The BarterDEX

Whitepaper - Draft v0.4

A Decentralized, Open-Source Cryptocurrency Exchange,
Powered by Atomic-Swap Technology

Created by The Komodo Organization

Introductory Note from Komodo

This whitepaper is a detailed explanation of the BarterDEX protocol—a decentralized method of exchanging cryptocurrencies via atomic-swap technology, without counterparty risk. The creation of this protocol is part of a larger open-source project known as Komodo.

Many readers of this whitepaper may be interested to understand how the BarterDEX protocol fits into the overall vision of the Komodo project. Such readers are currently finding it difficult to locate relevant and simplified information regarding Komodo, due to the project's technical complexity and comparatively young age. As a team, we are working diligently to provide simplified explanations of Komodo's open-source technology, and the BarterDEX whitepaper is but one step forward.

To summarize the Komodo project in one sentence:

Komodo is to blockchain technology what Linux is to operating systems.

The Komodo project is an advancement in the causes of the decentralization and open-source movements; we focus on blockchain technology.

Therefore, while The BarterDEX itself is necessary as we work to further the Komodo endeavor, it is also ultimately separate from Komodo. Were we to design The BarterDEX to be permanently locked to Komodo, it would make Komodo a centralized point of control. This would be contrary to the principles of decentralization and open-source development.

We are actively developing various other technologies that will interact with The BarterDEX. Watch for more information about these associated projects on our website: komodoplatform.com

- The Komodo Team, November 21, 2017

Abstract

The BarterDEX allows people to trade cryptocurrency coins without a counterparty risk. The protocol is open-source and trading is available for any coin that any developers choose to connect to The BarterDEX. The parent project, Komodo, freely provides BarterDEX technology through open-source philosophy. Our service fully realizes decentralized order matching, trade clearing, and settlement. The order-matching aspect uses a low-level pubkey-to-pubkey messaging protocol, and the final settlement is executed through an atomic cross-chain protocol. Like any exchange, our decentralized alternative requires liquidity, and we provide methods and incentives therein.

Introduction

The current, most practical method for cryptocurrency exchange requires the use of centralized exchange services. Such centralized solutions require vouchers to perform the exchange. Among many dangers present in this system, end-users are under the constant risk of their assets being stolen either by an inside theft or by an outside hack. Furthermore, the operators of centralized exchanges can exhibit bias in how they facilitate trading among their users. To eliminate such dangers and limitations requires the creation of a decentralized-exchange alternative.

Among all the centralized exchanges, trading tends to coalesce around a few of the most popular. There is a reason for this behavior. Trading via vouchers is fast; a central exchange can swap internal vouchers instantaneously, whereas trading actual cryptocurrencies through human-to-human coordination requires communication from both parties. It requires waiting for blockchain miners to calculate transaction confirmations. The speed advantage of a centralized exchange, therefore, creates a compounding effect on the centralization of traders. The faster processing time of vouchers attracts more people: the increased presence of traders creates higher liquidity: with more liquidity, the exchange can feature better prices: the higher quality of prices in turn attracts a larger community, and the cycle begins again. This effect is the reason that a few centralized exchanges dominate with high-volume trading, while smaller exchanges—both centralized and decentralized—suffer from a lack of liquidity.

The Beginnings and Travails of Decentralized Exchanges

In 2014 a project called The MultiGateway created one of the first decentralized resources for trading cryptocurrencies. The MultiGateway relied on a separate, though related, blockchain project called The Nxt Asset Exchange. The latter facilitated the decentralized exchange of blockchain coins by using proxy tokens (as opposed to vouchers), and these proxy tokens represented external cryptocurrencies (such as Bitcoin).

The underlying technology of this solution is still in use by many blockchain platforms, but the proxy-token protocol is too limited to compete with centralized exchanges. Because trading by the means of proxy tokens requires trading on an actual blockchain, the trading process loses the speed of a centralized exchange. Also, a proxy-token decentralized exchange must still have a storage center to hold the external cryptocurrencies represented by the proxy tokens. At best, this storage center is only distributed, and therefore end-users are under the same counterparty risk that exists in centralized exchanges. Furthermore, the process of trading on proxy-token platforms requires using a set of gateways (i.e. “The MultiGateway”) to convert external native coins (such as Bitcoin) to and from the affiliated proxy tokens. Together, these many problems make the proxy-token method of decentralized trading an impractical solution.

Therefore, a decentralized exchange alternative that seeks to successfully remove the threats and limitations of centralized-exchange services must feature the same speed, liquidity, and convenience of a centralized exchange. As of today, no decentralized exchange has successfully replaced any of their centralized counterparts.

The BarterDEX: A Complete Solution

We now present a new decentralized technology that we believe makes a competitive decentralized exchange possible. We call our technology The BarterDEX, and it allows people to freely and safely exchange cryptocurrency coins from person to person.

The BarterDEX decentralized exchange creates a competitive method for bartering cryptocurrencies, combining three key components: order matching, trade clearing, and liquidity provision. These components are combined into a single integrated system that allows users to make a request to trade their coins, find a suitable trading partner, and complete the trade using an atomic cross-chain protocol. Additionally, The BarterDEX provides a layer of privacy during the order-matching process, enabling two nodes to perform a peer-to-peer atomic swap without any direct IP contact.

The “order matching” component is the process of pairing an end-user’s offer to buy with another end-user’s offer to sell. This component is not the actual trade itself, but is only a digitally created promise between end-users stating that they will perform their parts of the trade.

The order-matching process is achieved by algorithms that define how the orders are paired, and in which order they are fulfilled.

After a successful order-matching execution, the next component is the “clearing” aspect of the trade, wherein end-users must fulfill their promises. This is the process wherein the assets are swapped between the trading parties. The BarterDEX facilitates this process and assures the safety of the users therein.

Recall that in previous decentralized exchanges there lies a problem when an exchange has low liquidity. The BarterDEX solves this problem by creating Liquidity Provider Nodes (LP nodes). LP’s are trading parties that act as market-makers, buying and selling assets. They provide liquidity to the exchange, and make their profit from the spread between bid and ask orders. LP's bring price stability to the market, and facilitate end-users in making fast and efficient trades.

Improvements in Our Current BarterDEX Iteration

The BarterDEX is the result of years of development and iterated versions, with each iteration adding the next layer of required functionality to achieve our eventual goal of large-scale adoption.

With this incarnation, The BarterDEX adds support for SPV Electrum-based coins, as well as dozens of normal bitcoin-protocol coins running native-coin daemons. Internally, the "SPV" aspect of a coin is abstracted so that most of the API calls work transparently for these SPV-mode coins and native-coin daemons.

The BarterDEX also enables a feature known as Liquidity Multiplication, a protocol that allows the same funds to be used in multiple requests on BarterDEX "orderbooks." The first request to fill completes the trade, and all the other outstanding requests are immediately cancelled.

Liquidity Multiplication therefore allows an initial amount of funding to create an exponentially higher amount of liquidity on the exchange. This also provides a special advantage for traders that like to wait for below-market dumps. While this feature is something that any other exchange could implement, very few do; on the BarterDEX, all orderbook entries are 100% backed by real funds, as opposed to a centralized exchange's vouchers, which are not as reliable and therefore would present yet another danger for their end-users.

The BarterDEX API

It is possible to create an API model that is the same for all coins—with the obvious exceptions of the electrum-API call itself, and within some of the returned JSON files that have different calls, such as "listunspent."

Furthermore, the underlying technology of the BarterDEX enables the API to treat all bitcoin-protocol compatible coins with a universal-coin model. Therefore, when working with the BarterDEX API, an independent developer working to feature their coin on The BarterDEX need only use the API "coin" symbol to receive the full set of BarterDEX features. A coin that is not built on the bitcoin-protocol CheckLockTimeVerify (CLTV) feature can still take advantage of the liquidity-taker side of the BarterDEX API.

For a coin to work in native mode, it must have a <gettxout> RPC call; if the coin has the CLTV opcode, it can be both the liquidity provider and the liquidity taker. For coins using SPV, The BarterDEX only supports the liquidity-taking side (for overall network-performance reasons). Also, we assume that any trader with ambitions of being a serious liquidity provider should also be serious enough to install the coin daemon for the coins they are trading, as this will increase their speed of processing.

BarterDEX Technology

Before we get into details regarding the nature of atomic swaps, there are several aspects of The BarterDEX that are critical to understand.

Order Matching

The first is the decentralized orderbook (which we briefly encountered earlier when speaking about Liquidity Multiplication). The orderbook is the collection of bids and offers that end-users place on the network. To create our orderbook, The BarterDEX creates a custom peer-to-peer network that employs two separate types of nodes: a full-relay node and a non-relay node.

Order Matching with Full-Relay and Non-Relay Nodes

The difference between a full-relay node and a non-relay node is that the former is typically a high-volume trader who provides liquidity to the network in exchange for being a trading hub on the network. This puts him in the position of being able to complete trades more quickly than his trading competitors. The latter type of node (non-relay) is the more common user, who engages with The BarterDEX exchange when trading one cryptocurrency for another, given the user's daily motivations.

There are no requirements or payments necessary to become either type of node, and so anyone desiring to become a high-volume full-relay node will find no restrictions. To be successful as a full-relay node, however, one must be able to carry out transactions on the network with a competitive Internet connection and high-capacity bandwidth.

There are several incentives encouraging users to become full-relay nodes, as these types of nodes are necessary to build the backbone of The BarterDEX network. One incentive to run a full-relay node is that by being at the center of a wide network of non-relay nodes, the full-relay node has better connectivity and thus a higher chance of being the first to complete a trade. Furthermore, those who have a significant share in the Komodo asset chain, DEX (which pays dividends based on The BarterDEX network performance), will likely desire to be a full-relay node to encourage high rates of return. Incentives such as these should ensure a sufficiency of nodes.

A non-relay node has all the same available trading options. Non-relay nodes are only limited, naturally, in terms of the total number of connections they maintain to other users. We expect that the vast majority of nodes joining the network will be non-relay nodes.

In theory, roughly 100 full-relay nodes should be able to support thousands (if not tens of thousands) of non-relay nodes, thus providing a large and high-volume network. We have performed all the necessary internal stress tests necessary for predictions of our limitations, and we are now in the process of achieving real-world implementation, where we expect to discover our actual limitations.

When these limitations do arise, we do have various contingencies in place, one of which is the creation of clusters. It is possible to create clusters of BarterDEX nodes that are separate from other clusters on the network. To achieve this, when one cluster approaches a level of user load that is overcapacity, users can opt to seed a new cluster by creating an independent set of seed nodes. This feature amplifies the scalability of The BarterDEX network, as it allows clusters of users to form in accordance with user desires. We assume that at large scales there will be sufficient inventory in the orderbooks for clusters to provide ample asset liquidity, especially if the act of partitioning into a new cluster is based on trading a coin that is overcrowded.

Furthermore, as we continue to develop this new technology, we may also create a protocol that will allow these separate clusters to share their orderboards via bridge nodes, which in theory can act to cross-pollinate desired orders from one cluster to another.

To optimize the network load, we minimize the hierarchical transmission of the orderbooks and the fetching of data. There are also several different methods of obtaining data by which we can maximize the number of nodes that can fully connect to the BarterDEX network.

Jumblr Technology Adds Privacy

While The BarterDEX does not require non-relaying nodes to publicly share their IP addresses, it is important to note that The BarterDEX itself is not private. Instead, we use Jumblr, an accompanying Komodo technology, to provide privacy options.

On the surface, non-relaying nodes perform addressing via a <curve25519> pubkey, and the IP address of one non-relaying node is normally not directly shared with their accompanying non-relaying trading partner. However, full-relay nodes are capable of monitoring IP addresses at the lower levels of the network, and therefore a malicious actor would be able to link IP addresses of non-relay nodes to pubkeys, thus uncovering the most crucial aspects of their privacy. Therefore, users should assume that if privacy is important for their given trading activity, they need to employ Komodo’s additional privacy technology, Jumblr. (To learn more about Jumblr, please visit: https://github.com/jl777/komodo)

The Roots in Iguana Core

The BarterDEX itself is a fork of one our earliest codebase experiments, Iguana Core. The technology underlying Iguana Core is complex and to fully explain would require a separate whitepaper. To briefly summarize, the Iguana codebase is a collection of customized technologies that enable many of the features that The BarterDEX and other Komodo technologies now provide. Iguana is primarily coded in the C programming language—the language of choice of our lead developer and Komodo founder, JL777.

Note to Users: Some of the features that Iguana Core enables are highly advanced, and therefore users interacting with The BarterDEX and other Iguana-compatible GUI software applications should always perform proper research and exercise caution.

All BarterDEX transactions that use the atomic-swap protocol are created and signed in a format that is managed by the Iguana Core codebase. This enables a powerful combination of features.

One specific feature is a specialty wallet that can manage and trade among a multiplicity of different blockchain coins. To explain the significance of this multi-coin wallet feature, let us observe how a standalone GUI app formerly interacted with cryptocurrencies.

Previously, for a GUI software application to manage cryptocurrencies, the software application usually required the creation of a wallet.dat file, which is locally stored on the user’s computer. This wallet.dat file held the privkeys (passwords that unlock funds on a blockchain) and other encryption-enabled protocols necessary for the user to manage funds. There are many limitations in the wallet.dat method; for instance, typically only one software application should access the wallet.dat file at a time, to prevent data conflict and corruption.

The Iguana Core codebase enables the user to interact with their funds on the blockchain(s) without requiring a wallet.dat file. Because the Iguana Core codebase works with raw transaction data, the codebase allows a user to first create and then manage a public blockchain “smartaddress” that can be accessed from anywhere, by any compatible standalone GUI, simply with a passphrase that unlocks their privkey.

To maintain control over their funds without requiring a wallet.dat file, users need only create a smartaddress and then retain a copy of the accompanying passphrase (typically a collection of 12 to 24 common dictionary words arranged in a specific order) that is provided at the moment of creation. By entering this passphrase into an Iguana Core compatible standalone GUI app, Iguana Core then activates their <privkey>, which then enables users to manage their funds.

Furthermore, the smartaddress created by Iguana Core can manage and maintain multiple types of coins and other blockchain assets. When a user sends any compatible coin to the smartaddress, Iguana Core stores those coins in a separate address that is compatible with the appropriate blockchain, and links this subaddress to the smartaddress of the user.

Therefore, in the underlying Iguana code, each of the unique coins gets an address that is compatible with its own blockchain, but the smartaddress enables the user to access these coins all at once. Funds deposited to this smartaddress are automatically eligible for trading, and therefore a BarterDEX GUI app can work with speed to enable users to trade between a multiplicity of coins.

One key function of the Iguana codebase that makes this possible is the <withdraw> command in the Iguana API. It is this command that allows individual GUI apps, such as a standalone BarterDEX GUI app, to work with the underlying funds in the user’s addresses.

Notice several of the freedoms this provides to the user. All the funds are only spendable by the user with the passphrase, and because there is no need for a wallet.dat file to be stored locally, there is less danger (though users should exercise caution) of data corruption between different standalone software applications all accessing these funds.

Therefore, an end-user can have a standalone BarterDEX GUI app running on their local machine, which they use to trade, and can also have a separate standalone GUI wallet app that is managing their long-term cryptocurrency holdings.

This also allows standalone GUI applications that are Iguana Core compatible to support each other. For instance, while a BarterDEX GUI can run without any native-coin daemon process running in the background simply by relying on Iguana Core and public Electrum SPV servers, the BarterDEX GUI can also work with a native wallet’s coin daemon background process to coordinate blockchain synchronization.

For instance, a Komodo user may run the Komodo Agama wallet, which runs a native Komodo coin daemon (and has a local wallet.dat file), alongside a BarterDEX GUI app. Iguana Core can then enable the BarterDEX GUI to rely on the native coin daemon running in the background of the Komodo Agama wallet, which speeds up the trading process for an end-user, as they do not have to wait for the public Electrum servers to update.

There is one caution with this, however, and it requires a deeper understanding of how Unspent Transaction technology, or UTXOs, work in the bitcoin-protocol (which we will explain later).

To briefly summarize this caution: The important thing for users to understand is that they should be careful not to spend the same funding in two different standalone apps. In other words, if they are trading with funds in a BarterDEX GUI, they should not also try to spend those funds in their Agama Wallet (or another Iguana-compatible wallet). Instead, they should wait for both apps to be in sync before moving forward.

UTXOs: An Elusive, Yet Fundamental Concept

The BarterDEX relies heavily on a rarely understood technology called the “UTXO” (short for Unspent Transaction) which was created in the original Bitcoin protocol. This technology is fundamental to the operations of any blockchain project that utilizes the original Bitcoin protocol. However, even the most active of cryptocurrency users rarely know what UTXOs are or why they exist.

Because UTXOs play an important part in The BarterDEX, and to provide a pleasant user experience, it is essential we adequately explain the UTXO concept. In the future, as the technology surrounding The BarterDEX iterates, and as the cryptocurrency community continues to learn, we hope that the concept of UTXOs will be less taxing on a user’s learning curve.

To begin our explanation of UTXOs, let us first examine the language of a common user when describing how much cryptocurrency money they have and how they perceive that money as a whole. We will therefore need to understand the concept of “satoshis”, the way a blockchain handles the collection and distribution of funds, and how we work with this technology when trading on The BarterDEX.

Comparing the UTXO to Fiat Money

Let us assume a cryptocurrency user, whom we name Michael, has $10,000 in his physical wallet. Naturally, when Michael thinks about the amount of physical (or “fiat”) money he has, he says to himself, “I have $10,000.”

However, there is no such thing as a $10,000-dollar bill. Instead, Michael actually has a collection of smaller bills stacked together. For instance, he could have a stack of $100-dollar bills, the total of which equals $10,000 dollars.

If Michael goes to purchase an item that costs $1, and he only has $100-dollar bills in his wallet, to make his purchase he will take out a single $100-dollar bill and give it to the cashier. The cashier then breaks that $100-dollar bill down into a series of smaller bills. The cost for the item, $1, remains with the cashier, and the cashier then provides change—perhaps in the form of one $50-dollar bill, two $20-dollar bills, one $5-dollar bill, and four $1-dollar bills.

Michael now thinks to himself, “I have $9,999.” Specifically, however, he has ninety-nine $100-dollar bills, a $50-dollar bill, two $20-dollar bills, one $5-dollar bill, and four $1-dollar bills.

We emphasize that not only does he not have ten thousand $1-dollar bills, he also does not have one million pennies ($0.01). Furthermore, because pennies are the smallest divisible unit of value in Michael’s wallet, we could point out that each bill is a collection of its respective units of pennies. For instance, a $1-dollar bill in Michael’s wallet we could describe as, “a bill that represents a collection of one hundred pennies and their value.”

Understanding Cryptocurrencies and Their UTXOs

A Satoshi is The Smallest Divisible Unit of a Cryptocurrency

Continuing with our explanation of UTXOs, we next need to understand the concept of “satoshis.” (The name “satoshi” is derived in honor of the anonymous person(s) who wrote the original Bitcoin whitepaper.) By convention in the cryptocurrency community, one satoshi is equal to one unit of a coin at the smallest divisible level. For instance, 1 satoshi of Bitcoin is equal to 0.00000001 BTC.

Let us suppose now that Michael has 9.99000999 BTC (Bitcoin) in his digital wallet. Assuming Michael correctly understands the concept of satoshis, Michael could say to himself, “I have nine hundred and ninety-nine million, nine hundred and ninety-nine satoshis of bitcoin.”

This is how Michael might mentally perceive the collection of money that exists in his digital wallet, similar to how he perceives the $9,999 in his fiat wallet.

A UTXO is a Packet of Satoshis, just as a Fiat Dollar Bill is a Packet of Pennies

Recall now that with fiat money, Michael did not think about how his original $10,000 was comprised of smaller individual $100-dollar bills. Similarly, Michael also does not think about how his 9.99000999 BTC could be comprised of smaller collections of satoshis.

Furthermore, just as Michael did not carry around fiat money as a collection of pennies, he also is not carrying around a raft of satoshis. Were he to try to carry a million pennies in his physical wallet, the weight of the wallet would be unmanageable. Similarly, if the Bitcoin protocol were to attempt to manage nine hundred and ninety-nine million, nine-hundred and ninety-nine satoshis, the “data weight” would be so heavy, the Bitcoin protocol would be enormous and unmanageable.

To optimize “data weight,” the Bitcoin protocol therefore bundles up the satoshis into something that is similar to the example of dollar bills earlier, but with one important difference. (In fact, here is where the Bitcoin protocol exercises a superiority over fiat money by deviating from the way that fiat money bundles smaller values into larger values.)

In fiat money, one hundred pennies are bundled into a one-dollar bill, which can then be bundled into a larger bill, and so on. All the sizes of fiat money are preset and predetermined by the issuer of the fiat money when they print their bills and coins.

The Bitcoin protocol, however, does not need to pre-plan the sizes of “bills” (i.e. the collections of satoshis) in the owner’s wallet. Bitcoin is freer in this sense; it can shift and change the sizes of its “bills” at will because there is no need to accommodate for the printing of physical coins and paper.

Instead, the Bitcoin protocol allows for the developer of digital wallets to write code that can optimize how bitcoin satoshis are packaged into “bills,” and thus the community of developers can work together to keep the data weight of the blockchain manageable. The better the digital-wallet developer, the more efficient the size of the “bills” (a.k.a. the packets or collections of satoshis).

The Bitcoin protocol does have one limitation, however: It must keep track of how these satoshis are being collected into larger “bills” in everyone’s digital wallets. After all, this is the very idea of Bitcoin: everything happens under the public eye, where it can be verified. Because the Bitcoin blockchain must keep track of the sizes of these packets of satoshis, the only time the packets can be assembled or disassembled into larger and smaller sizes is at the moment when the user is spending money on the public blockchain. This is the moment when the user is under the public eye, and therefore his actions need to be verifiable. (To compare this to fiat money, consider the effect created were Michael to cut a $100-dollar bill into smaller pieces: the $100-dollar bill would no longer be considered a valid form of currency.)

Because bitcoin is so flexible, and because the word “bill” does not well describe this flexibleness, we do not use the word “bill” in cryptocurrencies. In fact, we have yet to come up with a sonorous word for a packet of satoshis. For the time being, we simply use the word that the original bitcoin developers provided, UTXO, which stands for “Unspent Transaction.” As mentioned, the packet can be any size; the developer of the spender's software wallet application decides on this process. Most importantly, and to reiterate, a UTXO can only be resized during the process of spending, as this is the moment when the user interacts with the public blockchain.

To further clarify this, let us return to Michael's example with fiat money. Recall that when Michael went to purchase a $1-dollar item, he only had $100-dollar bills in his wallet. He had to give out one $100-dollar bill, and then receive a broken-down collection of dollar bills in return.

This is exactly how it works with UTXOs. Michael has a collection of UTXOs in his digital wallet, and when he goes to buy something, he will give out UTXOs until he surpasses how much he owes, and then the extra change from the last UTXO used will be broken down and returned to him.

For example, let us suppose that Michael's 9.99000999 BTC is comprised of three UTXOs worth the following values:

UTXOs in Michael's Wallet

UTXO #1:	0.50000000 BTC
UTXO #2:	0.49000999 BTC
UTXO #3:	9.00000000 BTC
Total:	9.99000999 BTC

Michael now desires to purchase an item that costs 0.60000000 BTC. He will have to hand out enough UTXOs from his wallet until he covers the costs of this transaction, just as he would if he were using fiat money. The Bitcoin protocol calculates the change from the transaction and then returns his change to him.

Remember that there is a fee when spending money on a blockchain. Since we are using Bitcoin in this example, the fee would be paid to cryptocurrency miners. Let us imagine that the fee the miners charge Michael is 999 satoshis.

We begin by looking at how Michael would see the process of making the purchase, assuming he does not understand the concept of UTXOs. For now, Michael only understands how much is in his wallet at the satoshi level as he conducts his transaction:

  9.99000999 BTC - The amount Michael initially owns

- 0.60000000 BTC - The amount Michael sends to the digital cashier for his purchase

- 0.00000999 BTC - The network fee paid to miners

9.39000000 BTC - The amount left in his wallet

This deduction for his purchase all appears very simple to Michael—a testament to the Bitcoin protocol's effective design.

In the background, however, the digital wallet handles the UTXOs and the change process in a manner as determined by the programmer. In Michael's example, let us assume that it proceeds this way:

The wallet first brings out UTXO #1, which is worth 0.50000000 BTC:

0.60000999 BTC - The total amount that Michael owes to the cashier and network

- 0.50000000 BTC - The wallet sends the full value of **UTXO #1** to the digital cashier

0.10000999 BTC - This is the remaining total amount that Michael still owes

The wallet now brings out UTXO #2, which is worth 0.49000999 BTC:

This UTXO is broken down or shattered into smaller pieces.

0.49000999 BTC - The size of Michael's **UTXO #2,** now in the process of change

- 0.10000000 BTC - This shatter of UTXO #2 goes to the cashier (payment fulfilled)

- 0.00000999 BTC - This shatter of UTXO #2 pays the network fee to the miners

0.39000000 BTC - This last shatter now returns to Michael's wallet as a new UTXO

Michael now has one new UTXO in his wallet, and it is worth 0.39000000 BTC:

Michael's New Wallet State:

UTXO #3:	9.00000000 BTC
UTXO #4:	0.39000000 BTC
Total:	9.39000000 BTC

If Michael wants to buy something later, these UTXOs will have to be broken up once more, according to the costs and programming of the digital wallet. Again, whatever is left over from his last UTXO comes back to his own wallet as a new UTXO.

Now let us suppose that Michael receives 0.4 BTC from someone else. In Michael's wallet, he will see a total of 9.79 BTC. However, in his wallet there are now actually three UTXOs:

Michael's New Wallet State:

UTXO #3:	9.00000000 BTC
UTXO #4:	0.39000000 BTC
UTXO #5:	0.40000000 BTC
Total:	9.79000000 BTC

As a result, the number and sizes of UTXOs in Michael's wallet will vary over time. He may have many smaller UTXOs that make up his full balance, or sometimes he might just have one large UTXO that comprises all of it. For Michael, it is normally possible to ignore this since the wallet developer could handle everything automatically.

However, understanding the nature of The BarterDEX currently encourages users to understand UTXOs, as the process relies on their UTXO inventory during trading, as explained below.

Trading on The BarterDEX

From our point of view as developers, the most difficult aspect of creating The BarterDEX was in matching the inventory of UTXOs between trading partners.

To illustrate this complexity, let us briefly return to the example of Michael and fiat currency. If Michael had only a $50-dollar bill in his wallet, and wanted to spend $35 dollars at a video arcade. He needs to trade $35 for the equivalent number of video-game tokens. However, he can only work with the bill that is in his wallet to trade for the tokens.

In a typical arcade, this process is simple. There are just two currencies—his dollars and the video game tokens—and he will have a human cashier available to manage the trade. He gives the $50-dollar bill to the human cashier, and the cashier returns $15 dollars in dollar bills, and $35 dollars' worth of video-game tokens.

In creating The BarterDEX, however, our goal is to decentralize all points of control. (The "cashier," in this sense, is a centralized authority who could be corrupted or could commit human error). That means that we cannot have a human cashier present in The BarterDEX to trade Michael’s three UTXOs into their appropriate sizes when he wants to swap for other currencies.

A further challenge lies in the number of currencies. For The BarterDEX there are not just two coins, but myriad currencies, with many users, each having a variety of unique UTXO sizes in their wallets. Furthermore, the trading happens in real-time, through automation, on a decentralized peer-to-peer network, supporting a countless number of separate blockchain projects, while providing a speed and (eventually) liquidity comparable to that of a centralized exchange. All of this must be accomplished while maintaining a level of security and safety that only decentralization can provide.

Finally, imagine if there were no cashier to break down Michael’s $50-dollar bill. What if instead, he had to approach other arcade customers to barter for their tokens? This would create a difficult scenario for Michael.

In its current iteration (continuing the use of the $50-dollar metaphor as applied to UTXOs), we limit The BarterDEX’s capability to only perform a trade for Michael’s $50-dollar bill in exchange for the currency that another customer holds. The BarterDEX does not provide a service whereby Michael can break down his $50-dollar bill into a convenient set of $10-dollar and $5-dollar bills for trading. He must give up his full $50-dollar bill for whatever he wants in return.

The process of breaking down UTXO inventory, therefore, is both in the hands of the user and in those of the developers creating the standalone GUI apps. We are working with our community to simplify this process. Naturally, it is complex and will take time. Therefore, we recommend that users who engage with The BarterDEX have a basic understanding of their UTXO inventory and how they are bartering with other users before using it.

How The BarterDEX Deals with Order Offers and UTXOs

When a BarterDEX user offers a trade to the network, the BarterDEX protocol itself does not prioritize the total number of satoshis that the user offers. Instead, The BarterDEX simply looks through the user’s inventory for the largest-sized UTXO that is below the amount the user offered.

For example, let us suppose that Michael has 100.01287001 KMD (Komodo coin) in his wallet. It is comprised of three UTXOs:

Michael’s Initial Wallet State:

UTXO #1:	90.00000000 KMD
UTXO #2:	00.01287001 KMD
UTXO #3:	10.00000000 KMD

Total:	100.01287001 KMD

Michael wants to trade 50.00 KMD on the BarterDEX network. He puts out an order for an alternate cryptocurrency called MNZ (Monaize), and he wants to exchange in a 1:1 ratio.

The BarterDEX itself will not attempt to manage for Michael's misunderstanding of his UTXO inventory. (The developer of Michael's standalone software could try to help him, but that is a separate matter.) Rather, The BarterDEX will simply look through his inventory for the largest UTXO that is below the total amount he offered. In this example, The BarterDEX will select his UTXO #3, worth 10 KMD. The BarterDEX will then calculate the necessary fee, which so happens to be exactly equal to the amount of UTXO #2: 0.01287001 KMD.

The BarterDEX can then take these two UTXOs and facilitate a trade for MNZ in a 1:1 price ratio. Michael's final wallet appears as so:

Michael's Final Wallet State:

UTXO #1:	90.00000000 KMD
UTXO #3:	10.00000000 MNZ
Total:	90.00000000 KMD
	10.00000000 MNZ

It is up to Michael, or to the creators of any standalone GUI wallet, to manage the UTXOs. The BarterDEX only manages the matching of the UTXOs once they are created.

Detailed Explanations of The BarterDEX Process

With an understanding of the specifics of what The BarterDEX is actually trading, we can now approach an explanation of how the trading procedure occurs.

Atomic Swaps on The Komodo BarterDEX

To facilitate trading among users, The BarterDEX implements a variation of the Atomic Swap protocol as described by Tier Nolan on BitcoinTalk.org. The original concept provided by Tier Nolan can be said to be "ahead of its time," as it is both complex and relies conceptually on technology that yet does not exist. Therefore, to create our Atomic Swap protocol, we adapted for the current technology. A thorough study of Nolan's original exposition can provide a solid background into the tradeoffs that we made as we selected our final version of the atomic-swap protocol.

We emphasize to the reader that the key aspect that we maintained from the original concept is that at each step there are both incentives to proceed to the next step in the proper manner, and disincentives to avoid abandoning the procedure. With this structure in place, regardless of where the protocol stops, each party receives their proper reward. If a party attempts to deviate from the proper path, their funding is penalized to the point of eliminating any potential rewards a user could gain by acting maliciously. These incentives and disincentives create the foundation for the requisite trustless nature of our atomic-swap protocol.

Meet Alice and Bob

To understand why the atomic-swap protocol is necessary, it is first important to recall that computer code is executed in linear fashion. Even if we were to assume that both parties in a trade may be honest, on a computer the process of taking money from each digital wallet and pulling the money into the open must happen one wallet at a time. Therefore, one person must send out their money first. The atomic-swap protocol protects that person from vulnerability. Without the atomic swap, any malicious party involved (whether it be a full-relay node, trading partner, or other external agent) would be able to destroy the fairness of the trade.

There are two parties in an atomic swap: the liquidity provider and the liquidity receiver. Once the process of an atomic swap begins, the behavior of each party’s public trading profile is recorded and added to their reputation on The BarterDEX network.

The process of an atomic swap begins with the person who makes the initial request—this is the liquidity receiver. Let us call this person, “Alice.” Alice will need two UTXOs to perform her swap. One UTXO will cover the protocol fee, which is roughly 1/777th the size of her desired order. We call this fee the <dexfee>, and its primary purpose is to serve as a disincentive to Alice from spamming the network with rapid requests.

The second UTXO required of Alice is the actual amount she intends to swap. The BarterDEX first verifies that she has these funds, but for the moment she retains these funds in the safety of her own digital wallet.

On the other side of the atomic swap, we have the liquidity provider—we call this person, “Bob.” Bob sees the request on the network for Alice’s atomic swap and decides to accept the trade. Now his part of the process begins.

To complete the trade, he must also have two UTXOs, but with one important difference: the first UTXO is equal to 112.5% of the amount that Alice requested; the second UTXO is exactly equal to the amount that Alice intends to swap. In other words, Bob must provide liquidity of 212.5% of the total amount of the currency that Alice requests.

The first UTXO (112.5%) Bob now sends out as a deposit, placed on the BarterDEX network; the network’s encryption holds the deposit safely in view, but untouchable. We call this UTXO

<bobdeposit>. It will remain there until his side of the bargain completes in full, or until Alice's request for a swap times out. Assuming Bob keeps his promises and stays alert, these funds will be automatically returned to him at the appropriate moment.

The second UTXO (100%) he retains within the safety of his own wallet for the moment.

Performing a successful connection between Bob and Alice, and verifying their requisite UTXOs, is the most complex and difficult aspect of creating the BarterDEX network. Myriad factors are involved in a successful attempt for Bob and Alice to connect: human motivation; the experience level of the users; economics; connection technology; user hardware setups; normal variations within Internet connections; etc.

We emphasize to users here that the process of performing these actions over a peer-to-peer network has almost an artistic element to it. An attempt to successfully connect Bob and Alice can be thought of more like fishing, where we must simply cast and recast our line until we successfully connect with our target. As The BarterDEX continues to iterate and improve, and as the number of users increases, we expect any required effort to lessen for users, the network, and the BarterDEX GUI apps.

Alice and Bob Make a Deal

Assuming Alice and Bob have now successfully connected, the process from this point forward becomes quite simple:

(Note: in some cases, it is possible to perform an atomic swap with fewer steps, but for the sake of brevity we will focus only on this scenario.)

From the beginning:

1. Alice requests a swap and sends the <dexfee> to the BarterDEX full-relay nodes
    a. The full-relay nodes receive her request and publish it to the network
2. Bob sees the request on the network, accepts it, and sends out <bobdeposit>
    a. <bobdeposit> enters a state of limbo on the BarterDEX network, held safely by encryption, awaiting either Alice to proceed, or for the swap to time out.
        i. If the latter occurs, <bobdeposit> is automatically refunded to Bob via the BarterDEX protocol.
3. Alice now sends her <alicepayment> to Bob
    a. She does not send the payment to Bob directly, but rather into a temporary holding wallet on the BarterDEX exchange
        i. Only Bob has access to this wallet, via the set of privkeys that only he owns.

        ii. However, the BarterDEX code does not yet allow Bob to unlock this temporary holding wallet; he must continue his end of the bargain first.

        iii. The <alicepayment> will remain in Bob's temporary holding wallet for a limited amount of time, giving him the opportunity to proceed.

4. Bob now sends his <bobpayment> to Alice
    a. Again, this is not sent to Alice directly, but rather into yet another temporary holding wallet.
    b. Likewise, only Alice has access to the necessary privkeys for this wallet.
    c. <bobpayment> will automatically be refunded if she does not complete her part of the process
5. Alice now "spends" the <bobpayment>
    a. By the word "spends", we simply mean that she activates her privkeys and moves all the funds to another wallet—most likely to her smartaddress wallet, as explained earlier.
    b. The BarterDEX registers that Alice's temporary holding wallet successfully "spent" the funds.
6. Bob "spends" the <alicepayment>
    a. Likewise, Bob simply moves the entirety of the <alicepayment> into a wallet of his own—again, it will most typically be his own smartaddress.
    b. The BarterDEX now knows that Bob also successfully received his money.
7. Seeing both temporary holding wallets now empty, the BarterDEX protocol recognizes that the atomic swap was a complete success. The BarterDEX now refunds <bobdeposit> back to Bob and the process is complete.

While it may seem inefficient to have seven transactions for a swap that could be done with two, the complexity of this process provides us with the requisite "trustless-ness" to maintain user safety.

Incentives and Disincentives to Maintain Good Behavior

As we will now explain, at every step along the way there are incentives for each side to proceed, and there are various financial protections in place should one side fail. Also, because payments are sent to these "temporary holding wallets" that exist within the BarterDEX protocol, the protocol itself can assist in the process of moving money at the appropriate steps. Let us now examine the incentives and disincentives at each step.

1 - Alice Sends <dexfee>

If Bob accepts the offer to trade, but does not send <bobdeposit>, Alice only stands to lose her <dexfee> UTXO. This is only 1/777th of the entire transaction amount, so she loses very little. Bob, on the other hand, stands to lose more. Since Bob did not follow through with his end of the bargain, the BarterDEX network indicates on his public BarterDEX trading profile that he failed in a commitment, thus decreasing his profile's reputation. If Bob continues this behavior as a habit, he may find it difficult to discover trading partners.

So long as the frequency of "Bobs" failing is low, the occasional extra <dexfee> paid by an Alice is a minor issue. However, if there is a sudden spike in misbehavior, the BarterDEX code has in-built contingency plans which can provide refunds to Alice(s), should a particular Alice node(s) experience a large loss via <dexfee>'s. This contingency plan is built into the DEX asset-chain infrastructure.

2 - Bob Successfully Sends <bobdeposit>

If Alice does not follow with her next step, the <alicepayment>, then Alice loses not only the <dexfee>, but she also receives a mark on her public BarterDEX profile. She gains nothing, and Bob has no reason to fear as <bobdeposit> will automatically return to him via the BarterDEX protocol.

3 - Alice Successfully Sends <alicepayment>

If Bob does not proceed with his next step, the <bobpayment>, then after 4 hours Alice can simply activate a BarterDEX protocol that will allow her to claim <bobdeposit>. Recall that <bobdeposit> is 112.5% of the original intended trade; Bob has every incentive therefore to continue with his end of the bargain, and Alice has nothing to fear should Bob fail. She even stands to gain a 12.5% bonus, at Bob's expense.

4 - Bob Sends <bobpayment>

Now, if Alice does not follow by "spending" the <bobpayment> (i.e. taking the money out of the temporary holding wallet and into her own smartaddress), then after 2 hours Bob can activate a BarterDEX protocol that allows him to reclaim his <bobpayment> immediately. Furthermore, four hours later Bob may activate a refund of <bobdeposit>; Bob is safe from Alice, should she fail. For Alice, the BarterDEX protocol allows Alice to reclaim her <alicepayment> after Bob reclaims both of his payments.

Everything herein is recorded to the respective users' BarterDEX trading profiles, ensuring their reputations are on the line. Recall also that the BarterDEX protocol requires each step to be performed in the proper order, thus ensuring that neither party can take any funds before the users' appropriate moment.

Thus, at this integral stage of the process, every step of the path is intricately interconnected and maintains various levels of protection.

5 - Alice Spends <bobpayment>

At this point she is entirely through with any risk to her reputation, her <dexfee> payment, or of the loss of her time.

If Bob does not follow by also “spending” the <alicepayment>, it is of no concern to Alice because she has already received her funds. If Bob is simply sleeping and forgets to spend the <alicepayment>, he can only hurt himself.

Naturally, for Bob this is slightly dangerous; Bob’s best course of action is to remain alert and spend the <alicepayment> once it is received.

If after four hours, Bob is still sleeping, Alice can activate a protocol that allows her to claim <bobdeposit> (this prevents the funds from remaining frozen in the chain forever). In this scenario, she receives both the <bobpayment> and <bobdeposit>, at only the costs of the <alicepayment> and <dexfee>.

Bob can still make a later claim for the <alicepayment> when he regains his awareness.

6 - Bob Spends <alicepayment>

Assuming all has gone according to plan, and having spent the <alicepayment>, Bob may now reclaim <bobdeposit>. Just as before, if Bob does not refund his own deposit, it is his loss; in four hours Alice will be able to activate a claim on <bobdeposit>.

7 - Bob Reclaims <bobdeposit>

The process is complete. Alice received the <bobpayment>; Bob received the <alicepayment>; Bob has <bobdeposit> back in his own possession; the entire process only cost Alice the original <dexfee>. At each step along the way, the side that needs to take the next step is motivated to do so, with greater and greater urgency until the process is complete.

Additional BarterDEX Atomic Swap Details

The BarterDEX implements the above series of commands in a cross-platform manner, enabling users to atomic-swap trade with hundreds of coins of many types, including both native coin daemon's and those that run on SPV Electrum servers. A swap that is not completed immediately can carry on as long as the time has not expired within the BarterDEX protocol.

Naturally, users must understand that outside forces can disable the process and thereby damage one of the users. For instance, an Internet outage for Bob could be particularly dangerous. Therefore, users are advised only to trade manageable sums that they are willing to put at risk, and only with nodes that have reliable reputations.

This atomic-swap protocol, with all its cryptographic validations and intricate key exchanges, is less than half of the difficulty Komodo experienced in creating The BarterDEX. Relatively speaking, it is "easy" to do an atomic swap in isolation between two test nodes, using UTXOs that are carefully prepared for the test.

It is an entirely different matter to open this up to the public at large, including the enabling of our orderbooks and order-matching features. Due to the peer-to-peer nature of The BarterDEX, on a live network it is impossible to guarantee that a user that indicates they would like to begin a swap will receive a successful reply.

For instance, a Bob may see a potential swap that he would like to make, but by the time his attempt to accept the swap crosses the expanse of The Internet, someone else could have already accepted the swap, thus leaving Bob in his original position. There are legion scenarios wherein the initial connection can fail; once the connection is made, however, the rest of the process maintains reliability and user safety.

Failed connection attempts only result in the loss of a few seconds of the user’s time, and there is no cost associated with making a failed initiation attempt; the <dexfee> paid by an Alice never occurs, and the BarterDEX disregards Bob’s attempt to send <bobdeposit>.

Therefore, while we cannot guarantee that The BarterDEX can initiate an atomic swap that appears to be available will ultimately connect, we can offer comfort in knowing that the users’ losses in these scenarios are insubstantial.

A More Detailed Explanation of The Atomic-Swap Connection Process

The following is a brief explanation of the complex process by which The BarterDEX establishes a connection between Alice and Bob.

For The BarterDEX to accept a request to begin an atomic swap, the code first needs to register and create all the necessary backend elements for the <dexfee>, <AlicePayment>, <BobDeposit>, and <BobPayment>. All four must be specified before The BarterDEX can indicate to Alice and Bob that The BarterDEX can successfully support this atomic swap.

This is more complicated than it appears. As we explained earlier, most users do not understand the true nature of how funds operate in a cryptocurrency. Rather, most simply view their balance as a single conglomerate of coins that they can spend at the “satoshi level.” This misperception is important to correct to understand how The BarterDEX performs an atomic swap.

Naturally, because users have varying sizes of UTXOs in their wallets, the true challenge in creating The BarterDEX was to create a method of maintaining a network that would coordinate each user’s list of UTXOs in their wallets, and to allow them to match with other users in trading pairs. In addition, The BarterDEX also automatically calculates the appropriate mining and transaction fees for the blockchains involved, according to a speed that maintains an optimized atomic-swap process.

As we created the necessary code to make the atomic swap possible, we found that it is not practical to have the user specify which UTXO pair they have sitting in their wallet when choosing to make a swap. This would also not be intuitive for the user. Furthermore, we did not even want to code a way for an Alice to know the UTXOs a Bob has available at the moment of negotiating a trade.

Instead, here is how The BarterDEX deals with the complexity of matching these unbroken and mismatching UTXOs to process an atomic swap. It is important to note that users do not have to have a sophisticated understanding of the backend UTXO process, and may simply trade using either a minimal understanding of UTXO inventories, or at least rely on the support of a cleverly coded standalone BarterDEX GUI app.

Assuming Alice has already indicated she desires to perform an atomic swap, The BarterDEX calculates out the proper divisions of her UTXOs, defines how they will be appropriated during the process, and sends an “Alice Request" to Bob with information regarding her pair of UTXOs (which are the <dexfee> and the <AlicePayment>). Also, The BarterDEX verifies her desired price and volume.

Bob, the human user (or an artificial intelligence bot acting on his behalf), indicates that he is willing to accept the trade. The automation of the BarterDEX Bob-side protocol now takes over in the background. It validates the "Alice Request" to make sure the UTXO pair is valid, and then the Bob-side protocol scans through Bob’s UTXO inventory for the most efficient way to create both the <BobPayment> and <BobDeposit> UTXOs.

The Bob-side protocol understands that the UTXOs will not perfectly match, and it will therefore calculate the most efficient method of making any “spare change” UTXOs as needed. An additional constraint the protocol needs to consider is that the result must match the price and volume Alice wants to pay. Finally, it accounts for the requirement that <bobdeposit> be at least 12.5% bigger than the “Alice Request.” (Note that The BarterDEX is directly involved with managing Bob’s UTXOs, but is not involved with managing Alice’s UTXO offers.)

Once The BarterDEX verifies all these conditions, the Bob-side protocol sends back a data packet, labeled “reserved,” to the Alice-side protocol to indicate that all is in order. All of this is optimized and conducted in a manner that prevents the human Bob from having his funds frozen in an unnecessary deposit duty, should the human Alice find another “Bob” in the interim.

Next, the Alice-side protocol validates the "reserved" packet from the Bob-side protocol, making sure all the UTXOs are valid, and the protocol verifies that the price and volumes are acceptable according to the original intent.

Assuming everything successfully validates, the Alice-side protocol sends a "connect" packet back to the Bob-side protocol with the same parameters, indicating that her funds are now "reserved” as well.

Between the "request" being sent and the "reserved" packet being received there is a 10-second timeout which prevents Alice from making further trade requests; this gives The BarterDEX the time necessary to perform all the calculations.

(Note: This 10-second timeout also provides a contribution to what we call “whale resistance” during the Komodo dICO process. Whale resistance is a way Komodo and The BarterDEX resist wealthy financial technocrats from purchasing an entire coin supply and thus forcing an artificial market scarcity. Read our dICO explanation to learn more.)

The Bob-side protocol now validates Alice’s "connect" packet and, assuming everything is in order, the protocol starts a new Bob-side thread of code, thus beginning the actual atomic swap. The Alice-side protocol also receives the "connect" packet, verifies, and then starts an Alice-side thread of code.

There is one more "negotiation" step that is needed between the Alice-side and Bob-side protocols: in the event the two sides to the protocol do not achieve consensus, the entire atomic swap aborts without any payments sent from either party (i.e. “no harm, no foul”).

(This final negotiation could have been included earlier, but due to the way the atomic swap organically developed during our creation process, it ended up inside the atomic-swap protocol itself.)

The Alice-side and Bob-side protocols have now properly performed their duties, and thus completed the most challenging aspect of the atomic-swap protocol. The BarterDEX returns control to the humans (or bots acting on their behalf) to send their respective payments.

The DEX Fee: <dexfee>

People will notice that there is a small <dexfee> required as part of the BarterDEX protocol. This is 1/777 of the transaction amount and it is calibrated to make spam attacks impractical. By forcing a would-be attacker to spend real money, attacking the network becomes costly. Without this spam prevention, the BarterDEX would otherwise be DOS attacked at the protocol level.

The 1/777 ends up being equal to 0.1287% of the <alicepayment>; this is already far less than the fees paid on centralized exchanges. Also, centralized exchanges charge both sides of the trade, so even if they charge you 0.2%, they are actually harvesting 0.4% in total fees between both parties. Furthermore, they often have fees and limitations for withdrawing funds; The BarterDEX has neither.

It is possible that some atomic swaps can initiate, and then fail to complete, which raises questions about what happens to the Alice <dexfee>. The <dexfee> is the first charge in the protocol; in this sense, there is a <dexfee> charged for these failed atomic swaps.

However, this failure should not be looked upon in isolation. The BarterDEX protocol is based on statistics. Statistically speaking, there will be some percentage of atomic swaps that start and will not complete. Let us suppose a 15% failure rate at this stage of the atomic swap (15% is three x’s higher than the rate of failure we currently observe in our testing). Even in this scenario, the effective <dexfee> cost is still only 0.15% to all “Alices” as a whole.

Therefore, if you experience the loss of a <dexfee> transaction for an atomic swap that fails to complete (which would be due to a failure to receive a response from Bob), know that this is all part of the statistical process. If you find yourself paying more than 0.15% of your completed trades in <dexfee>'s, please let us know. This would be a highly unusual statistical outlier, and we will therefore want to find and fix the cause.

As an organization, when speaking generally to our audience online, we typically state that the <dexfee> is just 0.15%. In this manner, we hope to create the expectation that 0.15% is normal; if the network performs perfectly, on the other hand, users will get a blessing in the form of a lower fee, 0.1287%.

Dealing with Confirmations

Since The BarterDEX is trading permanently on blockchains (as opposed to updating an internal database of vouchers, or managing a proxy-token account balance), both humans need to wait and watch as miners on the respective blockchains calculate transaction confirmations.

Because the payments that occur on one blockchain will proceed regardless of the actions on the other blockchain (i.e. a confirmation failure on one chain will not stop with the other blockchain performing its duties as normal), it is therefore important that The BarterDEX protocol observe and adjust as necessary. Each side of the BarterDEX protocol (Bob-side and Alice-side) therefore watches and attempts to provide a level of protection for the human users.

The BarterDEX achieves this protection by an array of <setconfirms> API calls, which gives each side the option to specify how many confirmations they expect before the automated process should be satisfied on behalf of the human users' interests. The setting for the <setconfirms> feature must be decided before the atomic swap begins, as the number of confirmations the users choose will persist until the process completes. If the users have differing preferences for the total <numconfirms> they prefer, the BarterDEX protocol automatically sets the larger of the two preferences as the requirement for both parties. Furthermore, this feature also includes a <maxconfirms> value to prevent one side from specifying an unreasonable or malicious number of required confirmations.

Zero Confirmations

The BarterDEX also supports a high-speed trading mode. Using this feature, a user can activate an extremely fast mode of trading: <zeroconf>. This initiates a form of atomic-swap trading that does not wait for any confirmations at all. When using this feature, atomic swaps can be completed in as little as 3 seconds. This is a high-risk endeavor, naturally, and users should exercise extreme caution when implementing it.

One potential application for the <zeroconf> feature is to allow groups of individuals to form their own organizations where they decide personal trust levels, and work together to correct any mistakes that are made in their accounting endeavors.

The BarterDEX also features a special Trust API that users can enable for themselves and groups that they form to indicate how much they trust different traders. By default, the Trust API is set to neutral for all users. A group of users can form their own organization and develop a trusted network for trading, using the Trust API to set each other's trader profile to Trust = Positive. In such cases, if a user, or a group of users, tells the Trust API to set another trader profile to Trust = Negative, that trader's <pubkey> is blacklisted for any of the participating individuals or groups.

High-Speed Mode: An Experimental Feature Using Time-locked Deposits

Using the <zeroconf> protocol, we developed a new feature for The BarterDEX network that is functional, but still experimental. This "High-Speed Mode" adds one additional step to the Alice and Bob process.

Alice places a one-time security deposit of an amount equal to or greater than the amount she would like to actively trade. This security deposit is sent to a conditional p2sh wallet address (currently controlled by the Komodo team). Alice indicates within her security-deposit transaction the amount of time the deposit should remain in the wallet. The p2sh wallet will lock the funds from Alice's end until the completion of the expiration date, though the wallet will allow the Komodo team to access the funds if necessary. This is called a "time-locked deposit." After her chosen date of expiration, she can reclaim her security at any time. Note that her KMD funds continue to accrue interest at the normal 5% APR.

This enables Alice to participate in our experimental High-Speed Mode feature, a fully automated protocol that tracks users' trading activities and monitors their unconfirmed swaps against their time-locked deposits. While using High-Speed Mode, Alice can trade funds with participating "Bobs" in amounts smaller than her time-locked deposit. Her trading partners dynamically decrease her trust level as she trades, monitoring the amount of her unconfirmed transactions against her total security deposit. Should she reach an unconfirmed trading capacity that is roughly equal to the amount in her deposit, the protocol blocks her from participating in the High-Speed Mode feature until her funds obtain more clearance through notarization on their respective networks.

Should Alice attempt to cheat during any period of zero confirmations, the Komodo team can activate the p2sh wallet security deposit and deduct the amount of her offense, and a penalty fee, from her security deposit to compensate the affected parties. The remainder will be available for her to reclaim at the date of the original expiration, at the latest.

With the security deposit in place, Alice can use the High-Speed Mode feature to complete a trade in as little as three to five seconds. Note that this feature is new, highly experimental, and we recommend users exercise extreme caution when participating. If a user cannot activate High-Speed Mode, The BarterDEX defaults to the normal, non-<zeroconf> atomic-swap trading method.

Realtime Metrics

Nodes on The BarterDEX use Realtime metrics (RTmetrics) to filter the possible candidates for atomic-swap matching. All nodes track global stats via a <stats.log> file. This log file allows each node to self-update the list of pending swaps on the network. By nature, The BarterDEX protocol has filters that give less priority to nodes that are already occupied. Additionally, the "Alice-side" protocol gives less preference to "Bob-side" protocols that do not have enough UTXO sizes visible in the orderbook. This is a new feature, and we expect to optimize and enhance it in future iterations.

Orderbook Propagation

When considering how prices compare between two cryptocurrencies, The BarterDEX uses the convention of "base/rel," which can be translated as "base currency to relevant currency." The price is calculated by determining (base currency)/(relevant currency). The relevant currency is the cryptocurrency Alice is using to make the initial purchase, and the base is the currency Alice intends to buy.

To construct a public orderbook, a node needs to have price information. Since The BarterDEX communicates primarily by means of pubkeys, the price for each currency must naturally be obtained from a pubkey. In the long run, for orderbook performance, we will need a specific <txid>/<vout> for each node, as each individual node could have hundreds of UTXOs. Currently, propagating all this information globally would use an excessive amount of bandwidth, so we therefore use a different solution.

The BarterDEX instead uses a hierarchical orderbook, where the skeleton of the orderbook is simply the (pubkey)/(price) for any particular (base)/(relevant) pair.

Note: this means that purchasing a cryptocurrency at the (base)/(relevant) price is directly comparable to selling the cryptocurrency using (relevant)/(base) at a ratio of 1/(price).

Using the (pubkey)/(price) pairing, all that is needed to populate the orderbook skeleton is for nodes to broadcast their pubkey and price for any particular (base)/(rel) pair. Nodes that are running a local coin daemon therefore broadcast their lists of UTXOs, which helps to propagate the orderbook. All of this is done in the background, on-demand.

Critical information is broadcast with fully signed encryption to prevent spoofing, thus all nodes can verify the smartaddress associated with a pubkey. In this way, nodes can validate the price broadcasted. (The electrum SPV coins have their own specific SPV-validation process for all UTXOs before they can be approved for trading on The BarterDEX.)

While all nodes could broadcast their UTXO lists constantly to keep them updated, this would result in the network rapidly being overrun with congestion. To eliminate this issue, The BarterDEX simply relies on the (pubkey)/(prices) as this is all that is necessary to maintain useful orderbooks.

Since there are N*N possible orderbooks (given *N* currencies), it is not practical to have The BarterDEX configured to update all possible orderbooks constantly. Instead, orderbooks are created on the user end when requested from the raw public data. During orderbook creation, if the top entries in the orderbook do not possess any listunspent data, a request is made to the network to gather this information.

This process ensures that by the time a trade completes, there is already a request for an orderbook, which in turn requests the listunspent data for the most likely pubkeys. The actual order-matching process then iterates through the orderbook, scanning all the locally known UTXOs to find a high-probability counterparty to whom The BarterDEX can then propose a "request" offer. In practice, early users on The BarterDEX can currently experience nearly instantaneous responses, assuming all the parameters are properly met.

A Brief Discussion on The BarterDEX's Future

This concludes a high-level summary of the The BarterDEX protocol as created by the Komodo organization. It is now fully functioning and live, and with the support of our community, we have successfully completed thousands of atomic swaps.

We are now preparing for our upcoming launch of the Monaize dICO. The MNZ token is an e-banking solution, built with our assistance by our partners, Monaize. This dICO date was originally set for November 10th, but upon observing that the standalone BarterDEX GUI apps lagged behind projected development speeds, together with Monaize we temporarily delayed the dICO launch. Once the frontend GUIs perform reliably, we will reset the dICO launch date. The delay is only temporary, and we hope to resume as soon as possible. For more information about the reasoning behind the delay, please refer to this blog post.

When the Monaize dICO officially begins, it will be a trial by fire. Potentially, many atomic swaps will be invoked simultaneously on The BarterDEX. Current stress testing indicates that the expected load is manageable. However, if we receive higher volume levels than we are expecting, it is possible we may experience a crash of The BarterDEX network. For this reason, we encourage all participants to proceed with caution. Every element of The Komodo Ecosystem is still considered to be highly experimental. We provide no investment advice, nor any guarantees of any funds utilized on our network. Use any of our products at your own risk.

Looking past the Monaize dICO, The BarterDEX will continue to evolve. The current iteration has already identified several areas of improvement for the next iteration. Several different GUI systems are under construction by community members, all of which are utilizing The BarterDEX 1.0 API. As we develop The BarterDEX API, we are making sure that future iterations are backwards compatible for developer ease-of-use.

Acknowledgements

References

BarterDEX – A Practical Native DEX (https://github.com/SuperNETorg/komodo/wiki/BarterDEX-%E2%80%93-A-Practical-Native-DEX)

Nakamoto Satoshi (2008): Bitcoin: A peer-to-peer electronic cash system. (http://www.bitcoin.org/bitcoin.pdf)

Mtchl (2014): The math of Nxt forging (https://www.docdroid.net/ahms/forging0-4-1.pdf.html)

King Sunny, Nadal Scott (2012): PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake (https://peercoin.net/assets/paper/peercoin-paper.pdf)

Delegated Proof-of-Stake Consensus (https://bitshares.org/technology/delegated-proof-of-stake-consensus/)

Miers Ian, Garman Christina, Green Matthew, Rubin Aviel: Zerocoin: Anonymous Distributed E-Cash from Bitcoin (https://isi.jhu.edu/~mgreen/ZerocoinOakland.pdf)

Ben-Sasson Eli, Chiesa Alessandro, Garman Christina, Green Matthew, Miers Ian, Troer Eran, Virza Madars (2014): Zerocash: Decentralized Anonymous Payments from Bitcoin (http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf)

Ben-Sasson Eli, Chiesa Alessandro, Green Matthew, Tromer Eran, Virza Madars (2015): Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs (www.diyhpl.us/~bryan/papers2/bitcoin/snarks/Secure%20sampling%20of%20public%20parameters%20for%20succinct%20zero%20knowledge%20proofs.pdf)

NXT Community: NXT Whitepaper (http://wiki.nxtcrypto.org/wiki/Whitepaper:Nxt)

Larimer Daniel, Scott Ned, Zavgorodnev Valentine, Johnson Benjamin, Calfee James, Vandeberg

Michael (March 2016): Steem, An incentivized, blockchain-based social media platform.(https://steem.io/SteemWhitePaper.pdf)

BitFury Group (Sep 13, 2015): Proof of Stake versus Proof of Work White Paper (http://bitfury.com/content/5-white-papers-research/pos-vs-pow-1.0.2.pdf)