

CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin

Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate

MMCI, Saarland University

{tim.ruffing,pedro,aniket}@mmci.uni-saarland.de

Abstract. The decentralized currency network Bitcoin is emerging as a potential new way of performing financial transactions across the globe. Its use of pseudonyms towards protecting users' privacy has been an attractive feature to many of its adopters. Nevertheless, due to the inherent public nature of the Bitcoin transaction ledger, users' privacy is severely restricted to *linkable anonymity*, and a few transaction deanonymization attacks have been reported thus far.

In this paper we propose CoinShuffle, a completely decentralized Bitcoin mixing protocol that allows users to utilize Bitcoin in a truly anonymous manner. CoinShuffle is inspired by the accountable anonymous group communication protocol Dissent and enjoys several advantages over its predecessor Bitcoin mixing protocols. It does not require any (trusted, accountable or untrusted) third party and it is perfectly compatible with the current Bitcoin system. CoinShuffle introduces only a small communication overhead for its users, while completely avoiding additional anonymization fees and minimalizing the computation and communication overhead for the rest of the Bitcoin system.

Keywords: Bitcoin, decentralized crypto-currencies, coin mixing, anonymity, transaction linkability, mix networks.

1 Introduction

Bitcoin [1] is a fully decentralized digital crypto-currency network that does not require any central bank or monetary authority. Over the last few years we have observed an unprecedented and rather surprising growth of Bitcoin and its competitor currency networks [2, 3, 4]. Despite a few major hiccups, their market capitalizations are increasing tremendously [5]. Many now believe that the concept of decentralized crypto-currencies is here to stay.

Nevertheless, these decentralized currency systems are far from perfect. Traditional payment systems rely on a trusted third party (such as a bank) to ensure that money cannot be spent twice. Decentralized currencies such as Bitcoin employ a global replicated append-only transaction log and proof-of-work (POW) instead to rule out double-spending. This requires managing a public ledger such that every transaction is considered valid only after it appears in the ledger. However, given that the Bitcoin transactions of a user (in particular, of

her pseudonyms, called *Bitcoin addresses*) are linkable, the public transaction ledger constitutes a significant privacy concern: Bitcoin’s reliance on the use of pseudonyms to provide anonymity is severely restricted.

Several studies analyzing the privacy implications of Bitcoin indicate that Bitcoin’s built-in privacy guarantees are not satisfactory. Barber et al. [6] observe that Bitcoin exposes its users to the possible linking of their Bitcoin addresses, which subsequently leads to a weak form of anonymity. Meiklejohn et al. [7] demonstrate how to employ a few basic heuristics to classify Bitcoin addresses that are likely to belong to the same user; this is further refined by Spagnuolo, Maggi, and Zanero [8]. Koshy, Koshy, and McDaniel [9] show that it is possible to identify ownership relationships between Bitcoin addresses and IP addresses.

Recently, some efforts have been made towards overcoming the above attacks and providing stronger privacy to the Bitcoin users by *mixing* multiple transactions to make input and output addresses of transactions unlinkable to each other. In this direction, some third-party mixing services [10, 11, 12] were first to emerge, but they have been prone to thefts [7]. Mixcoin [13] allows to hold these mixing services accountable in a reactive manner; however, the mixing services still remain single points of failure and typically require additional mixing fees. Zerocoin [14] and its successors [15, 16, 17] provide strong anonymity without any third party, but lack compatibility with the current Bitcoin system.

Maxwell proposes CoinJoin [18] to perform mixing in a manner that is perfectly compatible with Bitcoin, while ensuring that even a malicious mixing server cannot steal coins. CoinJoin is actively used in practice [19] but suffers from a substantial drawback: The mixing server still needs to be trusted to ensure anonymity, because it learns the relation between input and output addresses. To tackle this problem, Maxwell mentions the possibility to use secure multi-party computation (SMPC) with CoinJoin to perform the mixing obviously without a trusted server. Yang [20] proposes a concrete scheme based on SMPC sorting. However, against a fully malicious attacker, generic SMPC as well as state-of-the-art SMPC sorting [21, 22] is not yet practical for any reasonable number of parties required in mixing to ensure a good level of anonymity. Furthermore, it is not clear how to ensure robustness against denial-of-service (DoS) attacks in these approaches, because a single user can easily disrupt the whole protocol while possibly remaining unidentified. As a result, defining a practical and secure mixing scheme is considered an open problem by the Bitcoin community [23, 24, 25].

Our Contribution. We present CoinShuffle, a completely decentralized protocol that allows users to mix their coins with those of other interested users. CoinShuffle is inspired by CoinJoin [18] to ensure security against theft and by the accountable anonymous group communication protocol Dissent [26] to ensure anonymity as well as robustness against DoS attacks. The key idea is similar to decryption mix networks, and the protocol requires only standard cryptographic primitives such as signatures and public-key encryption. CoinShuffle is a practical solution for the Bitcoin mixing problem and its distinguishing features are as follows:

No Third Party. CoinShuffle preserves Bitcoin’s decentralized trust ideology: it is executed exclusively by the Bitcoin users interested in unlinkability for

their Bitcoin transactions, and it does not require any trusted, accountable, or untrusted third party. The unlinkability of transactions is protected as long as at least any two participants in a run of the protocol are honest.

Compatibility. CoinShuffle is fully compatible with the existing Bitcoin network. Unlike other decentralized solutions, it works immediately on top of the Bitcoin network without requiring any change to the Bitcoin rules or scripts.

No Mixing Fee. In absence of a third party that acts as a service provider, CoinShuffle does not charge its users any additional mixing fees. It also performs well in terms of Bitcoin transaction fees, because the participants are only charged the fee for a single mixing transaction.

Small Overhead. Our performance analysis demonstrates that CoinShuffle introduces only a small communication overhead for a participant (less than a minute for an execution with 20 participants), while the computation overhead remains close to negligible. Finally, CoinShuffle introduces only minimal additional overhead for the rest of the Bitcoin network.

Outline. In Section 2, we explain the basics of the Bitcoin protocol and Bitcoin mixing. We define the problem of secure mixing in detail in Section 3. In Sections 4 and 5, we outline and specify the CoinShuffle protocol. We analyze its properties in Section 6 and evaluate its performance in Section 7. We discuss related work in Section 8 and conclude in Section 9.

2 Background

We start by presenting the basics of Bitcoin as well as Bitcoin mixing, the most prevalent approach to strengthening users' anonymity in the system. We explain only the aspects of the Bitcoin protocol that are relevant for mixing and refer the reader to the original Bitcoin paper [1] and the developer documentation [27] for further details.

2.1 Bitcoin

Bitcoin (symbol: ₿) is a digital currency run by a decentralized network. The Bitcoin network maintains a public ledger (called *blockchain*) whose purpose is to reach consensus on the set of transactions that have been validated so far in the network. As long as the majority of computation power in the system is honest, transactions accepted by the system cannot be changed or invalidated, thus preventing double-spending of money.

User accounts in the Bitcoin system are identified using pseudonymous addresses. Technically, an address is the hash of a public key of a digital signature scheme. To simplify presentation, we do not differentiate between the public key and its hash in the remainder of the paper. Every user can create an arbitrary number of addresses by creating fresh key pairs.

The owner of an address uses the corresponding private key to spend coins stored at this address by signing *transactions*. In the simplest form, a transaction

Input Addresses	Output Addresses
A: ₿2	X: ₿3
B: ₿7	Y: ₿2
	Z: ₿4
σ_A	
σ_B	




Fig. 1. A valid Bitcoin transaction with multiple input addresses and multiple output addresses. This transaction is signed using both the private key for input address A and the private key for input address B ; the corresponding signatures are denoted by σ_A and σ_B , respectively.

transfers a certain amount of coins from one address (the *input address*) to another address (the *output address*). While multiple sets of coins may be stored at one address, we assume in the remainder of the paper that only one set of coins is stored at an address; these coins can only be spent together. This simplification is purely for the sake of readability.

As depicted in Fig. 1, transactions can include multiple input addresses as well as multiple output addresses. Three conditions must be fulfilled for a transaction to be valid: First, the coins spent by the transaction must not have been already spent by another transaction in the blockchain. Second, the sum of the input coins must equal the sum of the output coins.¹ Third, the transaction must be signed with the private keys corresponding to all input addresses.

2.2 Bitcoin Mixing

The most prevalent approach to improve anonymity for Bitcoin users is the idea of hiding in a group by *Bitcoin mixing*: the users in the group exchange their coins with each other to hide the relations between users and coins from an external observer. Assume that in a group of several users, every user owns exactly one Bitcoin (₿1). In the simplest form, mixing is done with the help of a trusted third-party mixing server, the *mix*: every user sends a fresh address in encrypted form to the mix and transfers her coin to the mix. Then, the mix decrypts and randomly shuffles the fresh addresses and sends ₿1 back to each of them. While such public mixes are deployed in practice [28, 10, 11, 12], they suffer from two severe drawbacks: First, the mix might just steal the money and never return it to the users. Second, the mix learns which output address belongs to a certain input address. Thereby, users' anonymity relies on the assumption that the mix does not log or reveal the relation between input and output addresses.

2.3 Bitcoin Mixing with a Single Transaction

Assume a group of users would like to mix their coins with the help of a third-party mix. To solve the problem that the mix can steal the money, Maxwell proposes CoinJoin [18]: The mix generates one single *mixing transaction* containing the users' current addresses as inputs and the shuffled fresh addresses as

¹ In practice, a small transaction fee is typically required. In that case, the sum of the input coins must exceed the sum of the output coins by the amount of the fee.

outputs. Recall that a transaction with several input addresses is only valid if it has been signed with all keys belonging to those input addresses. Thus each user can verify whether the generated mixing transaction sends the correct amount of money to her fresh output address; if this is not true the user just refuses to sign the transaction and the protocol aborts without transferring any coins.

Several implementations of CoinJoin are already actively being used [19, 29, 30], and the Bitcoin developers consider adding CoinJoin to the official Bitcoin client [31]. Still, the problem that the mix learns the relation between input and output addresses persists, and no fully anonymous and efficient solution has been proposed to the best of our knowledge.

3 Problem Definition

In this section, we define the properties that a Bitcoin mixing protocol should satisfy. Furthermore, we present the threat model under which we would like to achieve these properties.

3.1 Design Goals

A Bitcoin mixing protocol must achieve the following security and privacy goals.

Unlinkability. After a successful Bitcoin mixing transaction, honest participants' input and output addresses must be unlinkable.

Verifiability. An attacker must not be able to steal or destroy honest participants' coins.

Robustness. The protocol should eventually succeed even in the presence of malicious participants as long as the communication links remain reliable.

Besides ensuring security and privacy, a Bitcoin mixing protocol must additionally overcome the following system-level challenges:

Compatibility. The protocol must operate on top of the Bitcoin network, and should not require any change to the existing system.

No Mixing Fees. The protocol should not introduce additional fees specifically required for mixing. As every mixing transaction necessarily requires a Bitcoin transaction fee, the protocol must ensure that this transaction fee remains as low as possible.

Efficiency. Even users with very restricted computational capacities should be able to run the mixing protocol. In addition, the users should not be required to wait for a transaction to be confirmed by the Bitcoin network during a run of the protocol, because this inherently takes several minutes.²

Small Impact on Bitcoin. The protocol should not put a large burden on the efficiency of the Bitcoin network. In particular, the size of the executed transactions should not be prohibitively large because all transactions have to be stored in the blockchain and verified by all nodes in the network.

² Several confirmations are recommended, each taking 10 minutes on average. As mixing inherently requires at least one transaction, it is adequate to wait for confirmations at the end of a run, provided the run fails gracefully if the transaction is not confirmed.

3.2 Non-goals

Bitcoin users who wish to participate in a mixing protocol need a bootstrapping mechanism to find each other, e.g., through a public bulletin board acting as facilitator or through a peer-to-peer protocol specifically crafted for this purpose. A malicious facilitator may try to undermine unlinkability by forcing an honest participant to run the protocol only with malicious participants. Thus, in general, the bootstrapping mechanism should resist attempts to exclude honest users from the protocol. Since the Bitcoin network does not allow nodes to send arbitrary messages, the participants must additionally agree on a channel for further communication during bootstrapping. We consider bootstrapping to be orthogonal to our work and assume that it is available to all Bitcoin users.

The main goal of a Bitcoin mixing protocol is the unlinkability between input and output addresses in a mixing transaction. If *after* the mixing, a user would like to spend the mixed coins associated with the output address while maintaining her anonymity, she has to ensure that network metadata, e.g., her IP address, does not reveal her identity or make the spending transaction linkable to a run of the mixing protocol. This problem is not in the scope of the Bitcoin mixing protocol and can be addressed, e.g., by connecting to the Bitcoin network via an anonymous communication protocol such as Tor [32].

3.3 Threat Model

For unlinkability and verifiability, we assume an active network attacker. (Robustness cannot be ensured in the presence of an active network attacker, because such an attacker can always stop the communication between the honest participants.)

We do *not* require *any* trust assumption on a particular party: for verifiability and robustness, we assume that an honest participant can be faced with an arbitrary number of malicious participants. For unlinkability, we require that there are at least two honest participants in the protocol. Otherwise the attacker can trivially determine the mapping between input and output addresses and meaningful mixing is not possible.

4 Solution Overview

Our main contribution is *CoinShuffle*, a Bitcoin mixing protocol that achieves the aforementioned goals. In this section, we give an overview of our solution.

4.1 Main Idea

To ensure verifiability, our protocol follows the CoinJoin paradigm (Section 2.3): A group of users jointly create a single mixing transaction and each of them can individually verify that she will not lose money by performing the transaction. In case of a fraud attempt, the defrauded user can just refuse to sign the transaction.

Unlinkability and robustness, however, are the most challenging problems: To create a mixing transaction while assuring that input addresses are not linkable

to fresh output addresses, the participants shuffle their output addresses in an oblivious manner, similar to a decryption mix network [33]. This shuffling is inspired from one phase of the accountable anonymous group messaging protocol Dissent [26, 34], which builds on an anonymous data collection protocol due to Brickell and Shmatikov [35]. We are able to simplify and optimize ideas from Dissent. For instance, the number of encryption operations is reduced by a factor of four. Even though the special nature of the problem that we would like to solve enables most of these optimizations, we conjecture that one of them is not particular to our setting and can be applied to Dissent. We refer readers that are familiar with Dissent to Appendix A for details and a high-level comparison.

The shuffling provides robustness in the sense that attacks that aim to disrupt the protocol can be detected by honest users and at least one misbehaving participant can be identified and excluded.³ The other participants can then run the protocol again without the misbehaving participant.

4.2 Protocol Overview

The main part of the CoinShuffle protocol can roughly be split into three phases as depicted in Fig. 2. (As elaborated later, the complete instantiation contains more phases.) If the protocol does not run successfully, an additional blame phase will be reached. In the following we give an overview of every phase. Assume that every participant holds the same amount of coins at some Bitcoin address. This address will be one of the input addresses in the mixing transaction, and every protocol message from this participant is supposed to be signed with the private signing key that belongs to this address.

Announcement. Every participant generates a fresh ephemeral encryption-decryption key pair, and broadcasts the resulting public encryption key.

Shuffling. Every participant creates a fresh Bitcoin address, designated to be her output address in the mixing transaction. Then the participants shuffle the freshly generated output addresses in an oblivious manner, similar to a decryption mix network [33].

In more detail, every participant (say participant i in a predefined shuffling order) uses the encryption keys of all participants $j > i$ to create a layered encryption of her output address. Then, the participants perform a sequential shuffling, starting with participant 1: Each participant i expects to receive $i - 1$ ciphertexts from participant $i - 1$. Upon reception, every participant strips one layer of encryption from the ciphertexts, adds her own ciphertext and randomly shuffles the resulting set. The participant sends the shuffled set of ciphertexts to the next participant $i + 1$. If everybody acts according to the protocol, the decryption performed by the last participant results in a shuffled list of output addresses. The last participant broadcasts this list.

³ This property is called *accountability* in Dissent. We use a different term to avoid confusion with the concept of accountable Bitcoin mixing services in Mixcoin [13].

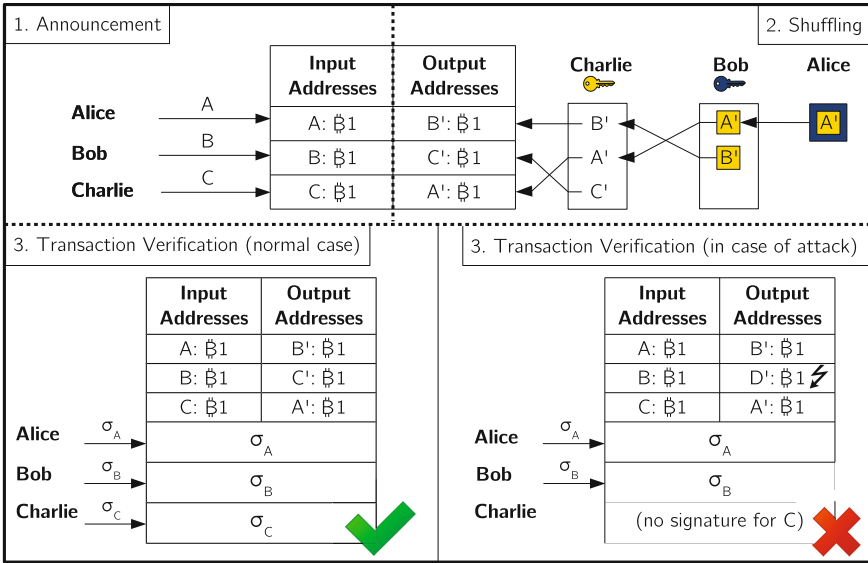


Fig. 2. Overview of CoinShuffle: First, the participants announce their input addresses. Second, they shuffle their fresh output addresses obliviously. (Colored boxes represent ciphertexts encrypted with the respective encryption key.) Third, the participants check if all their output addresses are contained in the final list of output addresses. In this case (left-hand side), the transaction is signed by the participants and submitted to the Bitcoin network. If, on the contrary, an output address is missing (e.g., C' has been replaced by D' , right-hand side), the transaction does not become valid and the participants enter the blame phase to find out which participant deviated from the protocol specification. Even though not explicit in the figure, all messages are signed.

Transaction Verification. Each participant can individually verify if her output address is indeed in the list. If this is true, every participant deterministically creates a (not yet signed) mixing transaction that spends coins from all input addresses and sends them to the shuffled list of output addresses. Every participant signs the transaction with her Bitcoin signing key and broadcasts the signature.

Upon receiving signatures from all other participants, every participant is able to create a fully-signed version of the mixing transaction. The transaction is thus valid and can be submitted to the Bitcoin network.

Blame. In every step of the previous phases, every participant checks that all other participants follow the protocol. If some participant deviates from the protocol, an honest participant would report the deviation and the protocol enters the blame phase, which is then performed to identify the misbehaving participant. The misbehaving participant can then be excluded from a subsequent run of the protocol. There are three cases in which participants enter the blame phase. First, the blame phase is entered if some participant does not have enough coins at her input address to perform the mixing transaction, or if she just spends the

money at the input address before the mixing protocol is completed. In both situations, the Bitcoin network provides evidence for the misbehavior. Second, the blame phase is entered if the shuffling has not been performed correctly. In that case, the participants can broadcast their ephemeral decryption keys, along with the messages they have received. This information allows every participant to replay the computations of the rest of participants and expose the misbehaving one. Third, participants could equivocate in the broadcasts of the protocol, e.g., by sending different public keys to different participants in the announcement phase. All participants exchange messages before creating the mixing transaction to ensure that nobody has equivocated. In case of equivocation, the blame phase is entered. Since all protocol messages are signed, the equivocating participant can be identified; two signed messages that are different but belong to the same sender and the same broadcast step provide evidence of the misbehavior.

5 The CoinShuffle Protocol

This section details the CoinShuffle protocol, first by covering its cryptographic building blocks and later by formally describing the protocol.

5.1 Cryptographic Primitives

To connect CoinShuffle to Bitcoin, the participants use the Elliptic Curve Digital Signature Algorithm (ECDSA) already deployed in Bitcoin. Formally, we require the signature scheme in CoinShuffle to be (weakly) unforgeable under chosen-message attacks (UF-CMA). Given a message m and a signing key sk , we denote by $\text{Sig}(sk, m)$ the signature of m using sk . The verification algorithm $\text{Verify}(vk, \sigma)$ outputs 1 if σ is a valid signature for m under the verification key vk .

CoinShuffle requires an IND-CCA secure public-key encryption scheme. We denote by $\text{Enc}(ek, m)$ the ciphertext that encrypts the message m with the encryption key ek . For all possible outputs (ek, dk) of the key generation algorithm, we have that if c is a valid ciphertext encrypted with encryption key ek , then the decryption algorithm $\text{Dec}(dk, c)$ outputs the message m contained in c , or \perp otherwise. The encryption scheme must adhere to several additional conditions: First, it must be possible to check if a pair of bitstrings (ek, dk) is a valid key pair, i.e., a possible output of the key generation algorithm. This can be achieved, e.g., as described in [26, Appendix]. Second, we require the encryption algorithm Enc to be length-regular, i.e., for all encryption keys ek and messages m and m' with $|m| = |m'|$, we have $|\text{Enc}(ek, m)| = |\text{Enc}(ek, m')|$ with probability 1. We denote the layered encryption of m with multiple keys by $\overline{\text{Enc}}((ek_1, \dots, ek_n), m) := \text{Enc}(ek_1, \text{Enc}(ek_2, \dots \text{Enc}(ek_n, m) \dots))$. Finally, we require a collision-resistant hash function H .

5.2 Core Protocol Description

We assume that every participant $i \in \{1, \dots, N\}$ already possesses a Bitcoin address, i.e., a public verification key vk_i and the corresponding signing key sk_i .

The address vk_i will be one of the *input addresses* of the mixing transaction. The order of the participants is publicly known, e.g., the lexicographical order of the verification keys. We further assume that every participant already knows the verification keys of all other participants. All participants have already agreed upon a fresh session identifier τ and an amount $\mathfrak{B}\nu$ of coins that they would like to mix. Since the participants use their private Bitcoin keys to sign protocol messages, we require an encoding that ensures that protocol messages are distinct from Bitcoin transactions. This guarantees that participants cannot be tricked into signing transactions unknowingly. During the whole protocol, parties ignore incorrectly signed messages and unexpected messages.

To simplify presentation, we assume implicitly that signed messages can be extracted from their signatures. We write $\sigma_{a,b}$ for the signature produced by participant a in phase b .

Phase 1: Announcement. Every participant $i \in \{2, \dots, N\}$ randomly chooses a fresh ephemeral encryption-decryption key pair (ek_i, dk_i) and broadcasts $\sigma_{i,1} = \text{Sig}(sk_i, (ek_i, 1, \tau))$. After participant i receives a correctly signed message $\sigma_{j,1}$ from each participant j , she checks that the address vk_j holds at least $\mathfrak{B}\nu$ to ensure that enough money is available to carry out the mixing transaction. Otherwise, participant i enters the blame phase.

Phase 2: Shuffling. Every participant chooses a fresh Bitcoin address, i.e., the verification key vk'_i of a fresh verification-signing key pair (vk'_i, sk'_i) . The signing key sk'_i is kept secret and can be used to spend the mixed coins that will be associated with the *output address* vk'_i after a successful run of the protocol.

Participant 1 creates a layered encryption $c_1 = \overline{\text{Enc}}((ek_2, \dots, ek_N), vk'_1)$ of her output address vk'_1 and sends $\sigma_{1,2} = \text{Sig}(sk_1, (C_1, 2, \tau))$ to participant 2, where $C_1 = (c_1)$ is the unary vector with the component c_1 . Upon receiving a vector C_{i-1} , participant $i \in \{2, \dots, N-1\}$ decrypts each message in the vector. Afterwards, she encrypts her Bitcoin output address vk'_i with the public keys of the remaining $(N-i)$ participants, obtaining $c_i = \overline{\text{Enc}}((ek_{i+1}, \dots, ek_N), vk'_i)$. Then participant i adds c_i to the vector of decrypted messages and shuffles the extended vector randomly, obtaining a new vector C_i . If a decryption fails or if two decryption operations lead to the same output, participant i enters the blame phase. Otherwise, participant i sends $\sigma_{i,2} = \text{Sig}(sk_i, (C_i, 2, \tau))$ to participant $i+1$.

Phase 3: Broadcast of the Output. Upon receiving $\sigma_{N-1,2}$, participant N strips the last layer of encryption of every ciphertext in the vector C_{N-1} . Then participant N shuffles the resulting vector of output addresses after extending it by her own output address vk'_N , obtaining the final vector T_{out} . Finally, participant N broadcasts $\sigma_{N,3} = \text{Sig}(sk_N, (T_{out}, 3, \tau))$ to the rest of the participants. If the protocol has been correctly carried out by all participants, every participant has received a copy of the shuffled vector T_{out} of output addresses at this point. Every participant i checks if her output address vk'_i is contained in T_{out} , and otherwise enters the blame phase.

Phase 4: Equivocation Check. To ensure that nobody has equivocated during a broadcast, every participant i computes $h_i = \text{H}((ek_2, \dots, ek_N), T_{out})$ and broadcasts $\sigma_{i,4} = \text{Sig}(sk_i, (h_i, 4, \tau))$. After having received a correctly signed

message from each participant j , participant i checks if there are two participants a and b with $h_a \neq h_b$. In this case, participant i enters the blame phase.

Phase 5: Transaction Verification and Submission. Every participant deterministically creates a (not yet signed) mixing transaction tx that spends $\mathfrak{B}\nu$ from each of the input addresses in $T_{in} = (vk_1, \dots, vk_N)$ and sends $\mathfrak{B}\nu$ to each of the output addresses in T_{out} . Participant i signs the transaction tx according to the specification of the Bitcoin protocol and broadcasts the signature $\sigma_{i,5} = \text{Sig}(sk_i, tx)$. Upon receiving a valid signature $\sigma_{j,5}$ from each participant j , participant i adds all signatures to tx and submits the resulting valid transaction to the Bitcoin network. Participant i checks if any of the other participants has spent her money reserved for mixing in the meantime. If this is the case, participant i enters the blame phase. Otherwise the protocol is finished.

Phase 6: Blame. This phase is only reached when any of the checks described above fails. When a participant i enters the blame phase, it broadcasts a signed message explaining the reason for entering the blame phase. Depending on the failed check, additional information may be included as follows:

1. If the Bitcoin network reports that the value of the coins at an input address is below $\mathfrak{B}\nu$, or that the coins at an input address vk_j have already been spent, participant i broadcasts the transaction that sent the insufficient coins to the input address or the transaction that spent the coins, respectively.
2. If there are participants i and j with $h_i \neq h_j$: Participants i and j publish all signed messages that have been received in phase 1 and phase 3. Note that these messages contain all encryption keys (ek_2, \dots, ek_N) and the final vector T_{out} . Every participant recomputes h_i and h_j and checks if they have been correctly reported. If not, this exposes participant i or j . If both h_i and h_j have been reported correctly, there are two cases: First, a participant has equivocated in phase 1 by sending different encryption keys to i and j . Second, participant N has equivocated in phase 3 by sending different vectors of output addresses to i and j . In either case, the published messages expose the misbehaving participant.
3. If in phase 2, a decryption fails, a duplicate ciphertext is detected, or if after phase 2 an output address is missing in the final vector, the participants perform the skipped equivocation check in phase 4, but only for the encryption keys: Every participant i computes $h'_i = H((ek_2, \dots, ek_N))$ and broadcasts $\text{Sig}(sk_i, (h'_i, 4, \tau))$. After having received a correctly signed message from each participant j , participant i checks that there are no two participants a and b with $h'_a \neq h'_b$. Otherwise, the protocol continues as in the case above. If the equivocation check succeeds, every participant i signs and broadcasts her decryption key dk_i together with all messages that have been received in phases 2 and 3. The participants verify that all key pairs (ek_i, dk_i) are valid and blame the participant with an invalid key pair otherwise. If all key pairs are valid, the participants have enough information to replay phases 2 and 3 on their own and identify at least one misbehaving participant.

At the end of the blame phase, at least one misbehaving participant is identified and excluded from the protocol. The remaining participants can start a new run of the protocol without the misbehaving participant, using a fresh session identifier.

It is worth noting that, whenever the blame phase is reached, the participants do not construct a transaction that is accepted by the Bitcoin network.

5.3 Practical Considerations

Transaction Fees. In practice, the Bitcoin network charges a small fee for mixing transactions⁴ to prevent DoS attacks that flood the network with a large number of transactions [36]. Transaction fees can easily be dealt with in CoinShuffle. Before creating the transaction, the N participants calculate the required fee μ and reduce the size of each output by μ/N , splitting the fee equally among all participants. This ensures that the transaction will be accepted by the Bitcoin network. If a participant tries to cheat by deviating from this policy, e.g., to pay a lower fee, the mixing transaction will not become valid as only the correct transaction will be signed by the honest participants.

Change Addresses. A user that would like to spend exactly $\mathfrak{B}x$ typically does not hold an input address with exactly this balance, but rather an address with a higher balance $\mathfrak{B}(x+y)$. In order to perform the payment, the user will create a transaction with one input $\mathfrak{B}(x+y)$ and two outputs: $\mathfrak{B}x$ go to the address of the payee and $\mathfrak{B}y$ go to a *change address* belonging to the original user.

The use of change addresses is supported in CoinShuffle: Participants can announce additional change addresses in phase 1, if they do not have an address holding exactly the mixing amount $\mathfrak{B}\nu$. In phase 5, every participant adds all the change addresses as outputs of the mixing transaction tx before it is signed. CoinShuffle still preserves the unlinkability between the input addresses and the (regular) output addresses of the honest participants.

Communication and Liveness. In practice, broadcasts can be implemented by sending all messages to a randomly chosen *leader* that relays the messages to all participants. Furthermore, instead of misbehaving actively, participants might passively disrupt a protocol run by simply going offline at any time, either maliciously or due to a network failure or asymmetric connectivity. This problem is not particular to CoinShuffle and can be handled using the same techniques as in Dissent [26], which in turn borrows ideas from PeerReview [37]. We only present the idea and refer the reader to the original papers for details. When the protocol states that a participant i must receive a properly signed message from participant j , but participant j does not send such a message within a predefined timeout period, i suspects j . In this case, i asks another participant k (or several participants) to request the desired message from j and relay it to i . If k does not receive the message either, also k suspects j and can in turn ask other members. In case nobody receives the message from j , i.e., everybody suspects j eventually, the participants can start a new run of the protocol without j .

⁴ At the time of writing, a fee of $\mathfrak{B}0.0001$ ($\approx \$0.06$) per 1000 bytes of transaction size is mandatory for transactions of at least 1000 bytes. Due to their nature, mixing transactions contain several addresses and are typically larger than 1000 bytes. A mixing transaction with 25 participants has an approximate size of 5000 bytes.

6 Analysis

We discuss why CoinShuffle achieves the design goals described in Section 3.1.

6.1 Security Analysis

Recall that we aim for three security and privacy properties, namely unlinkability, verifiability and robustness. We explain why CoinShuffle achieves all of these.

Unlinkability. A Bitcoin mixing protocol provides unlinkability if given a single output address vk' from a successful mixing transaction, the scenario that vk' belongs to some honest user a is indistinguishable from the scenario that vk' belongs to a different honest user $b \neq a$.

First, observe that we do not have to consider failed runs of the protocol. Indeed, if the blame phase is reached, the attacker might be able to link an output address to an input address, e.g., if the participants publish decryption keys. However, if the blame phase is reached, the mixing transaction and in particular the generated output addresses are discarded and the protocol will be restarted with fresh output addresses.

Now consider a successful run of the protocol, i.e., assume that the blame phase has not been reached. Observe that phase 4 of the protocol ensures that no participant has equivocated while announcing her ephemeral encryption key in phase 1. Let i be the highest index of an honest participant in the shuffling order and let $U_{<i}$ be the set of honest participants with index smaller than i . Participant i has received a vector C_{i-1} of $i - 1$ ciphertexts. All messages that have been sent so far in the shuffling phase have been encrypted with ek_i (and other keys). These messages do not reveal the link between input and output addresses, because the attacker does not know dk_i and thus cannot observe which output address is contained in which layered ciphertext.

We continue by arguing that the output of participant i does not reveal the link between input and output addresses either. Since the shuffling has been performed successfully and the blame phase has not been reached, the ciphertexts in vector C_{i-1} that belong to the users in $U_{<i}$ have not been tampered with. Furthermore, because we have excluded equivocation, these ciphertexts share the same structure, i.e., they are all of the form $\overline{\text{Enc}}((ek_i, \dots, ek_N), vk'_j)$ for $j \in U_{<i}$ and uniquely defined encryption keys ek_i, \dots, ek_N . Participant i strips one layer of encryption from the ciphertexts in C_{i-1} , adds her own ciphertext and shuffles the resulting vector C_i . Consequently, participant i outputs a randomly shuffled vector C_i that contains at least $|U_{<i}| + 1$ honestly generated ciphertexts of the form $\overline{\text{Enc}}((ek_{i+1}, \dots, ek_N), vk'_j)$ for $j \in U_{<i} \cup \{i\}$, where all output addresses vk'_j have the same fixed length, because they are Bitcoin addresses. Let D_i be the a vector that is obtained by keeping only those honestly generated ciphertexts and removing the others from C_i . D_i is implicitly associated with a permutation π of the output addresses of the honest participants in $U_{<i} \cup \{i\}$.

Since i is honest and does not collude with malicious participants, the IND-CCA property of the encryption scheme ensures that all pairs of possible output vectors D_i^0 and D_i^1 (resulting from potentially different permutations π_0 and π_1 of

the output addresses) are indistinguishable.⁵ Note that at least two different permutations $\pi_0 \neq \pi_1$ exist, because by assumption, there are at least two honest participants whose ciphertexts can be shuffled, which implies $|U_{<i}| \geq 1$.

Verifiability. A Bitcoin mixing protocol ensures verifiability if no attacker can steal or destroy honest participants' coins. This is immediate from the description of CoinShuffle: a honest participant i signs the final mixing transaction only if she has verified that (i) her own output address vk'_i is included in the list of output addresses, and (ii) the amount sent to the output address is the amount of coins taken from her input address (possibly reduced by a transaction fee).

Robustness. A Bitcoin mixing protocol ensures robustness if it finishes even in the presence of malicious participants. Since CoinShuffle enters the blame phase if a run does not successfully create a transaction, we have to argue why at least one misbehaving participant can be identified in the blame phase. We distinguish the same cases as in the blame phase of the protocol description:

1. In this case, the signed announcement message together with the evidence from the Bitcoin network proves that the participant in question misbehaved.
2. Recall that participants i and j publish all signed messages that have been received in phases 1 and 3. If h_i or h_j have been computed incorrectly, this is evidence that i or j , respectively, has misbehaved. If both h_i and h_j have been reported correctly, the published messages expose the equivocating participant.
3. If a participant detects an invalid key pair, the signed announcement message (containing the purported encryption key) and the signed message in the blame phase containing the purported decryption key provide evidence of misbehavior. Otherwise, the participants have enough information to replay the steps of each participant in phases 2 and 3 and identify the misbehaving participant. The signed messages of phases 1 to 3 prove the misbehavior.

Double-Spending. Note that due the nature of the Bitcoin network, a malicious participant might disrupt the protocol by a double-spending attempt: Shortly before all participants submit the mixing transaction to the network, the malicious participant submits a transaction that spends her input coins that have actually been designated for mixing. The Bitcoin network will eventually reach consensus which of the two transactions becomes valid and discard the other one to ensure that coins cannot be double-spent. If the malicious transaction is accepted, honest parties do not lose their coins, but the mixing will have failed. Then, it might be the case that a restart of the protocol is not possible because the participants have already gone offline, in the belief that the protocol has been successful.

We consider protection against double-spending in Bitcoin to be orthogonal to our work [38]. Typically, the attacker's goal in double-spending is to make a recipient of a transaction believe that she has received some coins. As a result

⁵ Note that the length-regularity of Enc implies that not only the output addresses but also the inner layers of encryptions (at the same depth) have the same length. This is necessary, because otherwise the IND-CCA property does not guarantee indistinguishability for the encryptions of these inner ciphertexts.

the recipient will, e.g., hand over a valuable good to the attacker, even though the transaction will be invalidated and replaced by a different one that sends the coins back to attacker. However, this attack is not possible in mixing, because sender and recipient are the same party. Instead, invalidating a mixing transaction is only an attack against robustness. If protection against a double-spending attack becomes necessary to ensure robustness, the participants will have to wait for the transaction to be confirmed by the Bitcoin network before they go offline.

6.2 System Analysis

We discuss why CoinShuffle achieves the desired system-level goals.

Compatibility. CoinShuffle does not require any change to the Bitcoin protocol or to the transaction format, because a successful run of CoinShuffle results in a transaction that is valid according to the current rules of Bitcoin. Thus the protocol is immediately deployable.

No Mixing Fees. Systems in which a trusted third party performs the mixing typically charge users two fees: a transaction fee as defined in Bitcoin and a mixing fee required by the trusted third party [10, 11, 12]. In CoinShuffle, however, no mixing fee is required. Users who jointly execute CoinShuffle are only charged the transaction fee as defined in the currently deployed Bitcoin protocol.

Efficiency. As signatures and hash functions are already used in Bitcoin, public-key encryption is the only cryptographic primitive added by CoinShuffle. This allows to run the protocol even on computationally restricted hardware. The performance evaluation (Section 7) shows the practical feasibility of CoinShuffle.

Small Impact on Bitcoin. Upon successful protocol execution, the participants jointly create only a single Bitcoin transaction that must be stored in the public blockchain and has to be verified by all nodes in the network. Thus, the execution of CoinShuffle introduces only a minimal overhead in terms of storage and computation for nodes in the Bitcoin network.

7 Performance Evaluation

We have developed a proof-of-concept implementation [39] of CoinShuffle leveraging an existing implementation of the Dissent protocol. In particular, we have implemented phases 1 to 5 of the protocol (Section 5.2), which suffice to measure the performance of a single successful run without disruption.

The implementation is written in Python and uses OpenSSL to sign and encrypt messages. As required by the Bitcoin network, signatures have been implemented using ECDSA on the `secp256k1` elliptic curve [40] at a security level of 128 bits. We use the Elliptic Curve Integrated Encryption Scheme (ECIES) [40] on the same curve together with standard AES in CBC mode for encryption. The communication among the participants has been implemented using TCP. When a message is broadcast, it is first sent to the first participant in the shuffling order, who in turn sends a copy to every participant.

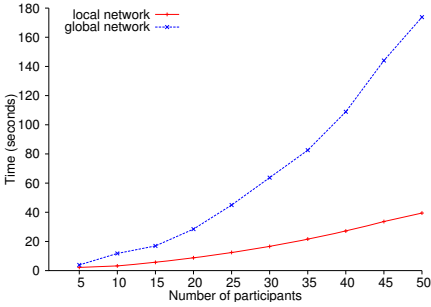


Fig. 3. Overall execution time

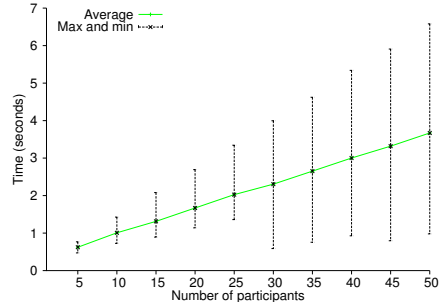


Fig. 4. Processing time per node

We tested our implementation in Emulab [41], a testbed for distributed systems, in which network parameters such as topology or bandwidth of links can be easily configured. In this setting, we have run several experiments under controlled network conditions. We consider two scenarios: a local network and a global network. In the former, we connected all the participants to a LAN with 100 Mbit/s bandwidth without delays. In the latter, we split the participants in two LANs of 100 Mbit/s bandwidth each. Both LANs were connected through a router with a bandwidth of 20 Mbit/s and a delay of 50 ms. In the global network scenario we considered the worst case for the shuffling phase: participants with an odd index in the shuffling order were placed in one LAN while participants with an even index were placed in the other LAN. Thus every message in the shuffling phase had to traverse the whole network.

We have run the protocol with different numbers of participants, ranging from 5 to 50. Figure 3 shows the overall time needed to create a Bitcoin transaction in a run without misbehaving participants. In the local network, 50 participants need approximately 40 seconds to run CoinShuffle, while in the global network, slightly less than 3 minutes are necessary to complete the protocol.

Figure 4 shows the overhead of the computation carried out by every participant on average. As expected, the average processing time increases linearly with the number of participants, because every participant must shuffle a vector of ciphertexts containing one ciphertext more than the previous participant. Furthermore, the computation overhead constitutes only a small fraction of the overall time. In the case of 50 participants, the average computation time is slightly larger than 3 seconds, which constitutes approximately 2% of the overall time in the local network scenario and less than 1% in the global network setting.

In summary, the experimental results demonstrate the feasibility of the CoinShuffle protocol even in scenarios with a large number of participants.

8 Related Work

Zerocoin [14], an extension to Bitcoin, was among the first proposals to provide unlinkability between individual Bitcoin transactions without introducing a trusted party. It employs a cryptographic accumulator of minted *zerocoins* and

a zero-knowledge proof of inclusion of a certain zerocoin within the accumulator. Zerocoin introduces a significant computation and communication overhead: the size of the proof that has to be stored in the blockchain for each transaction is prohibitively large (i.e., approximately 25 KB) and far exceeds the size of the Bitcoin transaction itself.

Recently, there have been some proposals to reduce the Zerocoin proof size. Garman et al. [16] propose a set of extensions to Zerocoin that reduces the proof size by modeling the cost of forging a coin and picking cryptographic parameters to make such forgery uneconomical. Both Pinocchio Coin [15] and Zerocash [17] are promising improvements of Zerocoin that significantly reduce the proof size (to less than 1 KB) and the computational costs. Nevertheless, this line of research is severely restricted in terms of adaptability. Zerocoin and all of the above extensions require substantial modifications to the Bitcoin system. Thus, Zerocoin and its variants cannot be directly deployed in Bitcoin. Instead, they would need an incremental deployment that requires acceptance by the majority of the Bitcoin nodes (measured in computational resources). So far, it looks unlikely for the Bitcoin network to employ the Zerocoin strategy [42]. In contrast, while requiring more communication, CoinShuffle is immediately adaptable and works on top of the existing Bitcoin network.

The Mixcoin [13] protocol facilitates anonymous Bitcoin payments without making any modifications to the Bitcoin protocol. Here, Bitcoin users send their coins to a central accountable mix which in turn replies with a guarantee of returning the funds to the user. Afterwards, the mix sends the coins back to the user ensuring unlinkability between the user input and output addresses. Although the mix can be held accountable for thefts, the system still has several drawbacks. First, the use of a central mix introduces a single point of failure, where the mix becomes a suitable target for DoS attacks from competing mixes as well as governmental agencies. Second, the provided accountability is reactive in nature, and the mix can still steal users' coins before going out of business. Third, a payment in Mixcoin requires two Bitcoin transactions and additionally a fee charged by the mix. Finally, unlinkability is only guaranteed against external observers, because mix learns which address belongs to which user. In comparison to Mixcoin, CoinShuffle relies on the interaction between the users in the mixing to achieve unlinkability, verifiability, robustness, and cost effectiveness without a trusted third party.

Maxwell [43] sketches a modification to the CoinJoin protocol using blind signatures to avoid the problem of a centralized mix learning the relation between input and output addresses. This protocol employs the anonymous communication network Tor [32] as a building block to provide unlinkability. In contrast, CoinShuffle provides full resistance against traffic correlation attacks by using a decentralized high-latency mix network run only by the participants.

9 Conclusion

The linkable pseudonymity provided by the Bitcoin system leads to significant privacy concerns for its users. A few solutions that aim at mixing transactions

of a group of users have been proposed in the last two years to address this concern; however, none of them has been found to be satisfy all requirements of a practical and compatible solution. In this paper, we have presented the Bitcoin mixing protocol CoinShuffle, which is secure, robust, and perfectly compatible with the existing Bitcoin system. Adhering to the Bitcoin ideology, CoinShuffle is completely decentralized, and it neither requires any third party nor introduces any additional anonymization fees for the users.

We implemented CoinShuffle and tested it in a local as well as in a global network scenario in the Emulab environment. Our experiments demonstrate that CoinShuffle introduces only a small (suitable for Bitcoin) computation and communication overhead to a participant, even when the number of CoinShuffle participants is large (≈ 50). Moreover, CoinShuffle leads only to a minimal overhead for the Bitcoin blockchain and thus for the rest of the Bitcoin network.

Finally, although we have focused on the crypto-currency Bitcoin in the paper, we stress that our protocol is compatible with all competing currencies derived from Bitcoin, e.g., Litecoin [2], Mastercoin [4], and others.

Acknowledgments. We thank Bryan Ford for his insightful comments on an earlier draft and Henry Corrigan-Gibbs for helping us with running the proof-of-concept implementation on Emulab. We further thank the anonymous reviewers for their helpful comments. This work was supported by the German Universities Excellence Initiative.

References

- [1] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Technical report (2008), <https://bitcoin.org/bitcoin.pdf>
- [2] Litecoin, <https://litecoin.org>
- [3] Ripple, <https://ripple.com>
- [4] Mastercoin, <http://www.mastercoin.org>
- [5] BitInfoCharts, <http://bitinfocharts.com/comparison/transactions-marketcap-btc-ltc.html> (accessed on March 28, 2014)
- [6] Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better — how to make Bitcoin a better currency. In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 399–414. Springer, Heidelberg (2012)
- [7] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: Characterizing payments among men with no names. In: IMC 2013, pp. 127–140. ACM (2013)
- [8] Spagnuolo, M., Maggi, F., Zanero, S.: BitIodine: Extracting intelligence from the Bitcoin network. In: FC 2014. Springer (2014)
- [9] Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in Bitcoin using P2P network traffic. In: FC 2014. Springer (2014)
- [10] Bitcoin Fog, <http://www.bitcoinfo.com>
- [11] BitLaundry, <http://app.bitlaundry.com>
- [12] BitLauder, <https://bitlaunder.com>
- [13] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J.A., Felten, E.W.: Mixcoin: Anonymity for Bitcoin with accountable mixes. In: FC 2014. Springer (2014)

- [14] Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from Bitcoin. In: S&P 2013, pp. 397–411. IEEE Press (2013)
- [15] Danezis, G., Fournet, C., Kohlweiss, M., Parno, B.: Pinocchio Coin: Building Zerocoin from a succinct pairing-based proof system. In: PETShop 2013, pp. 27–30. ACM (2013)
- [16] Garman, C., Green, M., Miers, I., Rubin, A.D.: Rational Zero: Economic security for Zerocoin with everlasting anonymity. In: 1st Workshop on Bitcoin Research (2014), https://fc14.ifca.ai/bitcoin/papers/bitcoin14_submission_12.pdf
- [17] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: S&P 2014. IEEE Press (2014)
- [18] Maxwell, G.: CoinJoin: Bitcoin privacy for the real world. Post on Bitcoin Forum (August 2013), <https://bitcointalk.org/index.php?topic=279249>
- [19] Qkos Services Ltd.: Shared Coin, <https://sharedcoin.com>
- [20] Yang, E.Z.: Secure multiparty Bitcoin anonymization. Blog posting (2012), <http://blog.ezyang.com/2012/07/secure-multiparty-bitcoin-anonymization/>
- [21] Jónsson, K.V., Kreitz, G., Uddin, M.: Secure multi-party sorting and applications. IACR ePrint Cryptology Archive 2011/122, <https://eprint.iacr.org/2011/122>
- [22] Hamada, K., Kikuchi, R., Ikarashi, D., Chida, K., Takahashi, K.: Practically efficient multi-party sorting protocols from comparison sort algorithms. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 202–216. Springer, Heidelberg (2013)
- [23] Rosenfeld, M.: Using mixing transactions to improve anonymity. Post on Bitcoin Forum (December 2011), <https://bitcointalk.org/index.php?topic=54266>
- [24] Murphant (pseudonym). Post on Bitcoin Forum (August 2013), <https://bitcointalk.org/index.php?topic=279249.msg3057216#msg3057216>
- [25] Maxwell, G.: Post on Bitcoin Forum (September 2013), <https://bitcointalk.org/index.php?topic=279249.msg3013970#msg3013970>
- [26] Corrigan-Gibbs, H., Ford, B.: Dissent: Accountable anonymous group messaging. In: CCS 2010, pp. 340–350. ACM (2010)
- [27] Bitcoin project: Bitcoin developer documentation, <https://bitcoin.org/en/developer-documentation>
- [28] Möser, M., Böhme, R., Breuker, D.: An inquiry into money laundering tools in the Bitcoin ecosystem. In: ECRIME 2013. IEEE Press (2013)
- [29] Duffield, E., Hagan, K.: Darkcoin: Peer-to-peer crypto currency with anonymous blockchain transactions and an improved proof-of-work system. Technical report (March 2014), <http://www.darkcoin.io/downloads/DarkcoinWhitepaper.pdf>
- [30] Buterin, V., Malahov, J., Wilson, C., Hintjens, P., Taaki, A., et al.: Dark Wallet, <https://darkwallet.unsystem.net>
- [31] van der Laan, W.J.: Implement coinjoin in wallet. GitHub Issue #3226 of official Bitcoin repository, <https://github.com/bitcoin/bitcoin/issues/3226>
- [32] Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. In: USENIX Security 2004, pp. 21–37. USENIX Assoc. (2004)
- [33] Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM 24(2), 84–90 (1981)
- [34] Syta, E., Johnson, A., Corrigan-Gibbs, H., Weng, S.C., Wolinsky, D., Ford, B.: Security analysis of accountable anonymous group communication in Dissent. ACM Transactions on Information and System Security (TISSEC) (to appear)
- [35] Brickell, J., Shmatikov, V.: Efficient anonymity-preserving data collection. In: SIGKDD 2006, pp. 76–85. ACM (2006)

- [36] Transaction fees. Bitcoin Wiki, https://en.bitcoin.it/w/index.php?title=Transaction_fees&oldid=45501 (revision as of March 28, 2014)
- [37] Haeberlen, A., Kouznetsov, P., Druschel, P.: PeerReview: Practical accountability for distributed systems. In: SOSP 2007, pp. 175–188. ACM (2007)
- [38] Karame, G.O., Androulaki, E., Capkun, S.: Double-spending fast payments in Bitcoin. In: CCS 2012, pp. 906–917. ACM (2012)
- [39] Ruffing, T., Moreno-Sanchez, P., Kate, A.: CoinShuffle: Practical decentralized coin mixing for Bitcoin. Full version of this paper and prototype implementation, <http://crypsys.mmci.uni-saarland.de/projects/CoinShuffle>
- [40] Certicom Research: Sec 1: Elliptic curve cryptography, <http://www.secg.org/download/aid-780/sec1-v2.pdf>
- [41] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. In: OSDI 2002, pp. 255–270. USENIX (December 2002)
- [42] Thread on Bitcoin Forum, <https://bitcointalk.org/index.php?topic=175156>
- [43] Maxwell, G.: Post on Bitcoin Forum (2013), <https://bitcointalk.org/index.php?topic=279249.msg2984051#msg2984051>

A High-Level Comparison with Dissent

CoinShuffle has been inspired by the shuffling phase of the Dissent protocol [26, 34], which adds robustness to a data collection protocol due to Brickell and Shmatikov [35]. The fact that CoinShuffle is crafted specially to be used on top of the Bitcoin protocol allows us to apply several optimizations. In the following we describe the two most important improvements as compared to Dissent. We assume the reader to be familiar with the Dissent protocol [26, 34].

First, observe that in Dissent, the shuffling phase must hide participants’ inputs even in case of failure, i.e., even if the blame phase is reached. For that, Dissent needs N additional inner layers of encryption, because each message is additionally encrypted with the encryption keys of all participants. This makes it possible to introduce an additional step after the shuffling: Participants check first if the shuffling was performed correctly, and they reveal their inner decryption keys only if the check succeeds. In contrast, hiding the plaintexts is not necessary in a failed run of CoinShuffle, because the plaintexts are only fresh Bitcoin addresses that are discarded when the protocol fails; it is not a problem to create new addresses in the subsequent run of the protocol. As a result, the additional inner layers of encryption are not necessary in CoinShuffle.

Second, the description of the shuffling phase of Dissent specifies that every participant sends and receives a vector of N ciphertexts. In CoinShuffle, every participant i receives a vector of only $i-1$ ciphertexts and sends only i ciphertexts to the next participant. The communication overhead is thereby further reduced, and fewer encryption and decryption operations are necessary as compared to Dissent. We conjecture that this improvement is also applicable to the shuffling phase of Dissent, but we leave a formal treatment for future work.