After the code has been transpiled, the actual unit tests are run. The testing framework *elm-test* is used to write and run these unit tests.

The parts of the API server that manages local user data are also tested. These parts are tested because they do rely on a certain amount of logic to handle communication with the user's file system that has to work correctly while at the same time they do not rely on any external API. The testing framework *jest* is used to write and run these unit tests.

The user interface and API server tests are executed in parallel. This is not an issue because the testing frameworks do not interfere with each other. The automatic tests are performed whenever new code is included to ensure that new features do not break the parts of the code that have already been tested.

4 Evaluation and Results

This chapter presents the evaluation and results. The evaluation sections describes how the prototype is evaluated and the results section presents the results from the evaluation. Table 4.1 offers a summary of all the tests.

Table 4.1: Summary of tests

Name	Evaluation	Result	Type	Description
Test A	4.1.1	4.4.1	Quantitative	Success rate of exchanges
Test B	4.1.2	4.4.2	Quantitative	Delay of an exchange
Test C	4.1.3	4.4.3	Quantitative	Cost of issuing an exchange
Test D	4.2.1	4.4.4	Qualitative	Trading pairs available

4.1 Quantitative Evaluation

The quantitative evaluation is done by executing tests on the prototype. The tests are performed by executing exchanges on the client and deriving information about their state by using blockchain software on the respective cryptocurrency involved in the exchange [66, page 41-42][67].

4.1.1 Test A : Success rate

The success rate is the amount of successful exchanges compared to failed ones. A successful exchange is an exchange where both parties end up with the predefined funds as outlined in the bid. Any other result is considered a failed exchange, including bids that need to be refunded.

The test is conducted by performing several exchanges between Ethereum ether and Ethereum Classic ether on the test networks Ropsten and Morden using the Geth client, along with 3 exchanges between two private Bitcoin chains. The result is then checked on the accounts to see if the trade succeeded or failed.

4.1.2 Test B : Delay

The delay is measured as the time it takes for a transaction to transpire after it has been accepted. If the exchange is issued through smart contracts, this is equivalent to the time it takes to execute the contracts on the blockchain and any delay associated with the database, which results in the time it takes for the different contract transactions to be included in blocks.

The delay is measured as the average in the relative time between two blocks, measured as the later block's timestamp subtracted by the former one's. For an exchange, two relative block heights are counted; one for the buyer and one for the seller.

The test is conducted by simulating several exchanges on the client and then recording the first block and last block respectively for the contracts deployed on the blockchains as shown in figure 4.1.

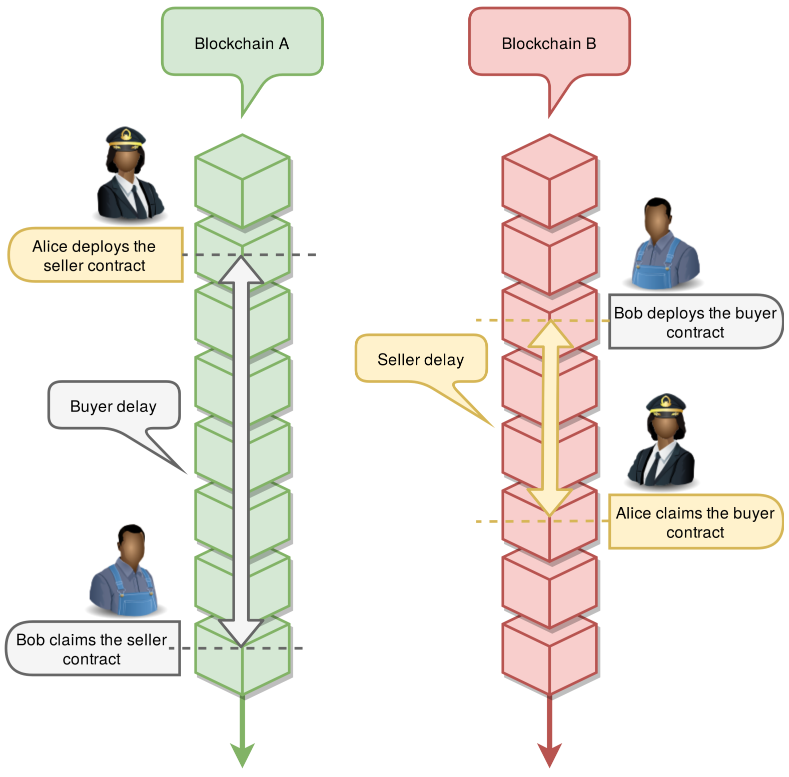

Blockchain A
Blockchain B
Alice deploys the seller contract
Bob deploys the buyer contract
Seller delay
Buyer delay
Alice claims the buyer contract
Bob claims the seller contract

Figure 4.1: Delay visualized

4.1.3 Test C : Cost

The cost refers to the cost required to activate, accept, or cancel a bid. As part of the exchange is made through smart contracts, there is a significant cost involved. A low cost allows for the users to engage in more trades, which in turn can increase the liquidity of the exchange. This cost is however fixed in terms of gas and therefore only one test run is needed to confirm the cost and an estimate is made on what the equivalent is in American dollars.

For transactions and deploying contracts on Ethereum and Ethereum Classic a gas price needs to be decided on. While the gas used is constant, the gas price can fluctuate depending on what miners are willing to accept. To estimate the gas price on the main network, Gas tracker [68] and ETH Gas Station [69] is used. They are two websites that list up to date information about how much users are currently paying for gas. The tests' gas cost is decided on the day of the tests based off on what is recommended on the websites.

4.2 Qualitative Evaluation

Only one qualitative test is performed on the prototype, and that is to confirm the possible trading pairs. Of the possible trading pairs, only one test iteration needs to be executed to confirm that it works as the success rate is not measured in this section.

4.2.1 Test D : Trading pairs

Trading pairs refer to the different currencies that you can exchange for and with. Exchanging within a network is not interesting, so therefore the total amount of possible trading pairs is not the interesting part. Rather exchanging cryptocurrencies between different blockchain networks is what is interesting as it makes the cryptocurrencies more accessible.

The test is done by making at least one successful exchange between two different cryptocurrencies to confirm that it works between those two. As this is mostly a qualitative measurement, a discussion is followed on how difficult it would be to put cryptocurrencies in new trading pairs or add new cryptocurrencies.

4.3 Testing environment

The tests are conducted by exchanging cryptocurrencies using the prototype and then deriving information about them using node software connected to the respective blockchains involved in the trade.

The two blockchains used to perform the tests between ETH and ETC are Ropsten (Ethereum test network) and Morden (Ethereum Classic test network). Fast nodes were used on both chains. For the the Bitcoin tests, private networks connected on a single computer were used, running full nodes with the Bitcoin Core software.

4.3.1 Hardware specification

The tests for ETH-ETC are run using two separate computers that perform the trades between them. The choice of computers is limited since OrbitDB has proven unreliable or outright incompatible with cross-platform databases. Furthermore, requirements on storage capacity exist since the chains involved in the tests have to be stored on both computers. In table 4.2 the hardware specifications of the computers are displayed.

Table 4.2: Hardware specification of the computer used for ETH-ETC.

Specification	Computer 1	Computer 2
Storage	206 GB	240 GB
CPU	I5-6300U	I7-4690K
RAM	8 GB	16 GB
Operating System	Windows 10	Windows 10

The tests for BTC-BTC are run using only one computer that performs the exchange. The hardware specifications are detailed in table 4.3.

Table 4.3: Hardware specification of the computer used for BTC-BTC.

Specification	Computer 1
Storage	227 GB
CPU	I7-3610QM
RAM	8 GB
Operating System	Ubuntu 16.04

4.3.2 Network specification

Network requirements exist when testing on private test networks. Ethereum's node manager Geth cannot find IPv6 peers. So two peers both on IPv6 will not be able to find each other. This is a non-issue on public networks since there are other peers to connect to in that case.

The computers used to execute the tests on ETH-ETC was connected to the same private network. A VPN provider was used on one computer to simulate trades between two different networks.

4.4 Results

In this section, the evaluated results are presented. In total, six tests were executed on the test networks of Ethereum and Ethereum Classic. This number was the result of time constraint. Although more tests had previously been run on private networks to ensure that the prototype works. Therefore, it was deemed to be enough to verify that the prototype works as desired.

On the Bitcoin network, 3 tests were executed. This was also due to time constraints and several less rigorous tests having been performed, showing similar results.

4.4.1 Test A : Success rate

After performing the 6 tests between Ethereum and Ethereum Classic all were proven to be successful. This yields a success rate of 100%.

Between the two Bitcoin networks, 3 tests were performed. 1 test passed with both parties being able to claim their fund. However the other 2 tests only partially succeeded in that only the first claimer was able to claim, yielding a succees rate of 33%.

4.4.2 Test B : Delay

The average delay for the seller was measured to 123 seconds and the average delay for the buyer was 322 seconds. The full data is outlined in table 4.4. The estimated total time is our own time measurement, using the computer clock, from the time of pressing accept bid on the buyer side to the time of the last contract being claimed.

Table 4.4: Summary of delays in MM:SS.

Exchange	Delay Seller	Delay Buyer	Estimated Total Time
ETH - ETC	00:33	04:47	05:44
ETH - ETC	00:27	02:04	04:35
ETH - ETC	01:17	05:42	06:34
ETC - ETH	00:22	02:11	04:10
ETC - ETH	00:46	03:37	04:37
ETC - ETH	05:00	08:20	09:40

For the BTC-BTC tests, the delay was less than 1 second due to the tests not going through the OrbitDB database and messaging. However, in order for the first claimer to redeem, one block was generated on the first chain, and in order for the second claimer to redeem, a second block was generated on the first chain. This delay is considered to be acceptable for the prototype since it is a proof-of-concept.

4.4.3 Test C : Cost

Table 4.5: Summary of costs for performing a trade rounded to three significant digits. Prices for gas [70, 71] and USD [72, 73] are set as of 2018-05-13.

Blockchain	Action	Gas used	Gas cost	Cost(Ether)	Cost(USD)
Ethereum	Submit	404715	3 GWei	0.00121	0.816
Ethereum	Claim	15976	3 GWei	0.0000479	0.0326
Ethereum Classic	Submit	404715	18.318 MWei	0.00000741	0.000135
Ethereum Classic	Claim	15976	18.318 MWei	0.000000293	0.00000534

The user that wants to exchange ETH to ETC would have to submit a contract on ETH and claim funds from ETC. This means that the total cost would be:

$$0.816 + 0.00000534 = 0.816 \text{ USD.} \tag{4.1}$$

The second user exchanging from ETC to ETH would instead have to pay a cost of:

$$0.000135 + 0.0326 = 0.0327 \text{ USD.} \tag{4.2}$$

This is a huge difference, with the first user paying 25.0 times more for the trade. This is due to that submitting a contract on Ethereum is significantly more expensive than claiming on Ethereum or doing any of the above on Ethereum Classic. As such, this is not an issue originating from the prototype.

For the Bitcoin to Bitcoin tests, several values were tried on the platform, with using different amounts of satoshi for the claiming party. The lowest amount achieved was 400 satoshi where the transactions continually went through. For the sender, the value was averaged over all sending transactions.

Table 4.6: Summary of costs for performing a trade on BTC-BTC. Price for sender and receiver in bitcon, and total price in USD 2018-05-13

Action	Cost(Bitcoin)	Cost(USD)
Send	0.0000363	0.310
Claim	0.00000400	0.0340

4.4.4 Test D : Trading pairs

The only interesting trading pair, as defined in section 4.2.1, that was tested was ETH-ETC. A BTC-BTC crosschain swap was also performed without the integration with the rest of the platform and the database. This swap was done in order to establish proof of the functionality, without having to integrate it fully. With the current implementation, all the cryptocurrencies in the prototype can be included in a trading pair, as the solutions for each cryptocurrency is isolated from each other. Any new blockchain has to include a library that fits the interface given in currency.js, to fit the function calls from the API. It would also require everyone trading with that cryptocurrency to have that type of functionality implemented as well.

Currently, there is functionality in the Ethereum library, that both Ethereum and Ethereum Classic uses, to add ERC20 tokens as available cryptocurrencies. The call to the library is similar as the call to ether on Ethereum, though it also needs an address to the contract of the token that is to be added. This would increase the amount of available trading pairs significantly as there are numerous ERC20 tokens on Ethereum [74].

5
Discussion

In this chapter, a qualitative discussion on to what extent the requirements were met as well as alternative paths that could have been taken will be discussed. Throughout the project, many decisions were made that altered the final product. Reflecting back on them, certain paths taken proved to be non-optimal. This shows that despite thorough investigating early on, non-optimal decisions can still be made. In this chapter, an attempt is done to reflect and evaluate over these and try to identify what could have been improved.

5.1 Discussion on the development process

The reasoning to split the development into the four categories database, communication, blockchain and, user interface was to make the development more efficient as one could specialize in one specific area. When the time came for the integration of the different parts, it took more time than expected due to the design and implementation being influenced by the person developing a specific part individually. A better approach could have been to spend more time on designing the architecture as a team which may have led to better integration.

To show that a decentralized cryptocurrency exchange is possible, a proof-of-concept approach was taken to develop a prototype. Since a full prototype was developed, the issues that come with developing a functioning product were also encountered in this prototype. One could ask if it would have been more productive to focus on simply developing the exchange method and trying to incorporate as many cryptocurrencies as possible instead of creating the surrounding platform.

Focus and time were largely put on developing the user interface and a model to make the exchange process automatic. This was to ensure that the requirements of message passing and the user interface were met. Instead, considering that the purpose was to achieve a secure exchange of cryptocurrencies, the focus could have been shifted towards incorporating more currencies rather than focusing on making the process automatic or the prototype more user friendly.

However, there is a difficult choice to be made here, whether one prioritizes a prototype that is easy to show what progress has been made or one that has made more progress with respect to incorporating more cryptocurrencies but is less user-friendly. The first path was clearly chosen for this prototype. The reason for this is that it lays a clearer path towards the incorporation of more currencies as well as being closer to a full specification.

It is now evident that this was not the preferred choice. In terms of quantitative results measured, the latter would have certainly yielded better and more significant results since the quality of the user experience was not measured outside the development team.

5.2 Discussion on testing and results

In this section, the results measured from the tests in Chapter 4 will be discussed. Although Chapter 4 provides its own evaluation, this section will delve more in-depth to try to understand the negative results in particular.

5.2.1 Method for measuring performance

Due to the high cost of trading on the main network of each blockchain, testing was done exclusively on test networks and private networks. The main network is considerably slower for completing transactions and therefore complete accurate data regarding performance of the prototype was not measured. The reason for not testing on the main network was due to time constraints and limited hardware. Testing on main networks require significant hardware space that was not attainable on the resources or budget present. Nevertheless, the procedure on the test networks are the same and the tests performed verify that the prototype is functional.

5.2.2 Testing on Ethereum

Throughout the testing process, several problems were encountered. The most substantial of these being problems relating to the test networks. Unreliable connections on both the Ethereum and Ethereum Classic public test networks have increased the difficulty of testing. On the private networks, the discovery protocol has also been unreliable, resulting in that two peers were sometimes unable to find each other.

During validation of the contracts, another issue was discovered; Ethereum networks previously thought of, and described as, having no differences did in fact have some non-apparent differences. For example, on the private networks, a certain function returned a non-case-sensitive string while the same function on public test networks returned a case-sensitive string.

These issues have limited our ability to continuously make tested increments to the prototype. This has in turn led to the continuous introduction of new bugs which considerably slowed down the development process. The differences between the different networks ought to have been noted earlier and the testing efforts should then have been focused on the test network that resembles the main network to the greatest extent.

5.2.3 Testing on Bitcoin

When executing the tests on the Bitcoin network, a few problems were encountered. The most troubling of these was the following error message:

```
  code: -26,
  message: '16: mandatory-script-verify-flag-failed
(Script evaluated without error but finished with a false/empty top
stack element)
```

Which was encountered when trying to claim from an HTLC address. This was the reason for the success rate being lower than 100%.

Extensive debugging was done to find out what caused this problem since it had been an issue in development of the Bitcoin solution from the start. However, since this appears to be generic error that can have a lot of causes, a solution to this issue is yet to be found and the problem does not seem to be directly related to the code. It does, however, seem to have something to do with the stack on the Bitcoin network that verifies transactions. Despite removing elements from the stack to try to pinpoint the problem, the error intermittently pops up no matter what elements are removed.

As for the delays on the Bitcoin network, the transactions were measured to be virtually instantaneous due to them not going through the database, and due to using the option to generate a block directly with Bitcoin Core's RPC method *generate*. This is, however, not a realistic scenario on the Bitcoin main network where new blocks are generated on average every 10 minutes. Having to generate two blocks, as was done in the tests, would result in it taking around 20 minutes for the full transaction to go through. Furthermore, with Bitcoin's convention to wait 6 blocks before a transaction is considered secure, the wait could be well over an hour.

When it comes to the costs, transactions that spend lower amounts of satoshi might take longer to be included in a block. Throughout the developing process, 400 satoshi were consistently shown to be enough to be accepted by the network, while lower amounts did not consistently get accepted. 3625 satoshi were spent on the submit transaction. Since the new blocks on the networks only contained the submit and claim transaction, the 400 satoshi for the claim transaction were enough to get the transactions accepted on the test network. However, 400 satoshi might not be enough on the main network. This might result in the timelock expiring before the claim transaction is added to a block, giving an opportunity for the other party to refund with a higher fee. For a more detailed explanation, see section 5.3.9. Since no tests were run on the main network, it is difficult to estimate the right fee but that is outside the scope of this thesis.

5.3 Discussion of the prototype

As the development process of the prototype is now finished, it can be concluded that it did not perform to all the expectations and there are several reason as to why that was the case. In this section, a qualitative discussion on each part of the prototype takes place in order to identify the issues. The prototype will be compared to the requirements specified in section 3.2.

5.3.1 API server

The server designed to manage communication between the user interface and the decentralized networks has a lot of issues regarding reliability and portability. These issues have appeared in both compilation and during runtime.

These issues have been pinpointed due to the development team lacking experience in Javascript. The libraries used for communication with the decentralized networks are also available in Python. This means that if the API server was written in Python, there could potentially have been less issues with these aspects as Python has a somewhat stronger type system and has a different compilation process than Javascript.

5.3.2 The bid database

The first two requirements of the database, allowing users to post and accept bids, were met and tested to work on both Windows and Linux. The last requirement, the database should be decentralized, was not fully met. OrbitDB markets itself as a decentralized database and it was believed to be the case. However, most of these technologies in use are mostly new and not fully developed and that proved to be the case with OrbitDB as well.

OrbitDB depends on the underlying infrastructure of IPFS, which is still in its early stages of development and some key functionalities, such as Circuit Relay, are not fully developed yet meaning that IPFS must rely on a TURN server to connect nodes to each other. This is a centralized point of failure which breaks the purpose of decentralization. This means that for the database to work, it needs to rely on a centralized point. It should be noted that a lot of work is in progress to fix this problem and that the team behind IPFS claims to be close to a solution [75].

Regarding the implementation of the database it has not been an optimal solution. Even though the integrity of the bids was ensured, the database holding the states of bids can still be manipulated to show faulty bids to users. Something that could be worked on in the future is ensuring that the users are only able to remove their own bids. This would eliminate the problem, and using several databases, as was done in the prototype, would not be needed. Users are also able to post bids they do not have the required funds for. The choice for not adding a local validation was decided on as malicious users would be able to simply remove it and add faulty bids. In a future implementation, a validation using the blockchain could be done to ensure that both parties have the required funds before executing a transaction.

The lack of complete support for Windows was unforeseen and resulted in a lot of overhead in finding a solution. Although the problem was ultimately solved, using another framework instead of OrbitDB may have been a wiser choice.

5.3.3 Exchange method

The first requirement for the exchange method was that it should be atomic. The chosen exchange method was HTLC, and it was specifically designed with the purpose of allowing for decentralized exchange of cryptocurrencies through atomic swaps [76]. As such, this requirement is fulfilled.

The second requirement was that it should be decentralized. Because HTLC is implemented to work directly in a blockchain, it is decentralized. The last requirement was that it should work for exchanges between several cryptocurrencies. This requirement has also been fulfilled since it was tested to work for ether on both Ethereum and Ethereum Classic, and bitcoin on the Bitcoin Network. In addition to that, HTLC is a proven technique that is supported by most cryptocurrencies.

5.3.4 Message passing

The first requirement was that the message passing should be automatic. To complete an exchange, a user should only have to either accept or push a bid as well as type in their password. This requirement has been fulfilled and was tested to work on both Ethereum and Ethereum Classic.

The second requirement was that message passing should not be dependent on any blockchain network, could not be fully tested. Although message passing did work on both Ethereum and Ethereum Classic, those platforms are too similar to conclude that the requirement has been met. The atomic swap that was performed on the Bitcoin network did not use the message passing. However, OrbitDB, which message passing uses, works independently of any blockchain platform and therefore the message passing system can be considered to also work that way.

The last requirement that the message passing should be decentralized, cannot, similarly to the bid database, be considered to be fully met. Since message passing is simply peer-to-peer communication, using OrbitDB was not necessary. The last requirement could have been met by using any peer-to-peer communication technology of which there are an abundance.

5.3.5 User interface

The first requirement, to use real world concepts in the prototype, is fulfilled through word choice and the approach used to display bids is easy to understand for any target group. The second requirement, to prevent users from performing common errors, is fulfilled by strong from validation including the use of drop-down lists to display the currencies that are available to the user.

However, the third requirement, to display errors in a clear manner and that the user should be able to recover from them easily, has not fully been met. Issues deemed to originate from the API server makes it impossible to predict all possible exceptions that may be thrown and write user-friendly error message for every specific exception. This results not only in a prototype that is not user-friendly but also a prototype that is difficult to debug. Therefore, the issues with the user interface has further far-reaching consequences than simply affecting what is seen despite the user interface being decoupled from the back end.

5.3.6 Extension for additional cryptocurrencies

Although only a simple prototype was developed, the aim was to leave the design open for extension for supporting other cryptocurrencies. Initially, the idea was to limit the prototype to the Ethereum platform but towards the end support for Bitcoin was added as well. Considering that the integration with Bitcoin was a rather arduous process that could not be tested in conjunction with the prototype, it could be concluded that the prototype did not fully achieve the goal of being easy to integrate with other cryptocurrencies. However, with the Bitcoin integration in place, the prototype would be easy to extend to support many of the altcoins that build on Bitcoin technology.

5.3.7 Reliability

Reliability of the prototype has consistently been an issue in the development process and continues to be. Working at the forefront of blockchain technology, most of the frameworks and platforms that are used in our prototype are in their early stages. Considering this, they also tend to be rather unreliable resulting in that our prototype is unreliable as well. The prototype has a plethora of dependencies which increases the difficulty of debugging, and causes reliability issues cross-platform.

As reliability has not been a focus in the prototype, it remains an issue that needs to be solved to create a full-fledged application. It was deemed that reliability was not necessary for a proof-of-concept prototype, but unfortunately, this did affect the development process.

5.3.8 Scalability issues

Issues pertaining to the scalability of our prototype is limited to the database since all exchanges will remain between two individuals peer-to-peer. OrbitDB is set-up so that all information is stored on all nodes. Due to the decision of using an append-only database to ensure the integrity of bids, it means that the database will keep growing. This results in unnecessary space used to store bids that are finished. It is therefore not a practical solution for the long term.

5.3.9 Security of the exchange method

Security is in this section defined as the possibility of a user losing funds due to an action by another malicious part. The exchange method has been tailored around the concept of atomic swaps and thereby the security of the prototype relies on how safe the protocol is. Therefore, no tests have been run to test the security. Below is an example where one user loses his funds:

1. Alice opens a payment channel to Bob
    (a) With a digest of a hashed password
    (b) With the specified amount Bob wants
    (c) With a reclaim time limit
2. Bob validates Alice's payment channel
3. Bob creates a payment channel to Alice
    (a) With the same digest
    (b) With the specified amount Alice wants
    (c) With a reclaim time limit
4. Alice secures Bob's payment channel by entering her password
5. Up to here everything is normal, for now. Bob now has access to Alice's password, and tries to secure her payment, but Alice's network is so clogged that the claim transaction cannot be processed before the contract time-lock opens.
6. Alice sees that the contract time-lock is up and Bob has not claimed the money, and so Alice send a refund transaction with a tip so that it is processed faster in the clogged network.
7. Alice get both hers and Bob's money.

As seen in the example, the exchange method is not completely foolproof. This is why it is important that the refund time and transaction cost is taken into careful consideration. In addition to this, the user loses funds when a contract is created. Malicious users accept posted bids with no intention of completing them. The counterpart then loses funds as the user creates and runs the contract. Doing this systematically can lead to great loss of cryptocurrencies.

5.4 Societal and ethical aspects

The prototype has the purpose of enabling cryptocurrency exchange. In this section however, the general implications of blockchain technology and the prototype, or the development of it, can have on the societal and ethical aspects will be discussed. Throughout the project, it has been known that there are both positive and negative societal and ethical aspects. These will be brought up in this section as well as the paths taken to minimize the negative impacts. The section concludes with a general discussion.

5.4.1 Societal advantages

Cryptocurrencies add several positive societal aspects ranging from more freedom to opening new technological possibilities. Cryptocurrencies particularly finds its use in countries with a restricted market. The issuing of new coins is predefined in the specification of cryptocurrencies so the same type of value manipulation of the currency as is seen with fiat money is not possible [77]. In countries with a volatile currency, such as Zimbabwe in 2008, cryptocurrencies could potentially have a positive impact simply by their potential ability to maintain a stable value [78]. However, cryptocurrencies today do not have a stable value.

Decentralized platforms avoids the necessity of trusting a central organization to handle your money. In the event of bankruptcy, a central organization would lose all your money, meanwhile a decentralized system is undisturbed as its risks are spread out on several nodes. It also frees up the market by enabling unrestricted trading without needing approval of a centralized third party and it also has the potential of reducing transaction costs due to it not being arbitrarily decided anymore.

Another example of what this technology could be used for has been shown by the Swedish government agency *Lantmäteriet*. The agency uses blockchain technology to verify real estate transactions and to avoid fraud. In their report they describe that blockchains can be used as a digital representation of signatures on paper and thereby reduce paper consumption [79].

5.4.2 Ethical issues and societal disadvantages

Despite the overall positive aspects of blockchain technology, the fact that the transactions are mostly anonymous leads to some negative consequences. Cryptocurrencies have been used to buy illegal goods such as narcotics and unlicensed weapons [80]. They have also been forbidden in certain countries and in others there are no regulations at all [81].

This project is limited to the development of a prototype. In a full-fledged decentralized platform, there is no effective way to prevent its usage in specific countries, such as countries where cryptocurrencies are forbidden. However, the platform is only used to facilitate decentralized exchange of cryptocurrencies, and a user first needs to own cryptocurrency to use the platform. A user in a country where these cryptocurrencies are forbidden must therefore already have committed a crime before using the platform. Having said that, trading of these currencies is regulated and taxable in several countries, such as in Sweden [82]. The prototype intends to allow for anonymous trading of these currencies, unlike centralized exchanges, which may result in that it is used to avoid tax payments.

Another unavoidable issue with the platform is that it allows for anonymous money laundering. When a transaction with the platform occurs, the ability to track the cryptocurrencies involved in the transaction are lost.

5.4.3 General discussion on Social and Ethical aspects

The implementation of the project has a minimal impact on the environment and society. That is however not true for using the prototype. It is estimated that the transaction of one bitcoin corresponds to the energy consumption for one house in one month [83]. Our prototype involves two transfers for every agreed upon transaction, and this energy expenditure is therefore doubled. New solutions, such as the Raiden Network, have been developed with the intention of reducing this environmental impact [84]. During the development, the aim is to reduce the total impact by limiting the testing to test networks. The energy expenditure is considerably less on test networks because the difficulty to break new blocks is much easier due to lower security demands.

What can be said is that cryptocurrencies and blockchains are both new technologies and it is too early to say what the long term consequences will be. However, it is clear that change is on the way. In addition to the countries that have forbidden cryptocurrencies, several other countries have introduced new laws and restrictions on it [85]. If cryptocurrencies continue to gain in popularity at the same rate [86] the probability that states introduce new regulations also increases. Blockchain technology is in its development phase, and new applications of it will likely arise such as the aforementioned example with Lantmäteriet. Therefore it is hard to predict what consequences the technology will have and what role our prototype will play.

6
Conclusion

The core purpose of this thesis, which was to research how to perform atomic swaps and develop a prototype for doing so, has been accomplished with a system that can exchange Ethereum and Ethereum Classic. The target was not to build a perfect system but instead to show that this is possible by utilizing a proof-of-concept approach. The results allow for the possibility to further investigate the area and reach the goal of developing a fully functional system that can be used in production. Possible future improvements could be to troubleshoot the code and remove bugs, enhance user interface portability and to improve handling of the decentralized database. Working with experimental technologies is exciting but they tend to have their own unsolved issues. However, the groundwork built in this thesis opens up opportunities for others to extend and improve.

Bibliography

[1] CoinMarketCap. *Historical Snapshot - Janury 03, 2016.* URL: https://coinmarketcap.com/historical/20160103/ (visited on 2018-03-26).

[2] CoinMarketCap. *Historical Snapshot - March 25, 2018.* URL: https://coinmarketcap.com/historical/20180325/ (visited on 2018-03-26).

[3] CoinMarketCap. *Ethereum Charts.* URL: https://coinmarketcap.com/currencies/ethereum/#charts (visited on 2018-05-10).

[4] Encyclopædia Britannica. *Fiat Money.* URL: https://www.britannica.com/topic/fiat-money (visited on 2018-02-20).

[5] Encyclopædia Britannica. *Crown.* URL: https://www.britannica.com/topic/crown-monetary-unit (visited on 2018-04-25).

[6] Encyclopædia Britannica. *Federal Reserve System.* URL: https://www.britannica.com/topic/Federal-Reserve-System (visited on 2018-04-25).

[7] M. Rees. *You Say Bitcoin Has No Intrinsic Value? Twenty-two Reasons to Think Again.* URL: https://bitcoinmagazine.com/articles/you-say-bitcoin-has-no-intrinsic-value-twenty-two-reasons-to-think-again-1399454061/ (visited on 2018-04-27).

[8] Bitcoin.org. *FAQ, Who created Bitcoin?* URL: https://bitcoin.org/en/faq#who-created-bitcoin (visited on 2018-01-26).

[9] Investopedia. *Currency Exchange.* URL: https://www.investopedia.com/terms/c/currency-exchange.asp (visited on 2018-03-29).

[10] Cryptoincome. *The Ultimate Guide to Centralized Cryptocurrency Exchanges.* URL: http://cryptoincome.io/ultimate-guide-centralized-cryptocurrency-exchanges/ (visited on 2018-03-05).

[11] S. Khatwani. *Decentralized Crypto Exchanges vs Centralized Exchanges Like Binance, Bittrex.* URL: https://coinsutra.com/decentralized-vs-centralized-crypto-exchange/ (visited on 2018-03-05).

[12] Reuters. *Cryptocurrency Exchanges Are Increasingly Roiled by Hackings and Chaos.* URL: http://fortune.com/2017/09/29/cryptocurrency-exchanges-hackings-chaos/ (visited on 2018-03-05).

[13] F. Chaparro. *Some of the biggest crypto exchanges are shutting out new users because they can't keep up with demand.* URL: http://nordic.businessinsider.com/crypto-exchanges-are-shutting-out-new-users-because-they-cant-keep-up-with-demand-2017-12?r=US&IR=T (visited on 2018-03-05).

[14] CryptoCompare. *What Are Atomic Swaps?* URL: https://www.cryptocompare.com/coins/guides/what-are-atomic-swaps/ (visited on 2018-02-02).

[15] B. Cohen. *The BitTorrent Protocol Specification.* URL: http://www.bittorrent.org/beps/bep_0003.html (visited on 2018-04-25).

[16] Omsar. *Centralized databases*. URL: http://www.omsar.gov.lb/ICTSG/104DB/6.1_Centralized_Databases.htm (visited on 2018-02-09).

[17] Investopedia. *Definition of blockchain*. URL: https://www.investopedia.com/terms/b/blockchain.asp (visited on 2018-05-07).

[18] Bitcoin.org. *FAQ, How does bitcoin work?* URL: https://bitcoin.org/en/faq#how-does-bitcoin-work (visited on 2018-02-02).

[19] OddsShark. *Bitcoin trading fees explained*. URL: http://www.oddsshark.com/bitcoin/transaction-costs (visited on 2018-02-02).

[20] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: https://bitcoin.org/bitcoin.pdf (visited on 2018-04-27).

[21] CoinMarketCap. *Cryptocurrency Market Capitalizations*. URL: https://coinmarketcap.com/ (visited on 2018-05-10).

[22] Investopedia. *Bitcoin definition*. URL: https://www.investopedia.com/terms/b/bitcoin.asp (visited on 2018-05-01).

[23] Investopedia. *Altcoin definition*. URL: https://www.investopedia.com/terms/a/altcoin.asp (visited on 2018-05-01).

[24] Bitcoin wiki. *Bitcoin Script*. URL: https://en.bitcoin.it/wiki/Script (visited on 2018-05-01).

[25] Computer Hope. *Turing completeness*. URL: https://www.computerhope.com/jargon/t/turing-completeness.htm (visited on 2018-03-08).

[26] Bitcoin Project. *Bitcoin developer guide*. URL: https://bitcoin.org/en/developer-guide#block-chain (visited on 2018-05-01).

[27] R. Aitken. *Digital Gold 'Done Right' With DigixDAO Crypto-Trading On OpenLedger*. 2016-04. URL: https://www.forbes.com/sites/rogeraitken/2016/04/23/digital-gold-done-right-with-digixdao-crypto-trading-on-openledger/ (visited on 2018-01-26).

[28] Ethereum foundation. *Create your own crypto-currency with Ethereum*. URL: https://www.ethereum.org/token (visited on 2018-05-28).

[29] Ethereum foundation. *computation and turing completeness*. URL: https://github.com/ethereum/wiki/wiki/White-Paper#computation-and-turing-completeness (visited on 2018-04-27).

[30] P. Bajpai. *Bitcoin Vs Ethereum: Driven by Different Purposes*. URL: https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp (visited on 2018-03-08).

[31] Ethereum community. *What is gas?* URL: http://www.ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html#what-is-gas (visited on 2018-04-24).

[32] Investopedia. *Hard Fork Definition*. URL: https://www.investopedia.com/terms/h/hard-fork.asp (visited on 2018-04-27).

[33] D. Siegel. *Understanding The DAO Attack*. URL: https://www.coindesk.com/understanding-dao-hack-journalists/ (visited on 2018-05-10).

[34] V. Buterin. *Hard Fork Completed*. URL: https://blog.ethereum.org/2016/07/20/hard-fork-completed/ (visited on 2018-05-10).

[35] M. Rouse. *nonce (number used once or number once)*. URL: https://searchsecurity.techtarget.com/definition/nonce (visited on 2018-05-12).

[36] Investopedia. *Proof-of-work definition.* URL: https://www.investopedia.com/terms/p/proof-work.asp (visited on 2018-04-30).

[37] Investopedia. *51% attack definition.* URL: https://www.investopedia.com/terms/1/51-attack.asp (visited on 2018-04-30).

[38] Bitcoin wiki. *Confirmation on the Bitcoin network.* URL: https://en.bitcoin.it/wiki/Confirmation (visited on 2018-05-28).

[39] N. Szabo. *Smart Contracts: Building Blocks for Digital Markets.* URL: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html (visited on 2018-04-30).

[40] BTCX. *btcx.* URL: https://bt.cx (visited on 2018-01-26).

[41] ShapeShift. *ShapeShift.* URL: https://shapeshift.io/#/coins (visited on 2018-01-26).

[42] C. S. Carr. *Arpanet protocol.* URL: https://tools.ietf.org/pdf/rfc33.pdf (visited on 2018-05-07).

[43] R. Fielding et al. *HTTP 1.1.* URL: https://tools.ietf.org/html/rfc2068 (visited on 2018-05-07).

[44] I. Fette and A. Melnikov. *The WebSocket Protocol.* URL: https://tools.ietf.org/html/rfc6455 (visited on 2018-04-27).

[45] P. Srisuresh and K. Egevang. *Traditional IP Network Address Translator (Traditional NAT).* URL: https://tools.ietf.org/html/rfc3022 (visited on 2018-05-08).

[46] P. Matthews R. Mahy and J. Rosenberg. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).* URL: https://tools.ietf.org/html/rfc5766 (visited on 2018-05-09).

[47] J. Benet. *IPFS - Content Addressed, Versioned, P2P File System.* URL: https://ipfs.io/ (visited on 2018-04-27).

[48] A. Béraud. *What are Distributed Hash Tables ?* URL: https://github.com/savoirfairelinux/opendht/wiki/What-are-Distributed-Hash-Tables-%3F (visited on 2018-05-12).

[49] IPFS. *Tutorial - Understanding Circuit Relay.* URL: https://github.com/ipfs/js-ipfs/tree/master/examples/circuit-relaying (visited on 2018-05-14).

[50] P. Lake and P. Crowther. *Concise Guide to Databases.* URL: https://www.springer.com/gp/book/9781447156000#aboutAuthors (visited on 2018-04-27).

[51] ITS. *Distributed Database.* URL: https://www.its.bldrdoc.gov/fs-1037/dir-012/_1750.htm (visited on 2018-04-27).

[52] E. Wong. *User Interface Design Guidelines: 10 Rules of Thumb.* URL: https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb (visited on 2018-05-10).

[53] M. Herlihy. "Atomic Cross-Chain Swaps". In: *ArXiv e-prints* (2018-01). arXiv: 1801.09515 [cs.DC].

[54] Bitcoin Wiki. *Hashed Timelock Contracts.* 2017-09. URL: https://en.bitcoin.it/wiki/Hashed_Timelock_Contracts (visited on 2018-05-13).

[55] C. Hatch. *An implementation of a Hashed Timelock Contract on Ethereum.* 2017-01. URL: https://github.com/chatch/hashed-timelock-contract-ethereum (visited on 2018-05-13).

[56] Bitcoin wiki. *Bitcoin opcodes.* URL: https://en.bitcoin.it/wiki/Script#Opcodes (visited on 2018-05-13).

[57] B. Clauss. *Comparing hashing algorithms.* URL: https://blog.novatec-gmbh.de/choosing-right-hashing-algorithm-slowness/ (visited on 2018-05-13).

[58] A. Castor. *One of Ethereum's Earliest Smart Contract Languages Is Headed for Retirement.* URL: https://www.coindesk.com/one-of-ethereums-earliest-smart-contract-languages-is-headed-for-retirement/ (visited on 2018-05-13).

[59] Ethereum Foundation. *Building a smart contract using the command line.* URL: https://www.ethereum.org/greeter (visited on 2018-05-13).

[60] OrbitDB. *OrbitDB.* URL: https://github.com/orbitdb/orbit-db/ (visited on 2018-05-12).

[61] J. Wilcke V. Trón and F. Lange. *Geth.* URL: https://github.com/ethereum/go-ethereum/wiki/geth (visited on 2018-03-28).

[62] Bitcoin Core. *About Bitcoin Core.* URL: https://bitcoincore.org/en/about/ (visited on 2018-05-10).

[63] E. Czaplicki. *Error Handling and Tasks.* URL: https://guide.elm-lang.org/error_handling/ (visited on 2018-05-13).

[64] M. Kotewicz and F. Vogelsteller. *Ethereum JavaScript API.* URL: https://github.com/ethereum/web3.js (visited on 2018-05-13).

[65] O. Hanhinen. *Elm in the real world.* URL: https://futurice.com/blog/elm-in-the-real-world (visited on 2018-05-13).

[66] Komodo Platform. *Komodo. An Advanced Blockchain Technology.* URL: https://www.komodoplatform.com/en/technology/whitepaper/2018-02-03-Komodo-White-Paper-Full.pdf (visited on 2018-02-09).

[67] CryptoCoinNews. *What Bitcoin Exchanges Won't Tell You About Fees.* 2015-07. URL: https://www.ccn.com/what-bitcoin-exchanges-wont-tell-you-about-fees/ (visited on 2018-02-09).

[68] Gas tracker. *Ethereum Classic Block Explorer.* URL: https://gastracker.io/ (visited on 2018-05-13).

[69] ETH Gas Station. *Token Tracker.* URL: https://ethgasstation.info/ (visited on 2018-05-13).

[70] ETH Gas Station. *Estimates over last 1,500 blocks.* URL: https://ethgasstation.info/index.php (visited on 2018-05-13).

[71] Gastracker.io. *Ethereum Classic Block Explorer.* URL: https://gastracker.io/ (visited on 2018-05-13).

[72] CoinGecko. *Ethereum Price Chart US Dollar (ETH/USD).* URL: https://www.coingecko.com/en/price_charts/ethereum/usd (visited on 2018-05-13).

[73] CoinGecko. *Ethereum Classic Price Chart US Dollar (ETC/USD).* URL: https://www.coingecko.com/en/price_charts/ethereum-classic/usd (visited on 2018-05-13).

[74] Etherscan. *Token Tracker.* URL: https://etherscan.io/tokens (visited on 2018-05-13).

[75] D. Dias. *Removing the Central Rendezvous Points for the *-star transports.* URL: https://github.com/libp2p/js-libp2p/issues/134 (visited on 2018-05-13).

[76] P. Traugott. *What is a Hashed Timelock Contract (HTLC)? – A Beginner's Guide.* URL: https://captainaltcoin.com/hashed-timelock-contract-htlc/ (visited on 2018-05-11).

[77] Bitcoin.org. *FAQ, How are bitcoins created?* URL: https://bitcoin.org/en/faq#how-are-bitcoins-created (visited on 2018-02-02).

[78] S. H. Hanke and A. K. F. Kwok. "On the Measurement of Zimbabwe's Hyperinflation". In: *Cato Journal* 29.2 (2009). URL: https://object.cato.org/sites/cato.org/files/serials/files/cato-journal/2009/5/cj29n2-8.pdf (visited on 2018-02-09).

[79] Kairos Future. *The Land Registry in the blockchain - testbed.* Tech. rep. The Swedish Mapping, Cadastre and Land Registration Authority, 2017-03. URL: https://chromaway.com/papers/Blockchain_Landregistry_Report_2017.pdf (visited on 2018-02-02).

[80] C. Albanesius. *What Was Silk Road and How Did It Work?* 2013-10. URL: http://uk.pcmag.com/internet-products/12660/news/what-was-silk-road-and-how-did-it-work (visited on 2018-02-02).

[81] U. Chohan. "Accessing the Differences in Bitcoin & Other Cryptocurrency Legality Across National Jurisdictions". 2017-09. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3042248 (visited on 2018-02-09).

[82] Skatteverket. *Virtuella valutor.* 2017-10. URL: https://www.skatteverket.se/privat/skatter/vardepapper/andratillgangar/virtuellavalutor.4.15532c7b1442f256bae11b60.html (visited on 2018-02-02).

[83] T. Brosens. *Why Bitcoin transactions are more expensive than you think.* URL: https://think.ing.com/opinions/why-bitcoin-transactions-are-more-expensive-than-you-think/ (visited on 2018-02-02).

[84] mooncryption. *Scaling Cryptos: Bitcoin Lightning Network vs Ethereum Raiden Network.* 2017-09. URL: https://steemit.com/bitcoin/@mooncryption/scaling-cryptos-bitcoin-lightning-network-vs-ethereum-raiden-network (visited on 2018-02-02).

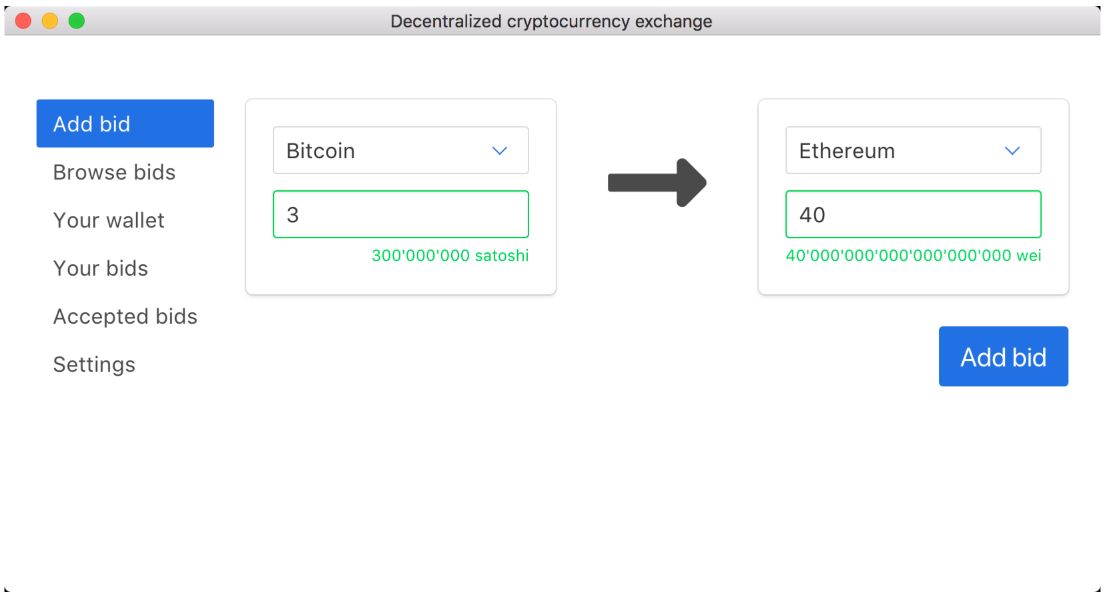
[85] F. McKenna. *Here's how the U.S. and the world regulate bitcoin and other cryptocurrencies.* 2017-12. URL: https://www.marketwatch.com/story/heres-how-the-us-and-the-world-are-regulating-bitcoin-and-cryptocurrency-2017-12-18 (visited on 2018-02-02).

[86] CoinMarketCap. *Total Market Capitalization.* URL: https://coinmarketcap.com/charts (visited on 2018-02-02).

A
Source code

The prototype is licensed under the MIT license and its source code is available at https://github.com/sutbult/DATX02-18-07.

B
User interface



Bitcoin

3

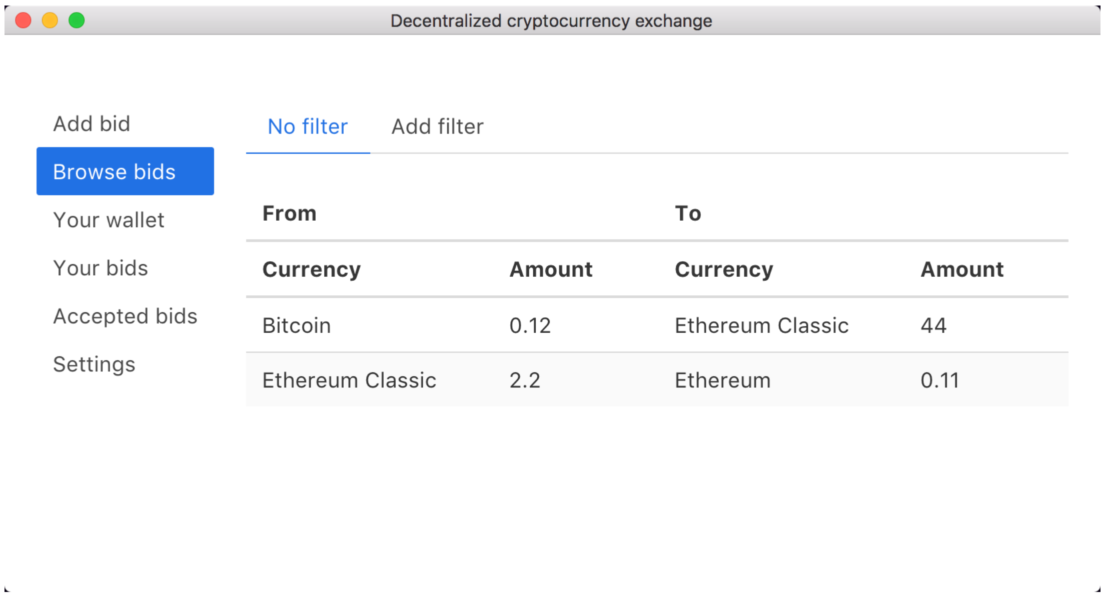
300'000'000 satoshi

Ethereum

40

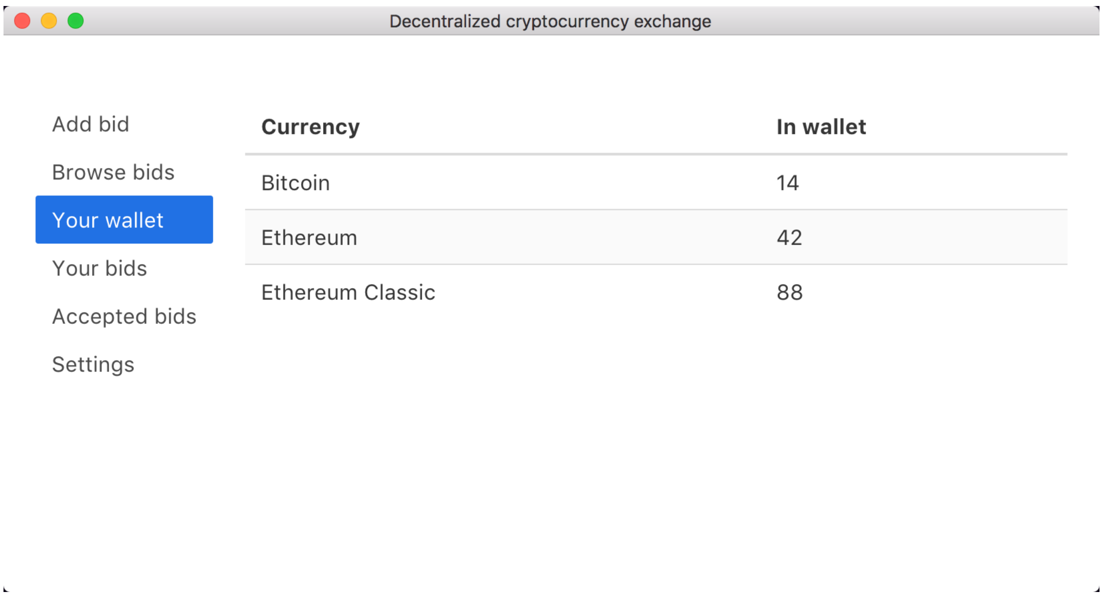
40'000'000'000'000'000'000 wei

Figure B.1: Add bid.



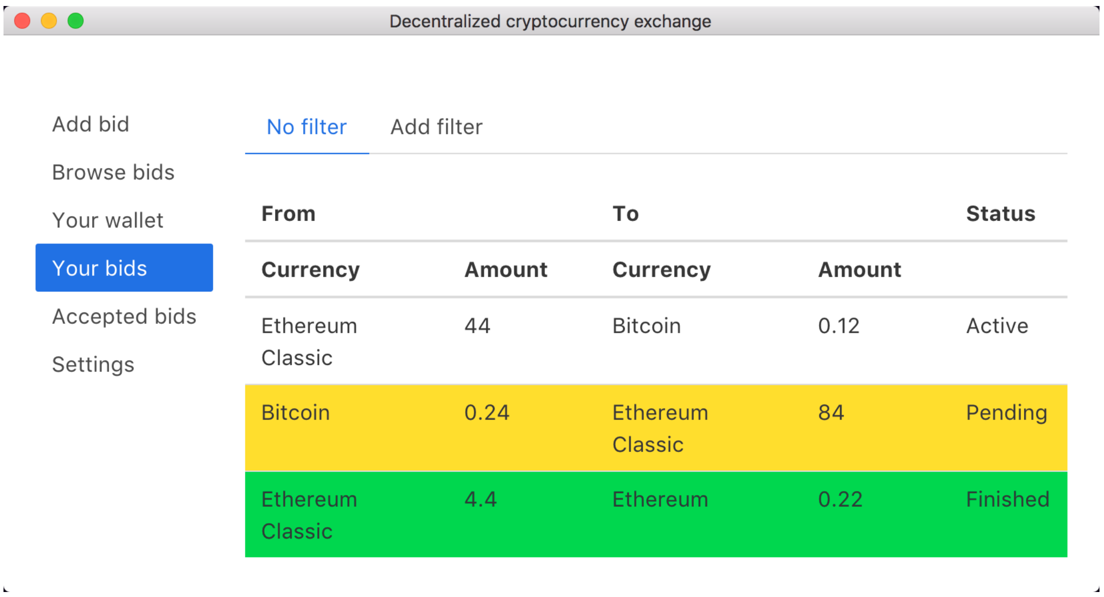
From		To	
Currency	Amount	Currency	Amount
Bitcoin	0.12	Ethereum Classic	44
Ethereum Classic	2.2	Ethereum	0.11

Figure B.2: Browse bids.



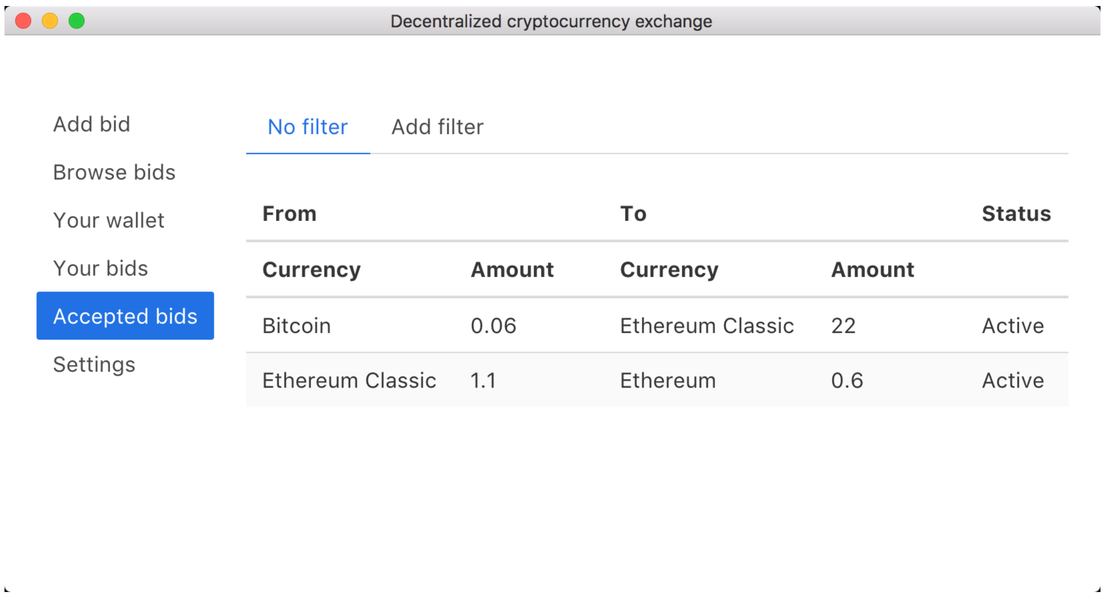
Currency	In wallet
Bitcoin	14
Ethereum	42
Ethereum Classic	88

Figure B.3: Your wallet.



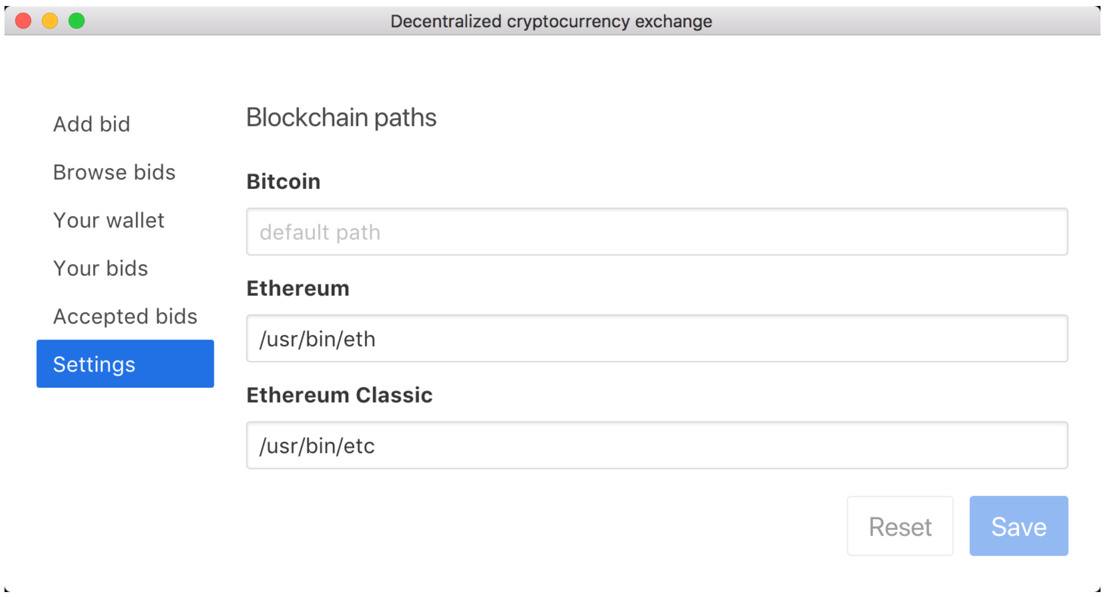
From		To		Status
Currency	Amount	Currency	Amount	
Ethereum Classic	44	Bitcoin	0.12	Active
Bitcoin	0.24	Ethereum Classic	84	Pending
Ethereum Classic	4.4	Ethereum	0.22	Finished

Figure B.4: Your bids.



From		To		Status
Currency	Amount	Currency	Amount	
Bitcoin	0.06	Ethereum Classic	22	Active
Ethereum Classic	1.1	Ethereum	0.6	Active

Figure B.5: Accepted bids.



Blockchain paths

Bitcoin: default path

Ethereum: /usr/bin/eth

Ethereum Classic: /usr/bin/etc

Figure B.6: Settings.