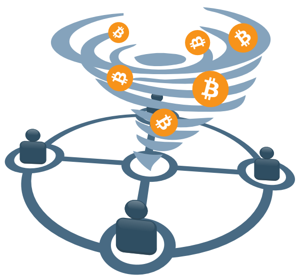
CoinShuffle: Practical Decentralized Bitcoin Mixing

The decentralized currency network Bitcoin is emerging as a potential new way for 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, and a few Bitcoin transaction deanonymization attacks have been reported so far.

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 relationship between the user and the coin. This is typically done via mixing services, e.g., Bitmixer.io and BitcoinBath. However, users have to *fully trust* the mixing service: First, anonymity is heavily restricted, because it is only guaranteed towards external observers. The mixing service itself can still determine the owner of a Bitcoin. Second and even more important, the users have to transfer their coins to the mixing service who could just *steal* them and never given them back. Moreover, the mixing services typically charge a fee.

CoinShuffle

CoinShuffle is a completely decentralized Bitcoin mixing protocol that addresses these two issues. CoinShuffle enjoys several advantages over its predecessor Bitcoin mixing protocols. CoinShuffle's features are



- **Decentralization:** The core protocol of CoinShuffle *does not require any* (trusted, accountable or untrusted) third party.
- **Anonymity:** *Nobody* can determine the relation between the users' old and new addresses after the mixing.
- **Security against thefts:** By relying on the principle of CoinJoin, CoinShuffle ensures that no money can be stolen during mixing.
- **Compatibility:** CoinShuffle runs on top of the existing Bitcoin protocol and requires no changes to the Bitcoin protocol itself.
- **Robustness against denial-of-service attacks:** While similar solutions based on CoinJoin can be disrupted by a single malicious participant, CoinShuffle ensures that disruptive participants can be identified and excluded from the protocol.
- **No additional fees**: Except for the standard Bitcoin transaction fee that is necessary to perform the mixing, CoinShuffle requires no additional mixing fee.
- **Performace:** CoinShuffle introduces only a small communication overhead for its users and minimizes the computation and communication overhead for the rest of the Bitcoin system.
- **Simple cryptography:** CoinShuffle requires only basic cryptographic primitives such as digital signatures and standard public-key encrpytion.

Our research paper contains the full details of the CoinShuffle protocol.

Comparison with Different Approaches

Zerocoin / Zerocash

Zerocoin and the upcoming optimized Zerocash are great because they provide "built-in anonymity" by using quite novel cryptography, e.g. ZK-SNARKS. However, Zerocash requires a trusted party for the initial setup of public parameters. Most importantly, all these approaches are currencies on their own. Zerocoin and Zerocash are not compatible with Bitcoin, they need their own protocol extensions and blockchain. CoinShuffle works directly on top of Bitcoin, without changing the Bitcoin protocol or forking the chain.

Publications

CoinShuffle: Practical Decentrelized Coin Mixing for Bitcoin
Tim Ruffing, Pedro Moreno-Sanchez, Aniket Kate
19th European Symposium on Research in Computer Security (**ESORICS '14**). LNCS 8713 (to appear). Springer, 2014.

[PDF] (technical report, revision 2014-08-14), **[BibTeX]**

News

`[09/09/2014]` We presented CoinShuffle at **ESORICS '14**. The slides are available.

`[06/08/2014]` CoinShuffle is discussed in the Bitcoin Forum and on the Bitcoin Development Mailing List.

`[18/07/2014]` We presented CoinShuffle at **HotPETs '14**, where some security concerns were raised during the presentation. We would like to clarify here that those concerns were *wrong*, and emerged as we could not present the complete protocol in the allowed 12 minutes of talk time. We clarified those concerns in a discussion right after the presentation. The supposed replay attack is not possible, as CoinShuffle uses IND-CCA encryption and each participant in the shuffle checks for duplicate outputs after decrypting all the received ciphertexts.

Mixcoin

The main innovation of Mixcoin is accountability for mixing servers (*mixes*): If the mix server steals coins, the user obtains a cryptographic proof of this theft and can hold the mix accountable. This is done in public: Everybody can verify this proof and the mix will hopefully lose its reputation, and nobody will use the mix in the future. The mix can still steal money but it will be caught and probably has to go out of business.

In contrast, the advantage of CoinShuffle is that it prevents stealing of coins in the first place instead of providing accountability only after the theft. Additionally, a centralized mixing server is not at all necessary in CoinShuffle. However, CoinShuffle requires the participants of a single mixing round to be online at the same time.

CoinSwap

Most important, the participants know which coins belong to which user in CoinSwap, so the anonymity is limited. Furthermore, CoinSwap needs at least 4 transactions and the corresponding fees. CoinShuffle needs only one transaction. However, CoinSwap is essentially a two party protocol, so it requires less interaction and coordination. The original CoinSwap thread provides a detailed comparison to CoinJoin, which provides the basis for CoinShuffle.

Prototocol Overview

To illustrate the protocol, consider a small example with four participants Alice, Bob, Charlie, and Dave. The participants have exactly 1 BTC at each of their respective addresses A, B, C, D. Assume the participants already know that they would like to run the protocol with each other and they know the addresses of each other. (Finding other participants can be done via a P2P protocol, for example.)

The participants create fresh addresses A', B', C', D' but do not show them to each other. The goal of CoinJoin-based mixing is to create one single mixing transaction with input addresses A, B, C, D and output addresses A', B', C', D' to hide the relation between the coins and their owners. (See the forum thread about CoinJoin for more details on this idea). However, if we would stick to that particular order A', B', C', D' of output addresses, everybody would learn that A belongs to A', B belongs to B', and so on. So we need to shuffle the list of output addresses to make sure that the linkage of input and output addresses remains hidden. But just shuffling the output addresses in the created transaction does not suffice: For example, if everybody just announced his output addresses during the protocol in plain, i.e., Alice announces A', everybody would learn that A' belongs to Alice. So we have to make sure that the messages that are sent during the protocol do not break the anonymity. CoinShuffle solves exactly this problem.

A successful run of the protocol looks as follows: (Note that the description is simplified; the full details are in the paper.)

All messages are signed using the private signing key belonging to the input address of the sender of the message. We omit the signatures in the following description to simplify the presentation.

Phase 1: Key exchange

Each participant (except for Alice) creates an key pair of a public key encryption scheme, consisting of a public encryption key and a private decryption key. We call the public encryption keys ek_B, ek_C, and ek_D. Each participant announces his public encryption key, signed with the signature key corresponding to his input address.

Phase 2: Shuffling

Once everybody knows the public encryption key each other, the shuffling can start:

Alice encrypts her output address A' with all the encryption keys, in a layered manner. That is, Alice encrypts A' first for Dave, obtaining enc(ek_D, A'). Then this ciphertext is encrypted for Charlie, obtaining enc(ek_C, enc(ek_D, A')) and so on for Dave. This resulting message is sent to Bob:
Alice ⟶ Bob: enc(ek_B, enc(ek_C, enc(ek_D, A')))

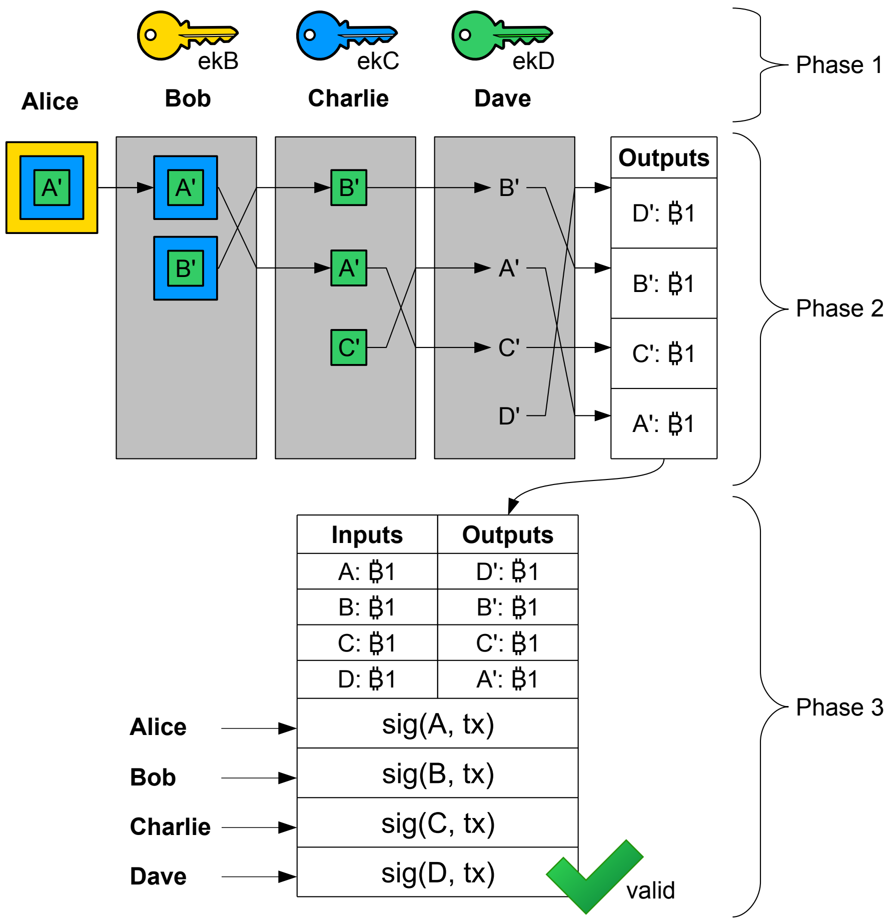
Bob gets the message, decrypts it, obtaining enc(ek_C, enc(ek_D, A')).
He also creates a nested encryption of his address, obtaining enc(ek_C, enc(ek_D, B')).
Now Bob has a list two ciphertexts, containing A' and B'. Bob shuffles this list randomly, i.e., either exchanges the two entries or leave them. Say we are in the case that they are exchanged. Bob sends the shuffled list to Charlie:
Bob ⟶ Charlie: enc(ek_C, enc(ek_D, B')) ; enc(ek_C, enc(ek_D, A'))

Charlie does the same: He decrypts the two entries in the list, adds his own entry and shuffles the list:
Charlie ⟶ Dave: enc(ek_D, B') ; enc(ek_D, C') ; enc(ek_D, A')

Dave does the same again: He decrypts all entries, obtaining B', C', A'. He adds his own address D' and
shuffles the list. The resulting shuffled list is sent to everybody:
Dave ⟶ everybody: D', B', C', A'

Phase 3: Creating the transaction

Every participant receives the list of output addresses and can verify that his output address is indeed there. If yes, he signs the transaction. If, e.g., Bob sees that his address is not there, he would lose his coin by performing the transaction, so he obviously does not want to sign. (This is the main idea of CoinJoin.)
In the case that Bob's address is not there, somebody must have cheated during the run of the protocol. Bob complains and the participants enter an additional phase to find out who cheated. CoinShuffle makes sure that this "blame phase" always exposes at least one cheating participant (and none can be accused falsely). This cheating participant can then be excluded from a subsequent run of the protocol: Say Alice cheated. Then Bob, Charlie and Dave can run the protocol again without Alice.
The key point is that in phase 2, only the participant that performed the shuffling knows the relation between the messages in the list that he received and the messages in the list that he sent.
For example, only Charlie knows that he left the message containing *B'* in the first position, because the encryption ensures that nobody can relate enc(ek_C, enc(ek_D, *B'*)) and enc(ek_D, *B'*). But even Charlie does not know that this was the message with Bob's address.
In the end, all addresses of honest participants are shuffled as explained in my previous posting. *Nobody knows the permutation.* (A detailed argument can be found in the paper.)



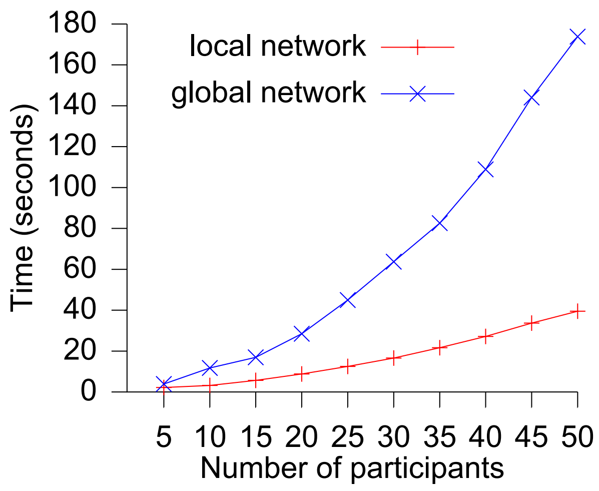
An overview over a successful run of CoinShuffle. [svg] [png]

Note: The protocol works even if participants do not have exactly 1 BTC at their input addresses. It suffices that they have at least 1 BTC. In that case, they create a transaction that sends 1 BTC to each of the shuffled output addresses and the remaining coins to a change addresses that the users announce in the beginning. This can be done as for normal Bitcoin transactions. (This idea is also described in CoinJoin already.)

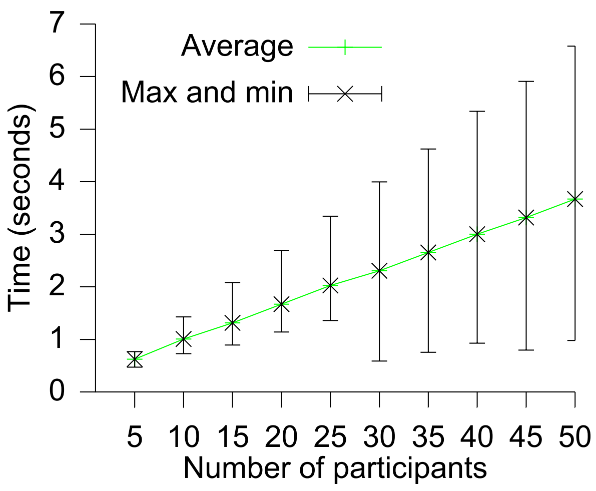
Performance

Using our proof-of-concept implementation, a protocol run with 50 participants takes roughly 3 minutes now in a setting with reasonable latency.

In more detail, we tested our implementation in Emulab, 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 ordering were placed in one LAN while participants with an even index were placed in the second LAN. Thus every message in the shuffling phase had to traverse the whole network.



Overall execution time



Computation time per node

We have run the protocol with different numbers of participants, ranging from 5 to 50. The left figure shows the overall time needed to create a Bitcoin transaction in a run without misbehaving participants. In the local network setting, 50 participants need approximately 40 seconds to run CoinShuffle, while in the global network setting, slightly less than 3 minutes are necessary to complete the protocol. The right figure shows the overhead of the computation carried out by every participant on average.

Proof-of-concept Implementation

As a proof of concept, we have re-used Dissent code to implement a prototype of CoinShuffle.

WARNING: The implementation is not at all developed with security in mind. It has been written purely to evaluate the feasibility and performance of CoinShuffle and is INSECURE.
DO NOT USE THIS WITH YOUR COINS.

A careful, full implementation is required before CoinShuffle can be used in practice. Still, you can download our prototype if you are interested.