CoinShuffle anonymity in the Block chain

Jan-Willem Selij <jan-willem.selij@os3.nl>
University of Amsterdam

July 20, 2015

Abstract

The decentralized crypto-currency Bitcoin allows one to perform online transactions. Due to the public ledger the Bitcoin protocol features, all transactions are inherently visible for everyone. Bitcoin address are merely pseudonymous and can in some cases be linked to identities. This calls for a need where addresses on their own can be used in a more anonymous way. Several methods that involve mixing services tend to unlink addresses from identities, but with mixed results. These third-party services leave parts open where anonymity is still defeated. The CoinShuffle protocol proposes a solution where anonymity between all participants would be secured. This thesis presents the current state of CoinShuffle transaction anonymity in the block chain. A CoinShuffle transaction is explored in the block chain where it can be classified as such using recognition points. The block chain is traversed to get an overview about the usage of the protocol. Several improvements are discussed to make CoinShuffle transactions harder to detect.

1 Introduction

Bitcoin is a decentralized crypto-currency which is used to make online financial transactions [27]. Bitcoin is a virtual currency and can be used to send and receive funds using *Bitcoin addresses*. Bitcoin features a cryptographic method, called the block chain, that acts as a public ledger in which all past transactions are stored. The block chain allows one to trace back a spent amount to the origin of first appearance. This thesis will examine the anonymous effectiveness of a CoinShuffle transaction in the block chain. Anonymity is becoming an important aspect in the Bitcoin environment. The primary concern for anonymity is that Bitcoin addresses can be linked to an identity. With an identity available, every transaction of an individual or organization can be followed. This leaves little room open for supporting certain causes, such as a political mission, without the possibility of being tracked. Another key point is the degree Bitcoins

are interchangeable. Because of the fact that every Bitcoin can be traced back, Bitcoins that have a particular origin, such as a theft, can be rejected or would be perceived as having less value.

To improve anonymity and interchangeability, the input addresses and output addresses of a transaction should have no ties. The addresses should be *unlinked*. Bitcoins can be *mixed* to unlink the input and output addresses. Mixing involves a group of participants that want to mix a certain amount of coins. All participants transfer coins to different addresses owned by the mixing service, and receive them back from other addresses owned by the service. Received the coins back is usually split in multiple transactions over multiple hours or days, making the transactions look like regular transactions. Mixing services suffer from some severe drawbacks. Any coins transferred can be forever lost in case the service goes out of business, the service learns all participant's addresses and usually charge a fee.

The mixing protocol CoinJoin emphasizes the use of a single transaction where the third-party is not able to steal Bitcoins. However, the third-party is still able to learn all participant's addresses. CoinShuffle is a protocol where no third-party service is required and none of the participant's learn each others addresses.

This thesis presents the current state of the CoinShuffle protocol transaction anonymity in the block chain. Section 2 outlines the Bitcoin concepts together with the need for anonymity and the methods that are used to work around the problem. Section 3 consists of the research questions this thesis goes over. Section 4 gives an overview of the related work. Section 5 describes the CoinShuffle mixing protocol. Section 6 is concerned with the methodology used to create a CoinShuffle transaction, where Section 7 explains the transaction characteristics. Section 8 presents the results of a block chain analysis about CoinShuffle transactions on the live Bitcoin network, and possible improvements to the protocol are discussed in Section 9.

2 Background

2.1 Bitcoin

Bitcoin features a public list of transactions in the form of the block chain. The block chain consists of every transaction made in the past and every future transactions will be appended. A Bitcoin address is made out of a cryptographic key pair, namely the private and public key. A transaction consists of sending (a part of) the available funds from an input address to an output address. The single transaction can also consist of multiple input and output addresses. The transfer of coins in the form of a transaction is signed by the private key to verify the address holder is really the person he says he is. This transaction is

now propagated to the Bitcoin network. Transactions are combined in a block together with others by *miners* that perform a cryptographical challenge called Proof-Of-Work. This challenge is a calculation that takes roughly ten minutes to complete. This challenge can not be calculated ahead of time, as the current block includes data from the previous block. The miners require a small fee for doing this work, and are also rewarded coins upon completion. In the case of multiple miners having completed the calculation, this eventually stabilizes in a way that the longest chain is chosen and forks of the chain are dismissed. Completing calculating the block confirms all included transactions, as this is now forever part of the block chain. This also prevents double-spending as it can be checked if certain funds aren't already spent. Because every block is based on the previous one, undoing certain transactions is nearly impossible as this would mean calculating the whole chain again up to the present time.

2.2 Anonymity

One of the methods to measure anonymity is *taint analysis*. Taint analysis traverses the transaction graph and analyses which addresses are the source of a transaction. A high taint value indicates a strong connection between the Bitcoin addresses and a transaction. A strong relation between an address and a transaction likely means that the input and output addresses are not unlinked properly, diminishing the anonymity [26].

Furthermore, transactions can be discriminated based on a high taint value. Taint databases can be maintained that store information about the history of a coin. Subsequently, transactions can be rejected based on taint value. For example, a transaction can be rejected if a coin comes from a theft. This has been seen in the case of the former Bitcoin exchange Mt. Gox rejecting coins having a theft as source [2] [13]. Another example concerns coins having less value than others when traded on an exchange. Buying or selling coins at a reduced value could pose a problem for the adoption of Bitcoin as new users have no knowledge about the potential issues [34].

2.3 Mixing

Anonymity can be improved by performing *mixing*. The purpose of mixing is to *unlink* the input and output addresses. Unlinking the input and output addresses severs the tie between the addresses, which assures a low taint value. A hypothetical Bitcoin mixing service is shown in Figure 1. Mixing involves multiple participants, but do not necessarily need to be available at the same time. Let's say Alice, Bob and Charlie want to mix 1 BTC. All participants create an output address, which is displayed as Alice_2, Bob_2 and Charlie_2. All participants send their 1 BTC to the mixing service. The mixing service then *mixes* the coins by transferring coins from one participant to another. As shown

in Figure 1, Alice receives her coins which were sent by Bob, Bob receives his funds coming originally from Charlie and Charlie receives this likewise coming from Alice. At the end, all participants have received back their 1 BTC. The mixing service uses funds from other participants to make it look like a participant creates a transaction with another participant, which is not questionable on its own. Some mixing services allow a participant to either send the coins in one transaction or multiple to different addresses owned by the mixing service. At the same time, mixing services often split up transferring funds back coming from multiple addresses, over multiple transactions spanning over hours or days. This method increases the ambiguity of the made transactions. Examples of mixers include Bitcoin Fog, BitLaundry and Blockchain.info's "Shared Send" service [1] [8] [11].

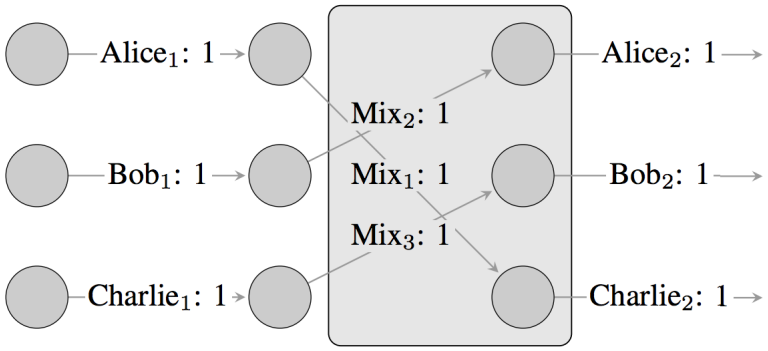

Figure 1: Hypothetical Bitcoin mixing service [26]

However, the usage of mixing services suffer from some serious drawbacks. It is a common occurrence that the service goes out of business, together with the Bitcoins. The mixing service may be victim of a hack, or was setup as a scam in the first place. While it is legally possible to sue the service, it can be difficult to trace the responsible party. Having a third-party also poses a risk for the anonymity of participants. Because the third-party learns all input and output addresses, this information can be used to link the addresses, and therefore identities, back together again. Mixing also usually requires a fee. This fee consists of a percentage calculated over the total sum of funds for making use of the service.

CoinJoin [23] is a mixing protocol which solves some of the previously mentioned problems. While the protocol uses still a third-party server to connect participants, all participants sign a transaction which includes transfer for funds to each participant. By using a single transaction, the third-party is not able to disappear with the funds anymore. However, the third-party still learns all addresses. With all the addresses available, the input addresses and output addresses can still be linked together again.

CoinShuffle [28] is a decentralized mixing protocol that proposes a method where no third-party server is required and participants can mix in an anonymous manner. The details of this protocol will be discussed in Section 5.

3 Research question

The CoinShuffle protocol aims for mixing anonymously. However, mixing still involves creating a transaction. This transaction should contain all addresses of the participants. Through analysis of the block chain, there could be a correlation visible between the input and output addresses. A correlation between addresses could indicate a CoinShuffle-transaction has taken place. A CoinShuffle transaction being able to be identified removes any plausible deniability of being involved in mixing coins, thus diminishing anonymity.

This results in the following research question:

- Can a CoinShuffle-transaction as such be detected in the block chain?

This raises the following sub questions:

- In which situations is it possible to detect the transaction, and what information can be derived from this?
- If a transaction can be detected, what can be done to improve the anonymity?

4 Related Work

Anonymity on the Bitcoin network is limited to the use of Bitcoin addresses as pseudonyms. Transactions made by the same address likely have the same owner. It has been demonstrated that multiple addresses can possibly be linked if a user sends back Bitcoin change to the same address, as is mentioned by Barber et al [3]. Meiklejohn et al. support this by applying a heuristic based on change addresses. This results in a cluster of addresses that are likely owned by a certain user [24].

De-anonymization is not limited to only clustering addresses together, but is extended by tracing an IP address that is linked to an address. The modular framework BitIodine is able to map Bitcoin addresses to actual IP addresses. This is done by merely using public sources such as requesting information from the block chain or scraping websites on the internet [32]. Mapping IP addresses can also be done by just connecting with a custom Bitcoin client that collects IP data from peers in the Bitcoin network, as is shown by Koshy, Koshy, and McDaniel [21].

In an attempt to improve on anonymity, several extensions have been made to the Bitcoin protocol. Zerocoin allows fully anonymous currency transac-

tions [25]. However, this extensoin was eventually not included in the original Bitcoin protocol and received some criticism because of the increased computation time. An improvement of this protocol was developed and renamed to Zerocash as a standalone alternate coin protocol [4]. A follow-up study introduced Pinocchio Coin, based on Zerocoin. Pinocchio coin features smaller proofs and faster verification [14].

Bitcoin Mixing Services make an attempt to provide a service on the current Bitcoin network to improve anonymity by unlinking the input and output addresses. Services such as Bitcoin Fog, BitLaundry and Blockchain.info's Shared Send functionality [1] [8] [11] are available, but have some drawbacks in terms of effectiveness and trust [26]. Mixcoin proposes a method where theft by mixes can be exposed [12].

CoinJoin [23] tries to solve the risk of losing coins by connecting participants that want to perform a mix by letting them sign for a single transaction. However, this still involves a third-party that can link the inputs and outputs.

The CoinShuffle authors provide an implementation of the protocol. However, the authors note that this implementation is only meant to evaluate the feasability of the protocol and does not include any security features. One other implementation is available consisting of a simulation tool, a server and a wallet [37] [36] [35]. However, there is no publicly available wallet software that implements the protocol for general use. One of the the Darkwallet's (a Bitcoin wallet) [15] developer mentioned implementing CoinShuffle in the wallet software. This implementation would include a tool independent of any wallet software to use CoinShuffle, but this feature is lacking at the time of writing [17] [16].

5 CoinShuffle Protocol

The CoinShuffle protocols aims to alleviate some of the problems the mixing services suffer from, while keeping compatible to the existing Bitcoin protocol.

No Third Party CoinShuffle does not require a third party in a sense mixing services are the intermediary. The protocol is decentralized where only the participants have to connect with each other, without the need for a third-party service. The anonymity of each participant is secured by not having a third-party service that learns the participant's identities and can link input and output addresses together.

Compatibility The protocol works on top of the existing Bitcoin network and does not require any modification of the protocol. Only the participants need to support the CoinShuffle protocol.

No Mixing Fee CoinShuffle is designed to have a single transaction. There is only a fee paid over this a single transaction, where the fee is divided

over the number of participants. There is no fee introduced for using the mixing protocol.

Small Overhead The CoinShuffle protocol uses messaging between participants to communicate with each other. Each message is signed with the private key of a participant. Signing adds a small overhead factor. Executing the protocol introduces a small time-based overhead due to the participants communicating.

Efficiency The CoinShuffle protocol introduces public key encryption to ensure the participants' anonymity. Because hashing and signing messages are already used in the Bitcoin protocol, only the added public key encryption requires extra computation. The public key encryption does not substantially increase computation, making this viable on computationally restricted hardware.

In addition to the goals described above, CoinShuffle aims to achieve the following security and privacy goals.

Unlinkability With a successful run of the CoinShuffle protocol, every participant's input addresses and output addresses should be unlinked.

Verifiability Every participant should be able to verify the transaction so a dishonest participant has no possibility of stealing coins from other participants without anyone noticing.

Robustness The protocol should be resilient in the case of one or more dishonest participants, given that the communication is stable.

The protocol is performed in three phases.

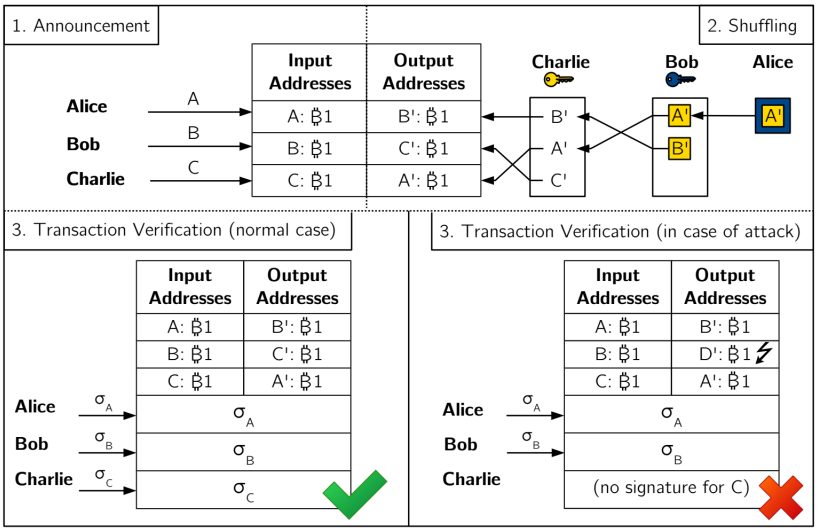

Figure 2: CoinShuffle Phases [28]

1. **Announcement** Every participant that wants to do a shuffle creates a new ephemeral encryption key pair. The public encryption keys are broadcast to every participant.

   Note: The CoinShuffle protocol does not indicate which concrete method of broadcasting or connecting with other participants is used. The paper mentions it can be done with the use of a bulletin board or a modified peer-to-peer protocol.

2. **Shuffling** Each participant uses their key pair to create a new output address. Alice, Bob and Charlie now have their output addresses (A', B' and C'). Each output address is now encrypted by all public keys, in a lexicographical way. Alice starts with encrypting her output address with Charlie's public encryption key, and then encrypts it again with Bob's. Alice now has her output address encrypted twice. She sends this to Bob.

   Bob decrypts Alice's output address with his private key. He adds his own output address, first to be encrypted with Charlie's public encryption key. Now Bob now has to ciphertexts: A' and B'. Bob shuffles the list of ciphertexts randomly and sends this to Charlie.

   Charlie receives the two ciphertexts and decrypts them, resulting in A' and B'. Since Charlie is the last participant, he does not need to encrypt his output address. He adds his own address and shuffles the addresses again. These addresses are then broadcast to the other participants.

3. **Transaction Verification** Each participant can now verify if their output address (A' for Alice, B' for Bob and C' for Charlie) is in the list. When

each address is present, a transaction is created. This transaction consists of each input address transferring the shuffle amount to the designated output address. This transaction is individually signed by each participant and broadcast. The transaction is then propagated to the Bitcoin network.

3. **Blame** (alternate phase) This phase is omitted if all participants behave according to the protocol. The blame phase is entered when some participant deviates from the protocol. Any participant can broadcast in case of deviation. There are a few cases this can occur.

   Not enough coins available The mixing transaction can't proceed if either a participant does not have sufficient funds or the funds are not available anymore when signing the final transaction. The Bitcoin network can be consulted if sufficient funds are available.

   Shuffling is not performed correctly To find out which participant did not behave according to protocol, all messages and ephemeral decryption keys are broadcast. This way, the communication can be replayed to single out the dishonest participant.

   Equivocate broadcasts Equivocating can happen when a participant communicates different public encryption keys to other participants. Since all public encryption keys are broadcast at the start, any differentiation in signed messages can be detected.

6 Setup

It was decided that the best method to investigate CoinShuffle transactions on the block chain was to create one. In order to create a CoinShuffle transaction, a test setup was configured. This test setup consists of the following parts:

- Bitcoin test network
- Bitcoin Core [5]
- insight Block chain Explorer [9]
- CoinShuffle Server
- CoinShuffle Sim

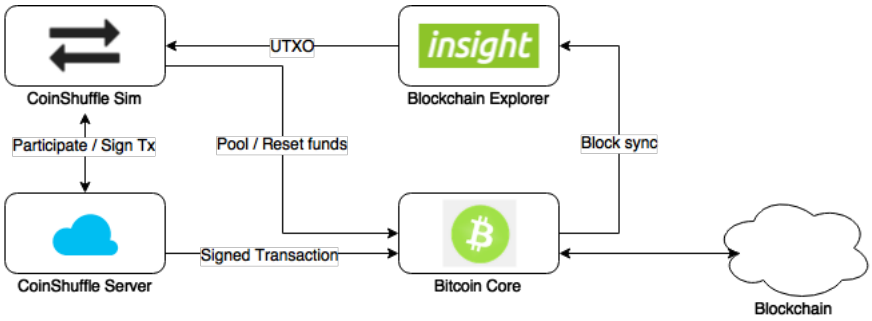

Figure 3: CoinShuffle Transaction Flow

Figure 3 shows the test setup layout with different flows between the individual parts.

Bitcoin test network The Bitcoin test network is a network run separately from the existing Bitcoin Live- or Testnetworks (Testnet3), which are publicly available. This network runs and behaves like a regular Bitcoin test network, with a few differences. There are no ties to the public existing networks and does not require a connection to them. This network identifies itself as a test network, and allows clients to connect with regression test mode enabled (regtest). Regtest allows miners to almost instantly mine blocks due to the lower difficulty, which is useful for testing purposes. Coins can be mined this way fairly easy, but have no value outside this network. The Bitcoin test network is indicated as the "Blockchain" in Figure 3. Where regular Bitcoin addresses are prefixed with a `1`, test network addresses are prefixed with either `m` or `n`.

Bitcoin Core Bitcoin Core is a Bitcoin client that can be ran either as a daemon using `bitcoind` or using the Qt-frontend with `bitcoin-qt`. When connected to a Bitcoin network, it downloads the complete block chain. This client can be configured to allow incoming RPC-connections as well as P2P-connections for requesting information and communicating changes from or to the rest of the network. Basic wallet functionality is available to send and receive coins from or to other addresses. This client is used in the setup to confirm transactions by acting as a miner, and propagating transactions to network.

insight Block chain Explorer The Block chain explorer insight provides insight in the different blocks and transactions made on the block chain. Insight constructs a separate database of blocks and transactions based on the information provided by Bitcoin Core. Upon start, it synchronizes the block chain information by connecting to the Bitcoin Core client using RPC calls. In this test setup insight provides information about the unspent transaction outputs (UTXO), the available funds to spend. The

graphical version is used in this setup, but the API-only version will also suffice as it is only the API functionality that is used.

CoinShuffle Server The CoinShuffle server acts as a bulletin board. CoinShuffle participants can "subscribe" to the server to join the shuffle. In the current implementation, when enough participants joined, the shuffle starts. As per the CoinShuffle protocol, every participant signs the transaction (Indicated by "Sign Tx" in Figure 3). In this implementation the server submits the final transaction to the Bitcoin network, where in the original protocol a participant would do this. Since the end goal is creating a CoinShuffle transaction, this deviation does not hinder this test.

CoinShuffle Sim The CoinShuffle Sim is a web page that can demo a shuffle with multiple participants (Figure 4). The Simulator controls ten wallets. Each wallet consists of input, change and output addresses. A *pool* wallet address is used that spreads the Bitcoins over the ten wallets so they have enough funds to be able to participate. This address is later used to collect the funds back. From the web page, a wallet can be instructed to join the shuffle with either 1 BTC or 0.1 BTC to shuffle. A request is done by invoking "Request N BTC" shuffle. After the CoinShuffle Server reaches a certain threshold of participants, the shuffle is set in motion. The participants follow the protocol by shuffling and signing the transaction. The server, in its turn then submits this transaction to the Bitcoin network.

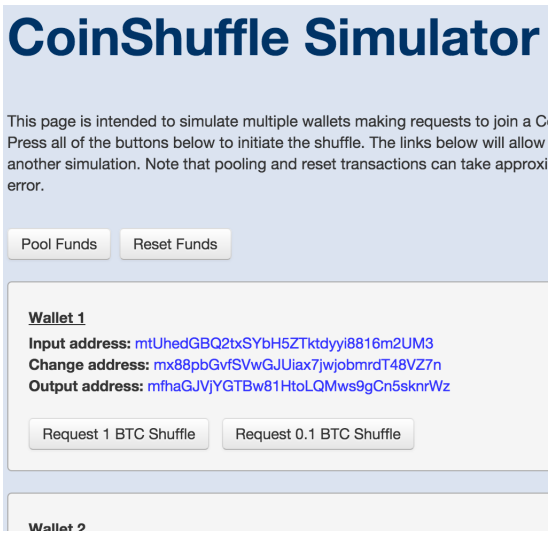

Figure 4: CoinShuffle Simulator Web Page

Sequence Each wallet is now able to participate in the shuffle. This shuffle consists of ten participants. With each wallet wanting to participate in the

shuffle by connecting to the server, insight is consulted if the wallet has enough funds available (UTXO). The wallet then participates. As said, the CoinShuffle Server starts after reaching a threshold. It requests all participants to sign the transaction (Labeled as "Sign Tx") and then sends the final transaction (Labeled as "Signed Transaction") to Bitcoin Core. Bitcoin Core propagates the transaction to the rest of the network. The Bitcoin Core client is manually instructed to generate a new block that includes this transaction, completing the sequence.

See **Appendix A** for configuration details of the test setup.

7 Experiment

The result of the CoinShuffle transaction can be seen in Figure 5.

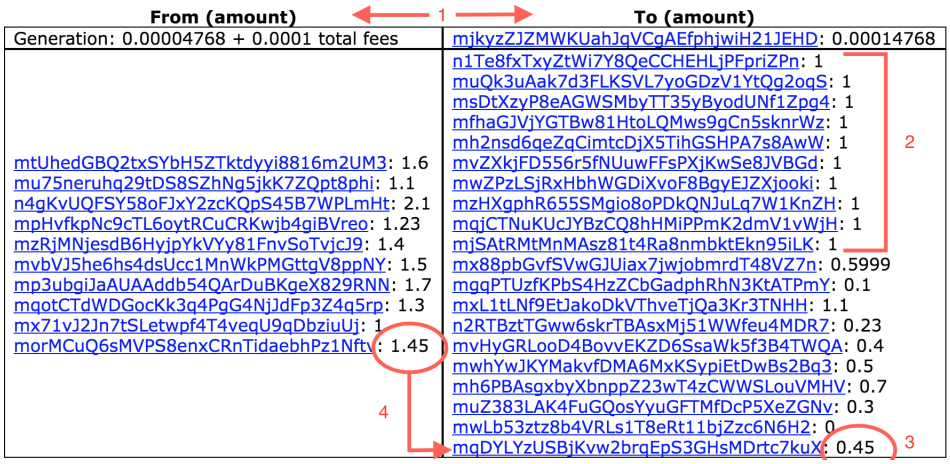

Figure 5: CoinShuffle transaction (ABE Explorer)

From the figure above one can recognize points that can possibly identify a CoinShuffle transaction. These recognition points are numbered in the figure.

1. **Number of outputs is twice the number of inputs**
   There are twice as many output addresses as the input addresses. From the figure, it can be seen that this transaction has ten input addresses on the left and 20 output addresses on the right.

2. **Mix amount can be deduced**
   In this figure there is a trend of a certain amount, which is equal to the amount of input addresses. There are ten output addresses with each 1 BTC, which is likely the amount of coins that is being mixed.

3. **Change addresses can be deduced**
   The ten remaining output addresses are likely the change addresses. If the mix amount is added to each change address, the resulting value is equal to, or close to the input amounts.

4. **Change addresses are linkable to the input addresses**
   The correlation between the change address and the input addresses as mention at the previous point are interesting, because this can link both together. For example, address `mqDYLYzUSBjKvw2brqEpS3GHsMDrtc7kuX` receives 0.45 BTC. When adding the mix amount, which is 1 BTC, this results in a total amount of 1.45 BTC. This amount is initially spent by the address `morMCuQ6sMVPS8enxCRnTidaebhPz1Nftv`.

Note: It might not be possible to determine every change address in every transaction. Singling out two addresses is less feasible when there are addresses that spend or receive the exact same amount. An example of this is shown in Section 8, Table 2. It is possible that in some cases a change amount of 0 BTC is not added to the transaction, also reducing the total amount of output addresses with one.

7.1 CoinShuffle using the CoinShuffle Wallet

Since the CoinShuffle Simulator is only a demonstration of creating a CoinShuffle transaction, there is no real control over the wallets used. In the real world, the shuffle is likely done by using a wallet that the user manages. The CoinShuffle Wallet is a fork of the existing Bitcoin wallet software CarbonWallet. CarbonWallet is a JavaScript-based Online Web Wallet. This application allows one to view the total wallet balance for each address and allows for making transactions. The CoinShuffle Wallet adds one feature: participating in a shuffle. Figure 6 shows the UI of the wallet.

Figure 6: CoinShuffle Web wallet

Participating in a shuffle is performed by first selecting an address with sufficient funds. The output address is specified as "Destination Address". As in the CoinShuffle Simulator, the current denominations are 1 BTC and 0.1 BTC. To initiate the shuffle, the "Verify and Request Shuffle" functionality is used. This connects to the CoinShuffle server, just as the CoinShuffle simulator would do for each of the wallets. For this experiment, four simulator wallets are used in combination with one CoinShuffle Wallet client with address `mtX7kF5BF1w4My77XSnoMP8AWb5QgHm8GK`. The destination address of the CoinShuffle mix amount will be `mmcXRBwrYVMpv9QjT7RdwtuWJiYJ3xk9fc`. After finishing the shuffle protocol, the transaction can be seen in the block chain, as shown in Figure 7.

From (amount)	To (amount)
Generation: 0.00004768 + 0.00011885 total fees	mjkyzZJZMWKUahJqVCgAEfphjwiH21JEHD: 0.00016653
mtX7kF5BF1w4My77XSnoMP8AWb5QgHm8GK: 2 mtUhedGBQ2txSYbH5ZTktdyyi8816m2UM3: 1.6 mu75neruhq29tDS8SZhNg5jkK7ZQpt8phi: 1.1 n4gKvUQFSY58oFJxY2zcKQpS45B7WPLmHt: 2.1 mpHvfkpNc9cTL6oytRCuCRKwjb4giBVreo: 1.23	mmcXRBwrYVMpv9QjT7RdwtuWJiYJ3xk9fc: 1 mqjCTNuKUcJYBzCQ8hHMiPPmK2dmV1vWjH: 1 msDtXzyP8eAGWSMbyTT35yByodUNf1Zpg4: 1 mvZXkjFD556r5fNUuwFFsPXjKwSe8JVBGd: 1 mfhaGJVjYGTBw81HtoLQMws9gCn5sknrWz: 1 mtX7kF5BF1w4My77XSnoMP8AWb5QgHm8GK: 0.9999 mx88pbGvfSVwGJUiax7jwjobmrdT48VZ7n: 0.6 mgqPTUzfKPbS4HzZCbGadphRhN3KtATPmY: 0.1 mxL1tLNf9EtJakoDkVThveTjQa3Kr3TNHH: 1.1 n2RTBztTGww6skrTBAsxMj51WWfeu4MDR7: 0.23
mz4kHFKqyPcSUuMTWsBCtffo1fiFwsK294: 0.01 mrYqpBUhJ6GLNkvdE2gK1BBvnpd2e9k8WU: 0.01003512	mw8Tv9hU8JmCJ9zdReeKmCxU7t7X8Xearc: 0.01001627 n2xs8UVE4V4G5USPWyMiJLMzkeqUcigvtv: 0.01

Figure 7: CoinShuffle Transaction made with CoinShuffle Wallet (ABE Explorer)

Note: Figure 7 shows two transactions, displayed in the two bottom rows. The first row displays the fee the miner receives. The focus will be the first transaction (second row) as the second is an effect of other tests performed on the Bitcoin test network. Since miners combine unconfirmed transactions, this CoinShuffle transaction ended up in this particular block. The miner that did the Proof-Of-Work receives the mined Bitcoins and fees for this block, seen in the first row with address `mjkyzZJZMWKUahJqVCgAEfphjwiH21JEHD`.

The recognition points mentioned previously are also visible in this transaction. In this case there are five input addresses together with ten total output addresses. The mix amount is deducible for the five addresses in the second column: 1 BTC. The five last output addresses are likely the change addresses.

The CoinShuffle Wallet was modified to work in the test setup environment. See **Appendix B** for details about the wallet software.

8 Block chain analysis

Together these results provide important insights on how to recognize a CoinShuffle transaction from a visual perspective using the recognition points. An automated approach was used to gain a detailed understanding about the usage of CoinShuffle on the Bitcoin network. First, the Bitcoin test network is traversed, followed by the live Bitcoin network. This approach consists of creating a script that implements two recognition points. Each transaction should have twice the number of outputs when compared to the number of input addresses. Furthermore, the mix amount should be deducible. The minimum amount of "participants" is set to five, as any lower would increase the risk of linking addresses together just by guessing. It may be that some CoinShuffle transactions are performed with a participant count as low as three, but these will not be included in the results. The mix amount is deduced by creating a list of each output amount and counting the occurrences. If the occurrences are equal to the amount of input addresses, it is marked as being the mix amount.

The block chain explorer insight was used as a source for the transaction information. To begin this process, a time span was defined for fetching blocks that were created during that time. Once all block hashes were retrieved all transactions hashes were extracted. Subsequently, all input and output addresses were requested from the block chain for each transaction. These input and output addresses were then matched against the recognition patterns. Matching was repeated for every individual transaction. The script outputs a log line in case of a match. Running the script on the Bitcoin test network yields two results, as seen in Figure 8.

Figure 8: CoinShuffle Transaction Recognition

Every match consists of three lines. Each line is prefixed with the matching transaction id indication (shortened to `TX id`) followed by the hash, all enclosed in square brackets. The first line indicates a possible CoinShuffle transaction has been found that matches the recognition points. The second line displays the input addresses (`Ins`) and output addresses (`Outs`). The third an final line consists of the possible mix amount. The value displayed is the mix amount in satoshi (100.000.000 satoshi is equal to 1 BTC) with the converted amount in parentheses. The two results in Figure 8 are not surprising, as two CoinShuffle transactions were created: using the simulator alone and in combination with the CoinShuffle Wallet.

In the next part that follows, the live Bitcoin network will be traversed to get an overview of CoinShuffle usage in the real world. All transactions were evaluated from April 2014 until June 2015. Criteria for selecting this period were as follows. A preliminary version of the CoinShuffle paper was posted on the Bitcoin forum bitcointalk.org in early April by one of the authors, Tim Ruffing [33]. This point in time would be a start where the protocol is made known to the public, and the Bitcoin community was able to implement the protocol specifics. Usage of the protocol could see a potential increase because a prototype was made available later by the CoinShuffle authors. There could be another increase after July 13, as the first commit of the CoinShuffle implementation by Bryan Vu appeared on July 13 on GitHub. Usage before April 2014 is possible, as the authors may have tested their implementation on the live Bitcoin network, besides testing on a Bitcoin test network. The end date of the sample set could go as far into the future as possible. It was not possible to traverse further than late June, as this was the current time of writing. Regardless of having an incomplete view of June, the sample set should provide a decent overview of the findings.

The traversal of the block chain was ran in multiple segments. Data collection was done in segments of one ore a couple months. Segmenting the data collection had no impact on the end result, and was only done due to limitations of the hardware and script. All output, as seen in Figure 8, was captured into plain text files for further processing. The script was modified for the larger data set to include the local time and Unix time stamp for easier analysis based on time, as will be seen in future listings. Each segment logged the total amount of block and transaction hashes, which were added up.

A total of 69.197 block hashes consisting of 37.597.942 transactions were checked

for possible CoinShuffle transactions. The total amount of possible CoinShuffle transactions results to 360 ($\approx 0.00096\%$ of the total amount of transactions). All matches are considered *possible* CoinShuffle transactions, as there is a possibility other transactions have the same pattern. These results therefore need to be interpreted with caution.

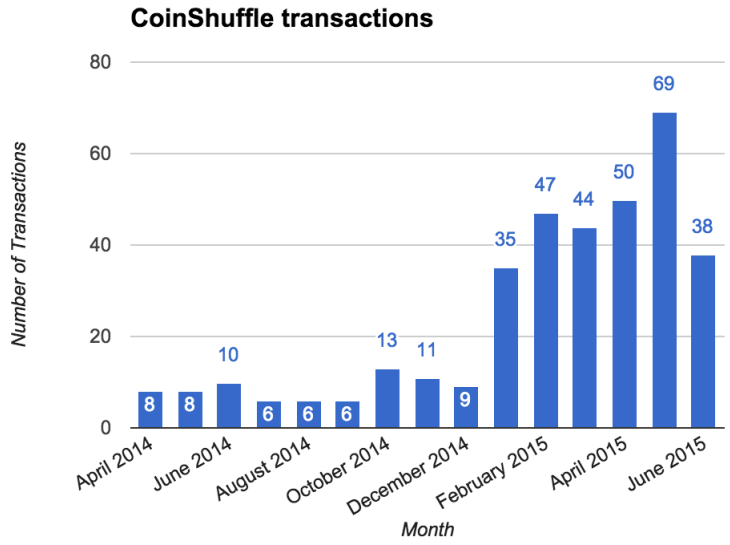

Figure 9: CoinShuffle Transactions per month

The results of the block chain traversal are shown in Figure 9, with the total amount of transactions per month. What is interesting is that April 2014 already has eight transactions. During the year the amount fluctuates a bit, with an all-time low of six transactions three times in a row for August, September and October. A peak occurs later in the year with 13 transactions in October of 2014. The amount of transactions is greatly increased with the start of 2015, where January already counts 35 transactions. There is is no buildup visible from the previous year. The cause of this influx is unknown. The amount of transactions in 2015 stay at a fairly high point with a peak of 69 in May. June appears to have a sudden drop May, but is not substantial. It is possible the total amount of transactions will eventually come out higher, as June was not completely traversed.

With all the transactions known, we will have a closer look at one of the transactions to determine if this could validate as a CoinShuffle transaction. The transaction that will be looked at will be `aa63353de65f76e2799d245e2d0448a430ea3e7058bcc88352f81ae3b79de16f`, which is shown in Table 1

Input amounts	Output amounts	Change + Mix
0.0156	0.0155	
0.0157	0.0155	
0.0157	0.0155	
0.0157	0.0155	
0.0157	0.0155	
0.0157	0.0155	
0.0157	0.0155	
0.01653	0.0155	
	0.0001	0.0156
	0.0001	0.0156
	0.0001	0.0156
	0.0001	0.0156
	0.0001	0.0156
	0.0001	0.0156
	0.0005	0.016
	0.00093	0.01643
Total: 0.12633	0.12603	0.12603

Table 1: Transaction input and output (addresses removed for brevity, amounts in BTC)

This transaction was detected with the following characteristics:

```
TX aa63353de65f76e2799d245e2d0448a430ea3e7058bcc88352f81ae3b79de16f

Possible CoinShuffle transaction
[2015-05-11 08:07:45 (TS: 1431324465)]
Ins: 8 Outs: 16
8 occurrences of 1550000 satoshi (0.0155 BTC)
```

Listing 1: Transaction detection (output modified for brevity)

Table 1 shows the breakdown of the transaction. The first two columns show the input and output amounts similar to Figure 5, but does not display the from and to addresses. All amounts are grouped to improve readability, where these are scattered in the actual transaction. According to the script output, the mix amount is likely 0.0155 BTC as this has an equal count of occurrences when compared to the total count of input addresses. As seen in the manually created CoinShuffle transaction in Section 7, the remaining values are likely the change.

Due to the nature of Bitcoin transactions the change addresses are not always evident. Change addresses are just another output address, which makes it difficult to link to a particular input address when a transaction has more than one input address. However, because the change amounts are known, adding the change and mix amounts should result in a value that is close to the input

amounts. The third column shows the result of this calculation. All the amounts are close to the input amounts, but not exactly the same. However, when all added up, logically the total amount is equal to the total output amount: 0.12603 BTC. The fee of this transaction is 0.0003 BTC, which is the difference between the total input and output amount. Each participant pays a certain amount of the fee and therefore does not receive the exact change back. A thing that stands out are the input amounts. Almost all of the addresses exactly transfer 0.0157 BTC. While there is a low probability wallets have exactly such amount available, there are eight other addresses that have the same or a similar amount in this transaction. However, it is possible these funds were deliberately transferred from other wallets first.

One unanticipated finding was that six of the input amounts are identical: 0.0157 BTC. This makes it impossible to correlate all change addresses to the input addresses. While there are also six change amounts of 0.001 BTC, it can not be determined which amounts, and therefore addresses, belong together. Interestingly, the mix amount is not one of the standard denominations provided by the known CoinShuffle implementations.

Amongst the CoinShuffle transactions, a number of them have different characteristics. These characteristics were initially not apparent before examining individual transactions on a larger scale. In particular, the transaction with id `851e5dc9cbed65838d859e02396798fa258968361243e0f6d9bf5d94f3068ff8` is chosen as example.

Input amounts	**Output amounts**
0.0127[†]	0.04[*]
0.0127[†]	0.04[*]
0.0127[†]	0.04[*]
0.2559[†]	0.04[*]
0.1279[†]	0.04[*]
	0.0417[†]
	0.045[*]
	0.045[*]
	0.045[*]
	0.045[*]

[†] Address 184PZyioFrLLC3xFians2vnopaaoUGTHge
[*] Address 1LuckyR1fFHEsXYyx5QK4UFzv3PEAepPMK

Table 2: Transaction input and output Example #2

```
TX 851e5dc9cbed65838d859e02396798fa258968361243e0f6d9bf5d94f3068ff8

Possible CoinShuffle transaction
[2015-05-01 14:59:57 (TS: 1430485197)]
Ins: 5 Outs: 10
5 occurrences of 4000000 satoshi (0.04 BTC)
```

Listing 2: Transaction detection #2 (output modified for brevity)

The example in Table 2 shows five input amounts and ten output amounts. This matches the recognition pattern where the number of outputs is twice the number of inputs. Listing 2 indicates 0.04 BTC as the mix amount. This amount and occurrences match the recognition point of the mix amount being deducible, as this also consists of five output amounts. However, when looking at the input amounts, only two of the input amounts can satisfy the 0.04 BTC mix amount. This suggests that the transaction is unlikely a CoinShuffle transaction. The protocol does not succeed when not all participants have sufficient funds. In addition to this, all the "participants" transfer funds from just one address. In general, there is little reason to perform the CoinShuffle protocol with just one address other than for testing purposes. Performing CoinShuffle with one address defeats the purpose of unlinking output addresses from input addresses, because the input address is already known.

When looking at the table, there are four output amounts that have a single destination address: `1LuckyR1fFHEsXYyx5QK4UFzv3PEAepPMK`. If this particular address is searched for online, one finds this address is used in combination with the website LuckyBit [22]. LuckyBit is an online gambling website where one can gamble Bitcoins. The LuckyBit address itself is an example of a *vanity address*, an address which contains human-readable messages. These addresses require one to generate a lot of private keys with matching addresses to find a certain pattern. The remaining output amount of 0.0417 BTC has the address of the sender, it being likely the change address that is unchanged. It can be suggested that someone wanted to make a deposit to play on the online gamble website.

Having a transaction that does not have sufficient funds for each "participant" to join in a shuffle means there are likely more transactions with the same pattern. To filter out false positives, one extra rule can be added: all input amounts should have sufficient funds. Filtering out these transactions decreases the CoinShuffle transactions from 360 to 298 hits. Figure 10 shows the transactions per month updated to the new situation where all transaction inputs have sufficient funds.

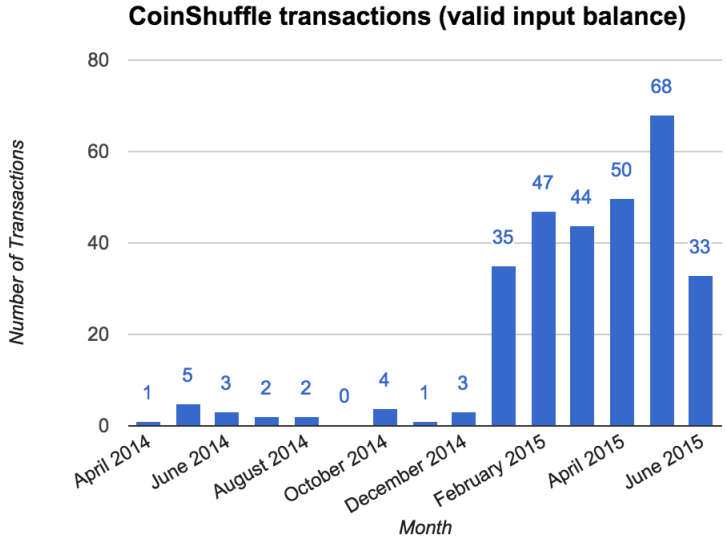

Figure 10: CoinShuffle Transactions (with valid input balance)

The year 2014 loses some transactions to a point where there are only a couple in a month, with a peak of five in May. September does not have any possible transactions at all. 2015 stays roughly the same over the whole period. It is not until May 2015 where the possible transactions is lowered from 69 to 68. June drops with five from 38 to 33. One thing that is still noticeable is the difference between the years 2014 and 2015, where 2014 even had less transactions and 2015 stays nearly the same.

While these transactions can be perceived as CoinShuffle transactions, they do not necessarily need to be so. It is not unlikely that between all the 37 million transactions there are a few that exactly match the CoinShuffle recognition points, but aren't actually such transactions. A possible explanation for this might be that some of the transactions are just merges from older wallet addresses. One transaction with multiple inputs could just merge the funds and redistribute these funds to other addresses in equal amounts, as also seen with the LuckyBit transaction in Table 2. Another possible explanation for this is the use of CoinJoin [23]. Because CoinJoin works in a similar way in comparison to CoinShuffle, some of these transactions could actually be CoinJoin transactions. CoinJoin is a mixing service that also has multiple participants that sign a single transaction. A CoinJoin transaction also requires each input amount to have sufficient funds and output amounts to be equal to the mix amount. With this information, it can be suggested that the possible CoinShuffle transactions are only a subset of the CoinJoin transactions. This also works the other way around, some of the transactions are CoinJoin transactions which are a subset of the CoinShuffle transactions. The CoinJoin mixing method was posted on a

Bitcoin forum on August 22, 2013. A search for possible CoinShuffle transactions in December 2013 resulted in at least one transaction to be present (see Listing 3). It can be suggested that users were actively trying out CoinJoin already in that period, or this could indicate early tests of CoinShuffle were being performed in the live Bitcoin network. Further data collection is required to determine if transactions with a CoinShuffle-pattern occur on a regular basis.

```
TX 989d42d6657a113470fd2d605e5457e763af99d14ff31f99cd8b92d8036ad242

Possible CoinShuffle transaction
[2013-12-07 15:45:22 (TS: 1386427522)]
Ins: 10 Outs: 20
10 occurrences of 1000000 satoshi (0.01 BTC)
```

Listing 3: Possible CoinShuffle transaction December 2013 (output modified for brevity)

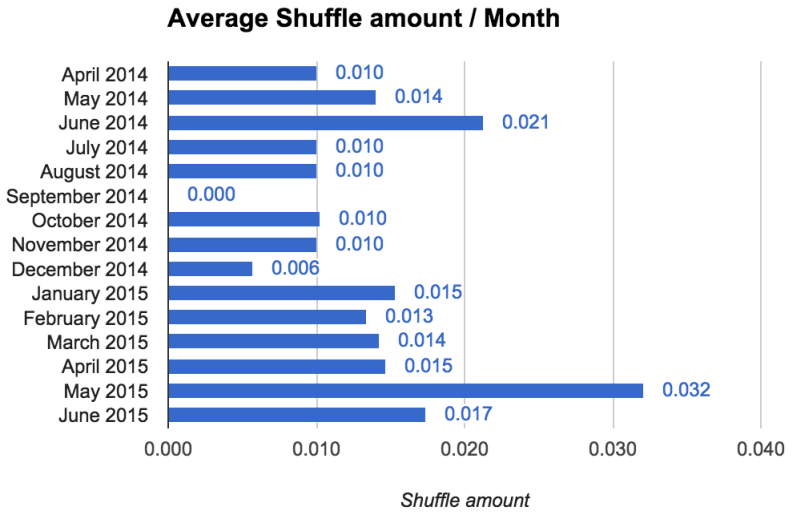

Figure 11: Average CoinShuffle amount used

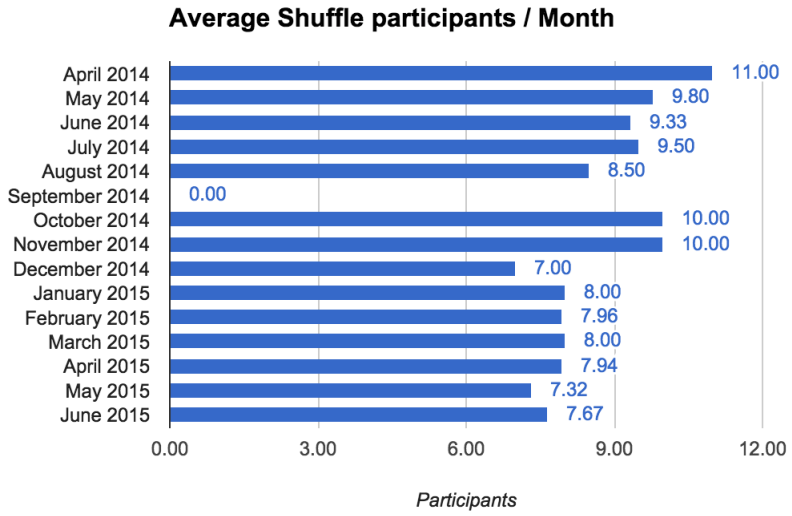

Figure 12: Average CoinShuffle participants

Figure 11 and 12 show the average shuffle amount used in a transaction per month and the average amount of participants per month. The most interesting finding was that the shuffle amount stays at a low level. By far the most popular amount is 0.01 BTC. This amount is consistent with the data in Figure 11, as most averages revolve around this amount. What is interesting to note is that the average amount of participants lies close to nine or ten participants in 2014, and is closer to eight in 2015. The results in 2015 are likely due a great number of transactions that consist of eight input addresses and 16 output addresses, with a few that have a lower number of participants.

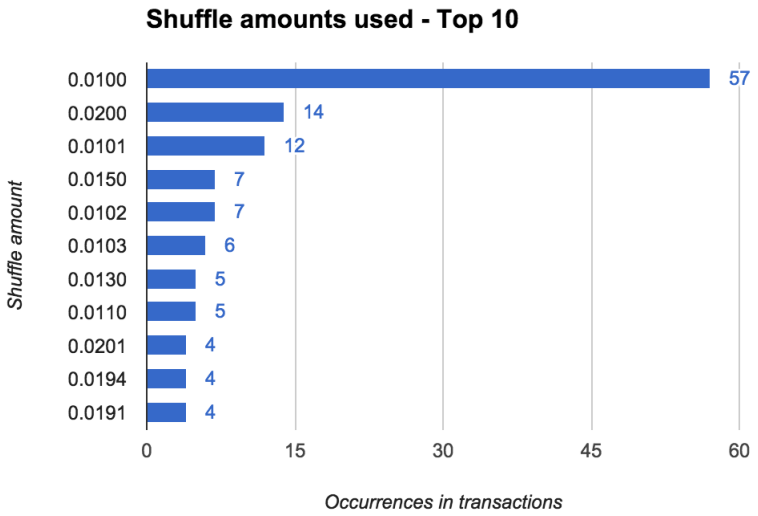

Figure 13: Shuffle Amount used - Top 10

The results in Figure 13 show that 0.01 BTC is the most popular mix amount for use in a shuffle. An amount of 0.02 BTC takes second place. Surprisingly, all other shuffle amounts are only a little higher. The amounts are of a very small denomination, rather than having 1 BTC or higher.

These results further support the idea that CoinShuffle transactions are rare and not very widely used on the live Bitcoin network. Following the hypothesis that all transactions are indeed CoinShuffle transactions, the protocol usage is not significant. This might be due the fact there are no known full implementations of the protocol. The only implementations available are a prototype from the authors, the simulator and the defunct CoinShuffle Wallet. The most shuffled amount is 0.01 BTC (€ 2.62 at the time of writing) which is not a substantial amount. It seems possible this is due testing, but is not actually used for real purposes. The CoinShuffle Simulator and CoinShuffle Wallet only feature the option to shuffle in 0.1 BTC and 1 BTC denominations, which could implicate transactions with other denominations have a different source. It may be the case the software was modified to use an arbitrary denomination, or the transactions stem from a completely different purpose.

See **Appendix C** for details about the analysis script.

9 Possible Improvements

Mixing services apply multiple methods to make transactions more obscure. One of them is splitting the total mix amount into multiple smaller amounts.

The total received funds will equal the funds sent, sometimes to multiple output addresses. This is usually paired with multiple transactions over multiple hours or days. Splitting a CoinShuffle transaction is not possible because it is designed to be a single transaction signed by every participant. Splitting return of the funds into multiple transactions would mean the exact same participants to be available in the future to sign another transaction. This would, however, be possible if certain transactions could be delayed until a certain point in time. Every participant would sign multiple transactions upon executing the CoinShuffle protocol. Signing a delayed transaction could pose a problem with double spending funds, because the transaction is only confirmed at a later stage. The funds are only later subtracted, and before freely spendable. There is a way to create a delayed transaction, but is not an official feature [20]. Forging such transaction involves manually modifying a transaction so that it is not executed until a certain block height. Because this is not officially supported, some clients deny propagating this to the Bitcoin network. Delaying the transaction also means that a transaction floats in limbo, waiting to be processed. With the transaction being in limbo, it could introduce some extra load on the miners as they are just waiting to include the transaction in a block. Multiple transactions do, however, introduce more fees.

While the CoinShuffle transaction visibly has connections between the input and change addresses, this might be more ambiguous when shuffling with 100 participants. However, a shuffle with a low number of participants might still benefit from increased obscurity. Mixing services allow users to provide multiple output addresses. Every participant would, in this case, be able to provide multiple output addresses. The protocol would only have to make sure to divide the funds into smaller denominations, eventually adding up to the mix amount. While this does not further obfuscate it for the participants, it would be harder to match change addresses due to the larger amount of addresses in the transaction. To evade recognition by just correlation the count of output and input addresses, several participants could have a varying amount of addresses. The participant creating the final transaction could also try to match other transaction amounts, resulting in a change amount that matches multiple addresses. An example can be seen in Table 1.

10 Conclusion

Returning to the question posed at the beginning of this thesis, it is now possible to state CoinShuffle transactions are visible on the block chain. Through a test setup, CoinShuffle transactions were created on a test Bitcoin network. These CoinShuffle transactions can be identified by several recognition points: number of outputs is twice the number of inputs, mix amount is deducible, change addresses are deducible and change addresses are in some cases linkable to the sending addresses. With the knowledge of the recognition points, the block

chain can be traversed for matching transactions. The live Bitcoin network was traversed in the period April 2014 until June 2015 where close to 300 possible CoinShuffle transactions were found. The results of this investigation shows that the CoinShuffle protocol is rarely used. It was not possible to assess if said transactions were actual CoinShuffle transactions, as there could be multiple explanations for these transactions to match a pattern. Within the data set, there are multiple transactions that indicate not being a CoinShuffle transaction. This is shown by a transaction having one and the same input address while transferring funds to a particular output address. Another possibility is that a part of the CoinShuffle transactions are a subset of CoinJoin transactions, or vice versa. Both protocols possibly create transactions that are similar to each other. More research is required to determine the origin of these transactions.

The CoinShuffle protocol could benefit from some protocol changes inspired by techniques applied by mixing services. On the contrary to a participant having a single input or output address, one could have multiple. Having multiple addresses would make detection harder as it becomes more ambiguous which mix amount was used. The participants could also try to match input funds to sever the link between input and change addresses. Whilst this thesis did not implement these changes, future work can be done to test the effectiveness.

11 Future work

The CoinShuffle protocol can be modified in a way where it incorporates certain mixer techniques. Implementing a feature that allows participants to have multiple addresses would have to point out if this technique makes detection harder.

The block chain can be traversed further back into 2014 or previous years to get more insight about the transactions that classify as CoinShuffle transactions. December 2013 showed at least one transaction that matched the CoinShuffle recognition patterns. It is possible the found CoinShuffle transactions are only a subset of CoinJoin transactions, or vice versa. There is no difference between the two as both are transactions with multiple input and output addresses, which happen to match the recognition points. Because CoinJoin was made public in August 2013, any early searches in 2013 could give more clarity. It can be suggested that transactions that still match the pattern in that period of time are just an effect of probability, amongst thousands of transactions created per day. However, it is possible this probability decreases further back in time. The number of transactions have increased in 2015 when compared to 2014. The probability is likely lower as 2013 has even less transactions per day.

The CoinShuffle Wallet was modified to work as a standalone wallet that is able to participate in the CoinShuffle protocol. However, this software still requires some development before public adoption can take place. Bryan Vu, the

author of the CoinShuffle Simulator, Server and Wallet already lists some future work in his presentation about the CoinShuffle demo [34]. The CoinShuffle Wallet currently is just a web page, which can be packaged into a Chrome App that causes the wallet to have a more standalone application feel. Currently, the blame phase is not implemented. CoinShuffle communication is preferable using peer-to-peer, where the CoinShuffle Server now acts as an intermediary. A (modified) peer-to-peer model would have to be implemented to make the protocol fully decentralized. The CoinShuffle Wallet is now dependent on the block chain explorer insight, but this can possibly be solved by using only the Bitcoin daemon `bitcoind`. Another point of attention is adding the ability to specify a change address, as the change now gets returned to the sending address, which diminishes the anonymity.

References

[1] Bitcoin fog. `http://bitcoinfog.com/`.

[2] Mt.gox thinks it's the fed. freezes acc based on "tainted" coins. (unlocked now), 2012. `https://bitcointalk.org/index.php?topic=73385.0` (visited 2015-06-24).

[3] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better—how to make bitcoin a better currency. In *Financial cryptography and data security*, pages 399–414. Springer, 2012.

[4] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[5] bitcoin. Download bitcoin core, 2015. `https://bitcoin.org/en/download`.

[6] bitcoin abe. Github: bitcoin-abe, 2015. `https://github.com/bitcoin-abe/bitcoin-abe`.

[7] bitcore, 2014. `https://github.com/bitpay/insight-api/issues/115`.

[8] BitLaundry. For all your bitcoin washing needs! `http://app.bitlaundry.com/`.

[9] bitpay. A bitcoin blockchain api for web wallets, 2015. `https://github.com/bitpay/insight`.

[10] Blockchain.info. 24 hour weighted price from the largest exchanges. `https://blockchain.info/q/24hrprice?cors=true`.

[11] Blockchain.info. Blockchain.info - bitcoin wallet - features. `https://blockchain.info/wallet/features`.

[12] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security*, pages 486–504. Springer, 2014.

[13] Vitalik Buterin. Mt.gox: What the largest exchange is doing about the linode theft and the implications, 2012. https://bitcoinmagazine.com/1323/mtgox-the-bitcoin-police-what-the-largest-exchange-is-doing-about-the-linode-theft-and (visited 2015-06-24).

[14] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*, pages 27–30. ACM, 2013.

[15] Darkwallet, 2015. https://darkwallet.is/.

[16] Darkwallet. Alpha 8, 2015. https://github.com/darkwallet/darkwallet/releases/tag/0.8.0.

[17] genjix. Forum post, 2015. http://redd.it/2ijsw1 (visited 2015-06-24).

[18] helloblock.io. Hello, block! — bitcoin api. https://helloblock.io/ (visited 2015-06-26).

[19] helloblock.io. Hello, block! — bitcoin api. https://helloblock.io/docs/ref (visited 2015-06-26).

[20] jekyll. Creating a delayed transaction, 2015. http://blog.coincreator.net/jekyll/update/2015/03/07/delayed_transaction.html.

[21] Philip Koshy, Diana Koshy, and Patrick McDaniel. *An analysis of anonymity in bitcoin using p2p network traffic*. Springer, 2014.

[22] LuckyBit. Luckybit - putting suspense back into bitcoin gambling, 2015. http://luckyb.it/ (visited 2015-07-09).

[23] G. Maxwell. Coinjoin: Bitcoin privacy for the real world, 2013. https://bitcointalk.org/index.php?topic=279249.0.

[24] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[25] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[26] Michael Moser, Rainer Bohme, and Dominic Breuker. An inquiry into money laundering tools in the bitcoin ecosystem. In *eCrime Researchers Summit (eCRS), 2013*, pages 1–14. IEEE, 2013.

[27] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.

[28] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *Computer Security-ESORICS 2014*, pages 345–364. Springer, 2014.

[29] Jan-Willem Selij. Github: coinshuffle-detect, 2015. `https://github.com/jan-willems/coinshuffle-detect`.

[30] Jan-Willem Selij. Github: coinshuffle-wallet, 2015. `https://github.com/jan-willems/coinshuffle-wallet`.

[31] Jan-Willem Selij. insight regtest genesis block, 2015. `https://gist.github.com/jan-willems/626c9a663387da8292c9`.

[32] Michele Spagnuolo, Federico Maggi, and Stefano Zanero. Bitiodine: Extracting intelligence from the bitcoin network. In *Financial Cryptography and Data Security*, pages 457–468. Springer, 2014.

[33] TimRuffing. Coinshuffle: Practical decentralized coin mixing for bitcoin, 2014. `https://bitcointalk.org/index.php?topic=567625.0` (visited 2015-07-08).

[34] Bryan Vu. Coinshuffle demo, 2014. `https://docs.google.com/presentation/d/19VEfEi6nZjtQtLVniDcPttGfr9xgudRgNIdhDVUKZWE/`.

[35] Bryan Vu. Coinshuffle demo wallet - forked from sf bitcoin devs hd wallet, 2014. `https://github.com/bryanvu/coinshuffle-wallet`.

[36] Bryan Vu. coinshuffle-server, 2014. `https://github.com/bryanvu/coinshuffle-server`.

[37] Bryan Vu. coinshuffle-sim, 2014. `https://github.com/bryanvu/coinshuffle-sim`.

[38] webbtc. Webbtc dumps, 2015. `http://dumps.webbtc.com/bitcoin/`.

A Practical Test Setup Configuration

A.1 Genesis block

Each Bitcoin network has a first block (block #0), called the genesis block. This block has a certain structure. Many clients have this block hard coded verify this is the same block on the current network operating on. The block chain explorer insight also performs this verification before synchronizing its databases. When used with a custom network, it will raise an error about a bad genesis block:

```
error: Bad genesis block. Network mismatch between Insight and
    bitcoind? Insight is configured for:testnet
```

Listing 4: Insight bad genesis block error

To make this work with a customer genesis block, the original genesis block has to be modified. Insight, as well as CoinShuffle Server and CoinShuffle sim make use of the JavaScript Bitcoin library `bitcore`. This library stores the genesis block in the file `networks.js` in the root of the module directory [7]. The genesis block hash of the current Bitcoin network can be requested using `getblockhash 0` and the full block details with `getblock <hash>`. Requesting information about blocks can be done from any Bitcoin client that already has the block chain downloaded.

The following values need to be modified:

- `magic`. The magic value, if used with a client running in regression test mode (parameter: `-regtest`). When in regression test mode, the official Bitcoin client only accepts P2P-connections with the magic value `0xFABFB5DA`, which was formerly used for *testnet*.
- `genesisBlock.hash`. The hash has to be changed, in reverse. As mentioned in the GitHub issue, this can be done with "`echo -n '<hash>' | dd conv=swab | rev`".
- `time`. The time stamp, if it differs.
- `nonce`. The nonce from the block info
- `bits`. The bits, in decimal.

Note: An older version of the `bitcore` library compatible with the implementation was used in this setup. The library may have changed the configuration in the meantime. For example, the P2P-communication is now split into a separate project `bitcore-p2p`. It may be possible to add another network without replacing an existing one.

The genesis block used in this setup is available on GitHub [31].

A.2 Running the simulator

The simulator runs on port 4000 out of the box. Running the simulator on this port does not enable all features. The application uses `socket.io` which tries to connect to localhost without having the same port configured, failing to connect The workaround used is to start the application on port 80. However, this does require privileges to do so. The environment variable `PORT` is used by Node.js to override the default.

```
PORT=80 npm start
```

Listing 5: Run CoinShuffle Sim on port 80

A.3 Installing insight

Installing insight's dependencies might fail on some systems. Insight requires `node-gyp`, which compiles native add-on modules for Node.js. This tool requires python version 2.7, as versions in the 3.x-branch will not work. When both versions are installed on a system, `node` can be configured to use python version 2.7.

```
npm config set python /usr/bin/python2.7
```

Listing 6: Configure python 2.7 path for npm

The tool `node-gyp` also fails when some modules require either the binary `node` or `nodejs` to be available. Failure of both being present can be recognized by the large amount of `ENOENT` errors produced by `node-gyp`. The missing file differs per distribution. If either one of them is not available, a symlink can be created pointing to the other binary.

```
ln -s /usr/bin/nodejs /usr/bin/node
```

Listing 7: Creating symlink to node(js) binary

The `nodejs-legacy` package on Debian and Ubuntu systems can also resolve this problem. `npm install` can then be continued as normal.

A.3.1 Running insight

The environment variable `INSIGHT_FORCE_RPC_SYNC=1` is used because insight's file synchronization does not work with `bitcoind` v0.10. Synchronizing using RPC is slower than the regular sync using Bitcoin core's database files.

```
INSIGHT_NETWORK=testnet BITCOIND_USER=<user> BITCOIND_PASS=<
    password> INSIGHT_FORCE_RPC_SYNC=1 npm start
```

Listing 8: Running insight with environment variables

Insight's web interface displays an error about not being able to connect to the Bitcoin daemon anymore after a while, whereas it was actually able to connect in the first place. No problems were encountered because of this, only live synchronization was not functioning.

A.4 CoinShuffle Sim Pool address

The CoinShuffle Simulator has a dedicated pool address:
`n2Zmh572s5wqFeDWy2V6kBndVGfaEr8CHM`. This pool address is used to transfer funds from to the ten wallets in the simulator to provide them with sufficient funds. This address is also used to transfer the funds from the change addresses back to the pool address, so the simulator can be run again. When the "Reset Funds" functionality is used, some wallets will receive more than 1BTC. This requires the pool address to have more than 10BTC (the total number of wallets * 1BTC). During testing with the simulator, an amount of 20BTC was sufficient.

B CoinShuffle Wallet

The CoinShuffle Wallet consists of an web application that can be run locally. The dashboard page of the application showed statistics which included an 24-hour average of the current Bitcoin value, the total wallet balance in Bitcoin and a 24-hour Dollar balance. These statistics were fetched from Blockchain.info [10]. Blockchain.info is a website which functions as a block chain explorer and provides an API for requesting block chain-related information. One of the features include the conversion to other currencies. The wallet value and balance for individual addresses was requested using an API available from helloblock.io [18]. Helloblock.io provided an API where information about the block chain such as blocks, transactions, and wallets could be retrieved [19]. Helloblock.io ceased providing functionality on April 1st 2015, as is posted on the main page. This meant the CoinShuffle Wallet was not able to function without the helloblock.io API available.

Because the wallet had to function in an isolated Bitcoin network, the data retrieval from Blockchain.info functionality was removed. The default implementation also requested this information every ten seconds, where Blockchain.info eventually replied that the request rate was too high through the API. Since helloblock.io did not provide its API service anymore, retrieval of wallet address balances was not possible anymore. The block chain exporer insight provides an API with similar functionality to helloblock.io. While the data structures are not one-on-one interchangeable, they were close enough. The helloblock.io functions were rewritten into functions that made use of insight, and also the callees were modified to make use of the new interfaces. The dashboard was modified to retrieve the current block height and mining difficulty.

The CoinShuffle Wallet was modified to work in the Bitcoin test network and be able to participate in the CoinShuffle protocol. However, the application was not extensively tested and still might contain some bugs. Signing a transaction might fail on occasion. This version does not include the ability to specify a change address.

Code is available on GitHub [30].

C Block chain analysis script

The block chain analysis script (or CoinShuffle transaction detection script) is CLI-based and was built on top of the block chain explorer insight. The API-only version `insight-api` (v0.2.13) was used as foundation. Because this version is an API on its own, it provides all the necessary API endpoints to retrieve information required for blocks and transactions. Insight is written in JavaScript and runs using Node.js, which is an event-driven I/O server-side JavaScript environment based on Google's V8 JavaScript engine. Node.js allows for writing JavaScript applications on the server, whereas JavaScript is usually executed in web browsers.

Insight uses the package `express` to provide the API endpoints. To circumvent the possible overhead HTTP calls might introduce, the API controller functions were directly called. However, because these functions actually expect a HTTP request coming in, several HTTP-related functionality or values were not available. While it was possible to create a shim that mimics HTTP behavior in certain cases, this turned out to be problematic and was eventually not used.

The API controller functions made use of different libraries that can query the database. By looking at the function interfaces and API endpoints, it could be determined which functions could retrieve the information required. The block database `BlockDb` and transaction database `TransactionDb` could fulfill this purpose.

The script was written with individual functions that were later tied together.

1. **Retrieve blocks from a certain time span** To begin the process, a time span was defined. The time span consists of a start date in combination with a time period that is repeated. The block hashes from this time span were retrieved.
2. **Retrieve all transaction hashes** With all block hashes, all transaction hashes from the blocks were retrieved.
3. **Retrieve transaction information** Once the transaction hashes were known, the transaction information was retrieved from the block chain. The transaction information included two important things: the input addresses and output addresses, together with the amount.

4. **Check for CoinShuffle recognition points** All individual transactions were passed in a function that checked for CoinShuffle recognition points. In case of a possible match, this is logged to the console.

The script is available on GitHub [29].

The block database featured an undocumented function that could retrieve block hashes from the database by providing a certain time span. These block hashes were then used to retrieve all the transaction hashes the block consisted of. Because these hashes on their own did not provide any information about input and output addresses, the information from each transaction hash was requested. However, the transaction database makes an RPC call to `bitcoind`. This requires the bitcoin daemon to be running to retrieve the transaction information. The RPC is done by requesting `getrawtransaction` for the transaction, which returns the hexadecimal-encoded data for the transaction id. Each call for transaction information also resulted in a separate connection to the bitcoin daemon. While this worked for the Bitcoin test network, this resulted in too many simultaneously connections. The bitcoin daemon rejected any connections above a certain threshold. The exact threshold is not known, but during testing the ceiling appeared to be near 3.000 connections. To initially limit the amount of connections, the package `limiter` was used. This package allowed a configurable way of limiting a certain amount of connections per a certain time span. The limit was set for 1.000 connections per 65 seconds. During development the timeout appeared to be 60 seconds, so a grace period was included to wait a bit longer to create room for new connections. Any attempts to modify the RPC library that was called by the transaction database to close connections at a faster rate were inconclusive. While the connection count stayed at a level that was below the threshold, this approach severely limited the throughput of data retrieval. An initiative to traverse a month of the block chain would take more than a day.

Because the RPC library made use of the `http` package from Node.js, some insight could be gathered on how to optimize this decreased performance. This package could make use of a certain *agent* that allowed connections that were already open to be reused. After modifying the library, an almost steady amount of 10 connections were maintained and reused for follow-up requests. This change sped up initial tests to traverse the block chain immensely.

All the block and transaction database functions work asynchronously by invoking a callback when the data is retrieved. The Node.js package `q` was used to tie all the functions together that would only advance in the case that a previous function was done. The package `underscore` was used to aid in object and array data retrieval.

C.1 Hardware

The block chain traversal was done on the following hardware.

CPU	Intel Core i7 950
Memory	12GiB
Storage	1x WDC WD1002FAEX-00Z3A0 (OS, Debian 8) 7x OCZ Vertex 2 RAID0 (Block chain databases)
RAID controller	LSI Logic / Symbios Logic MegaRAID SAS 2108

Traversing a month of transactions took about two hours to complete on above listed hardware.

C.2 Usage

The script works by creating a time span. This time span consists of a certain start date as Unix timestamp, incremental seconds and times to increment. This allows one to fine tune the period.

```
var timespansToCheck = createTimespan(1417399200, 86400, 31);
```

Listing 9: Time span for Decemer 2014

Only the time span has to be defined, the script will automatically run the sequence.

The detection script is started in a similar way as insight with the use of environment variables.

```
INSIGHT_NETWORK=testnet BITCOIND_USER=<user> BITCOIND_PASS=<
    password> INSIGHT_FORCE_RPC_SYNC=1 node
    detect_coinshuffle_transaction.js
```

Listing 10: Starting the CoinShuffle detection script

C.3 Possible problems

While the script is running, one may encounter an informational message shown by insight. The exact cause is not known, but stems from the function `_fillOutpoints()` in `lib/TransactionDb.js`, which is the transaction database interface. This function calls `fromTxIdN()` which has a comment "Gets address info from an outpoint". This comment suggests that it tries to retrieve address information from a certain output address. It is unknown if this causes some output addresses to miss from the transaction information, and therefore providing an incomplete overview. If this is the case, the recognition script might not be able to identify some transactions. The amount of these messages tend to increase the newer the block chain is. The website Blockchain.info lists some output addresses from such transaction as *Escrow* or identifies the transaction output address as being from a multi-signature address. Output addresses being multi-signature would not pose any problem for CoinShuffle transaction, as the original protocol does not make use of this.

```
info: Could not get TXouts in
f49cc29a871aff1448376fea8425ca0e64261c57123e13e74ef2b6f87a59bd1f,0
from
afcff654f8bd853e3f1345e60deed42f2ef892428204a4b223e8ad487eeb1d34
```

Listing 11: Unable to retrieve certain output address information

C.4 Database import

The block chain at the time of writing was about 42GiB. Downloading this using `bitcoind` takes a few days. Synchronizing the database with insight using the previously mention hardware takes about 12 hours.

C.5 Alternative traversal methods

There are a few alternatives to traverse the block chain. Public websites such as Blockchain.info [10] provides an API to retrieve information such as block and transaction information. These public API's usually have a hard limit of requests that can be made per day. Even though the data set that has to be queried only consists of a subset of the total block chain, this would put a lot of pressure on the API. Using the API also means creating a HTTP socket every time information has to be retrieved, as connections for a single request can not be reused. Creating this connection with the use of HTTP also introduces a bit of overhead for every request.

An up-to-date database dump for PostgreSQL is available, as well as dumps for the past six days [38]. Since the database is not tied to any language, one could be chosen to interface with the database that is fast for data parsing. While the database schema is provided, and the online API enables one to retrieve the joined data, no queries are available to join the different tables together. Due time constraints, this method was not chosen because creating an interface with the database with the added tooling to query the dataset would take too much time.

The block chain explorer ABE, which is written in python, provides a web interface where the block chain can be browsed [6]. ABE requires the block chain already being present on the system, which it will then synchronize to its own database. Because the web interface already provides an overview of blocks, transactions and addresses, there is already code available that interfaces with the database. These interfaces could be used to retrieve the required information. Because insight was already in place, it was decided not to use this method.