CoinJoin in the Wild

(Full Version)

Dominic Deuber and Dominique Schröder

Friedrich-Alexander University Erlangen-Nürnberg
firstname.lastname@fau.de

Abstract. CoinJoin is the predominant means to enhance privacy in non-private cryptocurrencies, such as Bitcoin. The basic idea of CoinJoin is to create transactions that combine equal-valued coins of multiple users. This mixing of coins aims to prevent linkage of the users' transactional in- and outputs. The cryptocurrency Dash employs a built-in CoinJoin service and, therefore, is ideal for empirically studying CoinJoin. This paper presents the first empirical analysis of Dash, which reveals that over 40% of all private transactions can be de-anonymized depending on underlying assumptions. The main issue of these attacks is the coin-aggregation problem, i.e. the need to combine outputs of several CoinJoin transactions. The coin aggregation problem is not specific to Dash and affects other cryptocurrencies as empirical evidence in Bitcoin suggests. We show that the logical solution to the problem, namely CoinJoin transactions with non-fixed arbitrary values, suffers from other privacy weaknesses. We propose a novel mixing algorithm to mitigate the need for coin aggregation without introducing additional privacy vulnerabilities. In contrast to prior mixing algorithms, our approach removes the need for fixed values by dynamically creating equal-valued CoinJoin transactions. The mixing algorithm is not specific to Dash, and integration into other cryptocurrencies, especially into Bitcoin, is possible.

Keywords: anonymous transactions, linking heuristics, de-anonymization, mixing

Table of Contents

1 Introduction

More and more it seems as if cryptocurrencies have come to stay and are not mere hype. The most widely used cryptocurrency Bitcoin [bit] is often perceived to provide anonymity. However, Bitcoin is not anonymous as it is possible to link addresses that belong to the same user [RS13,AKR$^+$13,MPJ$^+$13]. The goal of mixing protocols is the prevention of these linkage attacks. *CoinJoin* [Max13] is the most widely used protocol. It combines inputs and outputs from multiple users and creates a random permutation that hides the correlation between input and output addresses and, thus, between users. Even though Monero [mon] and Zcash [zca] are two cryptocurrencies that achieve privacy by design, it is crucial to study and improve *CoinJoin* for two reasons. First, Bitcoin is still the most commonly used cryptocurrency, especially in the dark web [Eur], and supports *CoinJoin* to improve privacy without requiring to swap coins to a more privacy-preserving currency. Second, Monero and Zcash are already banned in South Korea [Ike], and there is a risk that other jurisdictions will follow. If privacy-preserving currencies are banned, *CoinJoin* on top of Bitcoin is among the only possibilities for people in the corresponding jurisdictions to add privacy to their cryptocurrency activities.

The cryptocurrency Dash [dasa] employs a built-in *CoinJoin* mechanism. Dash has not been researched before, although Dash has a market capitalization of over four billion USD, is the second-largest cryptocurrency with built-in privacy features, and the third most established privacy coin for transactions in the dark web [Eur]. By the beginning of 2021 Dash's blockchain accounted for approximately 1.4 million blocks including over 31 million transactions.

1.1 Empirical Analysis of Anonymity

We present the first empirical analysis of anonymity in Dash that combines new and existing attacks to evaluate Dash's anonymity level and gain insights on *CoinJoin* and its privacy. We introduce a novel attack that we call Backlink attack. In essence, Backlink attack carefully combines *multi-input heuristic* [RS13,AKR$^+$13,MPJ$^+$13] and a newly developed heuristic to find address clusters.

We put forward the DC attack, which is a modification of the *cluster-intersection attack* according to Goldfeder *et al.* [GKRN18]. The DC attack revealed a fundamental problem with *CoinJoin*, namely the coin aggregation problem. As Dash uses fixed values in their *CoinJoin* transactions, users generally need to aggregate coins of several *CoinJoin* transactions that fuel the attack. To ascertain whether the coin aggregation problem is also present in other cryptocurrencies, we analyze the impact of our attacks for Bitcoin.
Results: It was found that 15.1% of non-mixing transactions that spend private coins are linkable by the Backlink attack. In terms of address clusters, applying the newly developed heuristic reduces the number of clusters by almost two-third compared to only applying the *multi-input heuristic*. By applying the DC attack, we were able to link over 40% of Dash's private transactions depending on the underlying assumptions of the attack.

In Bitcoin, around 23% of all transactions which spend outputs from a *CoinJoin* transactions contain backlinks. In addition, more than one-tenth of all transactions do so from *CoinJoin* transactions spend from at least two different *CoinJoin* transactions, which indicates that coin aggregation is also a problem in Bitcoin.

1.2 Cookie Monster Mixing

Our analysis suggests that the privacy issues in Dash result from the fact that Dash only supports equal-valued mixing with fixed values and allows users to combine their coins in a way that might de-anonymize them. We analyze arbitrary-value *CoinJoin* as proposed by Maurer *et al.* [MNF17] and show that it has other privacy weaknesses, which is why it is not a suitable way to solve the coin aggregation problem. To remove the issue of only fixed values being mixed without introducing additional privacy vulnerabilities, we propose a novel mixing algorithm that we call Cookie Monster Mixing. The algorithm is inspired by the cookie monster problem [BK15] and removes the need to split and combine coins before and after mixing. Thus, the information that multiple coins of different mixing transactions belong together is no longer present *on-chain*. As a consequence, *cluster-intersection attacks* without additional *off-chain* information are no longer possible. We have formalized the problem as an integer quadratic problem and propose an efficient greedy algorithm to solve it. A prototype implementation reinforces the practical efficiency. Through experimentation, we have validated that the greedy algorithm is nearly optimal.

1.3 Responsible Disclosure

We reported our findings to the Dash Core Group, one of the organizations working for the Dash network, and declared our willingness to support the implementation of the suggested countermeasures. With Dash Core Release 0.16.0.1 [dasc], Dash has implemented some of our suggested countermeasures to improve privacy.

1.4 Related Work

A major concern of *CoinJoin* is that the users need to trust an external mixing service that creates the transaction. Alternative approaches to mitigating this weakness have been proposed, such as *CoinShuffle* [RMK14] or its more efficient successor *CoinShuffle++* [RMK17]. For Ethereum, a trustless tumbler Möbius has been presented that achieves mixing through a smart contract based on ring signatures and stealth addresses [MM18].

While *CoinJoin* is a mixing service that can be used as an extension of traditional cryptocurrencies, new privacy-preserving cryptocurrencies have also evolved, spearheaded by Monero [mon] and Zcash [zca]. Monero is based on the CryptoNote protocol [VS13] and mainly uses ring-confidential transactions [NML16] to achieve privacy. Conversely, Zcash is based on the Zerocash protocol [BCG$^+$14] and mainly uses zero-knowledge, succinct, non-interactive arguments of knowledge to achieve privacy. The anonymity of both cryptocurrencies has since then been subject to analyses [KFTS17,MSH$^+$18,KYMM18,Que17].

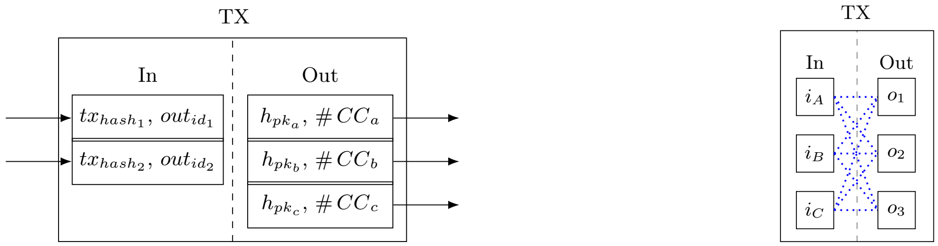
Goldfeder *et al.* [GKRN18] showed that *CoinJoin* transactions in Bitcoin are vulnerable to the so-called *cluster-intersection attack*. Kalodner *et al.* [KGC$^+$17] experimentally validated the applicability of the *cluster-intersection attack* to Dash on simulated transactions. In contrast, we apply the attack to the entire Dash blockchain data. To do so, however, we needed to refine it, as there are several underlying assumptions to take into account.

The major services for *CoinJoin* in Bitcoin are Wasabi Wallet [was], Samourai Wallet [sam] and JoinMarket [joi]. All distinguish between *pre*- and *post*-mixing. However, Wasabi and Samourai require fixed output values and thus might benefit from the flexibility Cookie Monster Mixing provides in building their *CoinJoin* transactions. JoinMarket allows for flexible output values, albeit in a different setting. Its protocol distinguishes between takers and makers where the taker pays the makers to participate in the mixing.

2 Preliminaries

In this section, we briefly explain concepts necessary to understand our attacks and countermeasures. We introduce transactions, the *multi-input heuristic* and *CoinJoin* followed by a high-level description of the *cluster-intersection attack*.

2.1 Transaction



(a) standard - with two in- and three outputs

(b) *CoinJoin*

Fig. 1: Transaction

A transaction consists of a list of inputs and outputs. In simple terms, an output comprises an amount of a given cryptocurrency CC and the hash h_{pk} of a public key pk, which is also called an address. Inputs are references to outputs of previous transactions. A transaction with two inputs and three outputs is depicted in Figure 1a. The two inputs refer to the outputs at indices out_{id_1} and out_{id_2} of transactions with hashes tx_{hash_1} and tx_{hash_2} respectively. Each output o_i for $i \in [a, b, c]$ of the transaction specifies an address h_{pk_i} and an amount $\#CC_i$ of the cryptocurrency. To spend an output o_i of this transaction in a succeeding transaction, a public key pk must be provided whose hash equals h_{pk_i} and a signature that verifies for pk. It is common for a transaction to have multiple in- and outputs, as the input value needs to be spent completely. For example, if the inputs amount to 5 CC but the user only wants to spend 4 CC to h_{pk_a} and h_{pk_b}, they will create an output o_c to send back the remaining 1 CC to an address they control (h_{pk_c}), which is also called *change* (address). Outputs that can be referenced by a transaction, but have not yet been, are called *unspent-transaction outputs*.

2.2 *Multi-Input Heuristic*

If a transaction has multiple inputs, the following address-linking heuristic can be applied.

Heuristic 21 (*multi-input heuristic* [RS13,AKR+13,MPJ+13]) *All addresses referred to in the inputs of a transaction are controlled by the same entity.*

The reason is that the computation of the signature of each input requires the knowledge of the secret key. Other heuristics take advantage of the fact that coins can only be spent in their entirety with the spending user usually sending back the remaining amount of the cryptocurrency to a *change* address they control [RS13,AKR+13,MPJ+13]. Linking addresses results in sets of addresses, so-called address clusters, which are likely controlled by the same entity.

2.3 *CoinJoin*

The basic idea of *CoinJoin* [Max13] is special *CoinJoin* transactions, which combine the in- and outputs of multiple users. An example of such a transaction is shown in Figure 1b. The transaction has three inputs and three outputs from three different users A, B and C. Here, we assume that all in- and outputs have the same value. By merely examining the transaction it is not possible to determine which input i_x for $x \in [A, B, C]$ belongs to which output o_y for $y \in [1, 2, 3]$. As the inputs are controlled by three different users, the *multi-input heuristic* (Heuristic 21) cannot be applied.

2.4 *Cluster-Intersection Attack*

Goldfeder *et al.* [GKRN18] showed that *CoinJoin* transactions in Bitcoin are vulnerable to the so-called *cluster-intersection attack*, which works as follows. For each output of a *CoinJoin* transaction, its anonymity set is determined by inspecting the inputs of the transaction as, ideally, each input could be the origin of each output. The anonymity set contains all possible address clusters that might be the output's origin. Additional information may likely reveal that the same entity controls certain outputs of different *CoinJoin* transactions. In that case, the corresponding anonymity sets can be intersected, *i.e.* address clusters that are present in all sets can be identified. If there is only one address cluster in the intersection, this cluster might be the origin of those outputs. Additional information revealing that the same entity controls certain outputs of different *CoinJoin* transactions can, for instance, be a single transaction spending such outputs. Then, the information follows from the *multi-input heuristic* (Heuristic 21) and thus is *on-chain* information. Furthermore, it is also possible that the payment recipient can derive the information *off-chain*, as seen in the following example. Imagine a merchant receives two payments from the same customer, and each of the payments is the output of a different *CoinJoin* transaction. If the anonymity sets of both outputs only have a single address cluster in common, the merchant can assume that this cluster belongs to the customer.

3 Dash

In this section, we introduce Dash and explain how it addresses privacy in its PrivateSend feature as a necessary prerequisite for our attacks in Section 4.

3.1 Overview

The cryptocurrency Dash, having forked from Bitcoin in 2014 [dasb], follows the same basic structure: the decentralized transaction ledger is maintained in a peer-to-peer network that uses a consensus mechanism to agree on new transactions. The transactions are organized in blocks and the ledger is often called blockchain; the nodes in the consensus mechanism are called *miners*. They are rewarded for their participation through block rewards, *i.e.*, newly generated units of the cryptocurrency and transaction fees. Dash differs from Bitcoin mainly by implementing a native *CoinJoin* feature, PrivateSend, and a feature that reduces the time it takes until a transaction can be considered final, InstantSend. Both features are achieved by so-called *masternodes*, which are special nodes participating in the Dash network. In contrast to *miners*, *masternodes* do not directly participate in the consensus mechanism but mainly provide PrivateSend and InstantSend as a service. They are rewarded for their services with fees. Additionally, they also receive parts of the block rewards in so-called *CoinbaseTXs*. InstantSend solves the problem of confirmation time that is present in Bitcoin. To do so, a quorum of *masternodes* locks the inputs of a proposed transaction, which leads to competing transactions being rejected [dase]. We do not consider InstantSend in the rest of the paper, as we are concentrating on privacy.

3.2 PrivateSend

PrivateSend is a service provided by *masternodes* to prevent the linkage of addresses from different transaction outputs potentially belonging to the same entity. Put in simple terms, PrivateSend implements *CoinJoin* (see Section 2.3). There are several services that support the process of finding other users to group with in order to build a *CoinJoin* transaction. In an ideal scenario, only these services learn the input-output mapping, *i.e.*, the mapping of inputs to corresponding outputs.

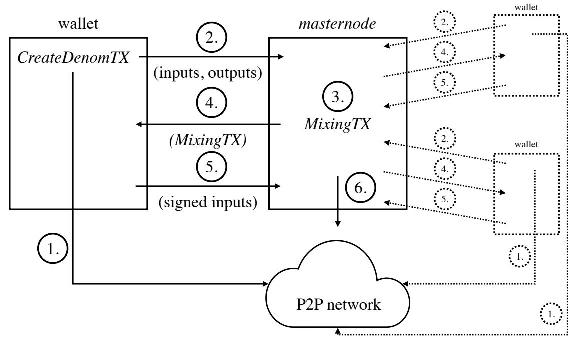


Fig. 2: PrivateSend mixing procedure

The mixing process of Dash is depicted in Figure 2 and works as follows: a user's wallet splits the value of some *unspent-transaction outputs* in a *CreateDenomTX*, and sends it to the network 1. This step is a necessary prerequisite, as mixing in Dash requires equal and fixed values. Next, the wallet reports a mixing request to a randomly selected *masternode* 2. This request includes certain *unspent-transaction outputs* of the *CreateDenomTX* as inputs and equally as many outputs with addresses that the wallet controls. If enough other users (dashed lines) also reported their request to the *masternode*, it builds a *MixingTX* 3, consisting of all of the users' input-output pairs. At this point, the *masternode* reports the *MixingTX* back to each user's wallet 4, such that it can sign the inputs. Before doing so, the wallet ensures that the outputs initially reported in its request are contained in the list of outputs of the *MixingTX*. This check is crucial in guaranteeing that the *masternode* cannot steal any coins by replacing outputs with its own. If each wallet only signs so long as the check is successful, the *masternode* cannot redirect money to their addresses since the sum of the input values must exactly match the sum of the output values in *MixingTXs*. The reason is that the fees required for mixing are decoupled from the *MixingTXs* and therefore omitted for the sake of simplicity. After each wallet has signed their inputs and sent the signatures to the *masternode* 5, the *masternode* can send the *MixingTX* to the network 6. Each wallet then has *private unspent-transaction outputs*, which can be used as inputs for further mixing rounds or spent in *PrivateSendTXs*, the final transaction type used in PrivateSend. A *PrivateSendTX* is a transaction, whereby the wallet implementation ensures that it only spends *private unspent-transaction outputs* from *MixingTXs*.

4 Empirical Anonymity Analysis

In this section, we analyze the anonymity provided by *CoinJoin* in the context of Dash and Bitcoin. For the analysis of Dash, we ran a Dash full node, version 0.16.1.1 [dasc] and build an analysis pipeline using BlockSci [KML+20] with version 0.5.0. First, we retrieved the raw blockchain data up to December 31, 2020, which corresponds to a chain of 1 397 530 blocks. Then we detected the type of transactions that are relevant for our attacks. In the backlink attack, we linked address clusters based on the *multi-input heuristic* (see Heuristic 21) and a new clustering heuristic. Finally, we refined the *cluster-intersection attack* by adding false-positive rejection mechanisms and addressing uncertainty about its underlying assumptions to make the attack applicable to Dash (DC attack). The differences of our analysis of Bitcoin are stated in Appendix A.

4.1 Transaction Type Detection

We ran a transaction type detection algorithm for the identification of relevant transactions for PrivateSend. This algorithm processes the data retrieved from our full node, and it takes advantage of the fact that the *mixing denominations* in Dash are of the form 1.00001×10^k for $k \in [-3, \ldots, 1]$. Due to this structure, it seems unlikely that it would not be a *mixing denomination* if we were to encounter such a value. As a consequence, our detection mechanism should produce few to no false positives. By design of our detection mechanism, a transaction can only have one type, *i.e.*, there is no ambiguity. We consider each transaction that does not match any of the following types to be an *OtherTX*. ***MixingTX*** is a transaction with equally many inputs and outputs, all with the same denomination. This is due to the fact that the fee is decoupled from mixing. Thus, there is exactly one output for each input. We additionally require that there are at least three inputs as at least three participants are required for mixing (see Dash's whitepaper [DD]). ***CreateDenomTX*** is a transaction that is not a *MixingTX* if there are at least two outputs, while one output needs to have one of the *mixing denominations*. Furthermore, we allow at most two *non-mixing-denominated* outputs since one of them might be the *change* output and thus of arbitrary value. The other might be a special output required to pay mixing fees. ***PrivateSendTX*** is a transaction that is not a *MixingTX* if it has more than one input and all the inputs are *mixing denominations*. However, we only consider it a *PrivateSendTX* if it has exactly one output since a *PrivateSendTX* does not allow *change* [dasg].

In Figure 3 the transactions are listed by their type, where the total number of transactions was 31 563 841. Only 0.4% (110 846) of all transactions are *PrivateSendTXs*.

Bitcoin In Bitcoin, we found that out of 493 118 000 transactions, 1 767 452 (0.4%) were *CoinJoinTXs* (the counterpart of Dash's *MixingTXs*). For the transactions entering into and spending from *CoinJoinTXs*, we detected 5 865 534 (1.2%) *PreCoinJoinTXs* and 7 228 843 (1.5%) *PostCoinJoinTXs*.

CoinbaseTXs	4.4%
PrivateSendTXs	0.4%
MixingTXs	10.1%
CreateDenomTXs	1.5%
OtherTXs	83.6%

Fig. 3: Dash transaction types

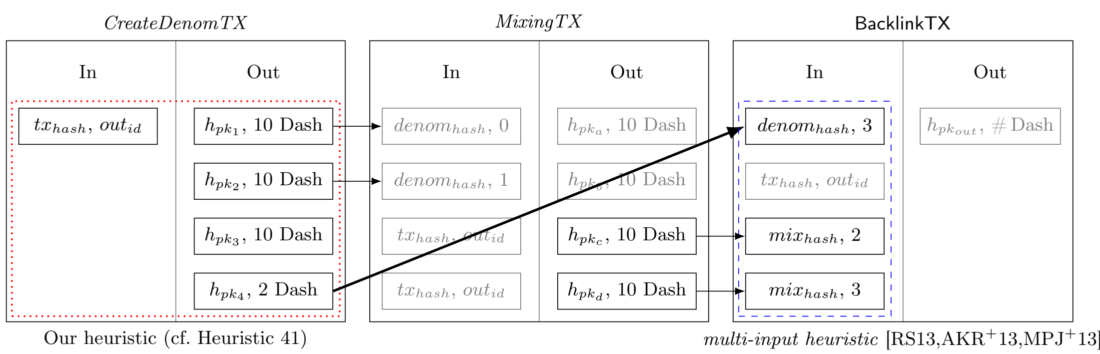


Fig. 4: Backlink analysis

4.2 Backlink Attack

We introduce the Backlink attack, which directly links addresses occurring in the output of a *MixingTX*, *i.e.*, linking them to output addresses of a *CreateDenomTX*. There are transactions in Dash that spend outputs of *MixingTXs* and at the same time outputs of *CreateDenomTXs*. We call such a transaction a BacklinkTX and the output of the *CreateDenomTX* a backlink. Such a transaction is shown in Figure 4. The transaction's first input is a reference to the fourth output of the *CreateDenomTX*, which is the backlink.[1] Thus, as a direct result of the *multi-input heuristic* (see Heuristic 21), the addresses of the *MixingTX*, h_{pk_c} and h_{pk_d}, are linkable, as under the assumptions of the *multi-input heuristic* there is a link to h_{pk_4}, which is an output address of the *CreateDenomTX*.

However, the linkable addresses can be further linked as all input and all output addresses of a *CreateDenomTX* are most likely controlled by the same entity. The reason for this is that *CreateDenomTXs* are generated by a user's wallet. This leads to the following address clustering heuristic:

Heuristic 41 *All in- and output addresses of a* CreateDenomTX *are controlled by the same entity.*

[1] Note that the reference says [$denom_{hash}$, 3] to refer to the fourth output as indexing starts with 0.

As a result, h_{pk_c} and h_{pk_d} of the BacklinkTX in Figure 4 can not only be linked to h_{pk_4} (*multi-input heuristic*, dashed line) but also to h_{pk_1} to h_{pk_3} and to the address corresponding to the output of tx_{hash} at index out_{id} (Heuristic 41, dotted line). Note that reasonable clustering results are only achieved by combining both heuristics. Applying the *multi-input heuristic* would allow linking to the backlink. However, without Heuristic 41, the backlink address would, in general, be in a single cluster and not reveal any additional information about the user's transaction history before mixing.

To detect backlinks, we do the following. We first iterate over all transactions. Then, we check each transaction as to whether it has inputs referencing *MixingTXs* as well as inputs referencing *CreateDenomTXs*. To identify the corresponding clusters, we add our heuristic to the clustering module of BlockSci, which already implements the *multi-input heuristic*.

We found that out of the 174 834 transactions that are not *MixingTXs* but spend mixing outputs, 26 402 (15.1%) have backlinks. In terms of addresses from the outputs of *MixingTXs*, we found that out of 6 833 911 addresses, 836 230 (12.2%) are linkable. Applying only the *multi-input heuristic* resulted in 23 580 clusters. We reduced that number by almost two thirds by additionally applying our Heuristic 41, which resulted in only 7 920 clusters.

Bitcoin The attack slightly differs in Bitcoin as there are no explicit *CreateDenomTXs*. Instead of *CreateDenomTXs*, we consider the *PreCoinJoinTXs*. Thus, a BacklinkTX in Bitcoin is a *PostCoinJoinTX* with at least one input from a *PreCoinJoinTX*. We found that out of 7 228 843 *PostCoinJoinTXs*, 1 674 070 (23.2%) have backlinks. This shows that backlinks are also present and even more problematic in Bitcoin.

4.3 DC Attack

We introduce the DC attack as a modification of the *cluster-intersection attack* (Section 2.4). First, we give a high-level description of the *cluster-intersection attack* in the context of Dash, followed by a discussion of which modifications are necessary to apply the attack. Finally, we present the Dash *cluster-intersection attack* (DC attack).

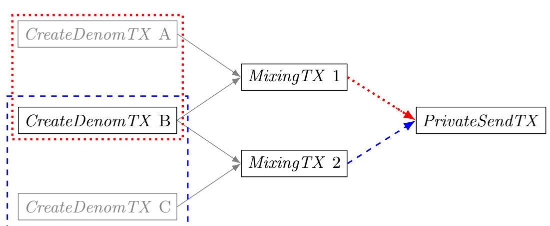


Fig. 5: Cluster intersection in Dash

An overview of the *cluster-intersection attack* in Dash is depicted in Figure 5. The *PrivateSendTX* has inputs from both *MixingTX* 1 and *MixingTX* 2. If we trace back the inputs, the set of *CreateDenomTXs* that can be reached from the *MixingTX* 1 input contains *CreateDenomTX* A and *CreateDenomTX* B. Likewise, the set reachable from the *MixingTX* 2 input contains *CreateDenomTX* B and *CreateDenomTX* C. If we intersect the sets, *CreateDenomTX* B is the only *CreateDenomTX* remaining.

Modifications For actual transaction data, we do not know how many rounds users have been mixing for and whether the mixing originated from a single linkable address cluster [AK18,GKRN18]. Thus, we need to modify the attack. To compensate for not knowing how many rounds of mixing the inputs of a *PrivateSendTX* took, we consider a range of mixing rounds. To address the assumption that all inputs originated from a single cluster, we developed a two-fold approach.

First, we add a mechanism to reject a cluster if there is a subset of inputs that would result in another cluster. This cluster can be seen as an *alternative* explanation for the subset of inputs. The minimum size of the subset is adjustable via a parameter (*alt*). If, for example, ground-truth data indicated that clusters containing 90% of the inputs are common, then an *alt* value of 80% could be suitable for blockchain analysts to safeguard the evidential value of their findings. In that case, the analysts would reject a cluster, if 80% or more of the inputs could be explained by another cluster.

Second, we add a mechanism to detect some obvious false positives that are based on the following observation. A cluster cannot have more Dash spent in its *PrivateSendTXs* than have been created in its *CreateDenomTXs*. This is why we compute a *mix balance* for each cluster as follows. Firstly, we sum the value of all outputs of *CreateDenomTXs* that are spent in *MixingTXs*. Then, we subtract the value of all inputs of a transaction that is not a *MixingTX* but is spending from *MixingTXs*. In simple terms, we determine the value that has been input into mixing and subtract the value that has been spent after mixing. Suppose the attack now suggests linking two clusters, such that the sum of their *mix balances* is negative. In that case, we know that we either encountered a false positive or that our clustering of pre-mix addresses was incomplete. In either case, we must reject the result because the two cases cannot be distinguished without ground-truth data.

This results in the Dash *cluster-intersection attack* (DC attack), which is a modified version of the algorithm proposed by Goldfeder *et al.* [GKRN18]. The algorithm is stated in Algorithm 1. The `LEN` method always returns the number of elements of the passed argument. The algorithm's input is a *PrivateSendTX* *ptx* and the parameter *alt* as described above. We set the starting value for the number of rounds r to 2, since 2 is the minimum number of mixing rounds required in Dash. The maximal possible number of rounds changed at the beginning of 2019 from 8 to 16 with protocol version 0.13.0.0 [dasd]. Thus, to prevent the algorithm from detecting obvious false positives, we determine for every *PrivateSendTX*, the maximal possible number of rounds max_r in `DETERMINE_MAX_ROUNDS` as follows. We retrieve the block in which the transaction occurred. If the year extracted from the block's timestamp is greater than 2018, we set max_r to 16 and 8 otherwise. Next, the algorithm iterates over all rounds. In every round, for each input, all clusters that are attainable within r rounds of mixing are determined (`EXTRACT_CLUSTS`). Then, the intersection *intersec* of all found cluster sets is computed. If there is exactly one cluster in the intersection, we perform an additional check, `CORR_REJ`. This checks whether there is a subset of the inputs whose size is greater than or equal to *alt* of `LEN`(*ptx.inps*), which would lead to a different cluster than *intersec*. If this is not the case, *candidate* is set to the cluster in *intersec* (`GET`). The loop is left regardless of `CORR_REJ`. Finally, we check that *candidate* is not a false positive according to the *mix balance*, which is performed by `BALANCE_CONF` and works as explained above.

Algorithm 1 DC attack

```
procedure DC_ATTACK(ptx, alt)
    candidate = None
    r = 2
    max_r = DETERMINE_MAX_ROUNDS(ptx)
    while r ≤ max_r do
        clusts = ∅
        for inp ∈ ptx.inps do
            clusts[inp] = EXTRACT_CLUSTS(r, inp)
        intersec = ⋂_{inp ∈ ptx.inps} clusts[inp]
        if LEN(intersec) == 1 then
            if CORR_REJ(intersec, alt, clusts) then
                candidate = GET(intersec)
            break
    if BALANCE_CONF(candidate) then
        return candidate
    return None
```

The results of the DC attack are shown in Figure 6. Setting parameter *alt* to 100% corresponds to complete certainty in the assumption that all inputs resulted from one linkable cluster. In that case, no alternative explanation is taken into account and over 40% of the *PrivateSendTXs* are linkable. In the case of *alt* being equal to 0%, all results would be rejected by definition.

alt (%)	10	20	30	40	50	60	70	80	90	100
linkable (%)	0.2	0.7	2.2	4.1	5.6	11.9	17.5	22.4	32.1	43.8

Fig. 6: DC attack linkable *PrivateSendTXs* for parameter *alt* ranging from 10% to 100%

Bitcoin Goldfeder *et al.* already demonstrated the applicability of the *cluster-intersection attack* in Bitcoin [GKRN18]. Thus, the crucial question in this work is whether there is *on-chain* information fueling the attack. The special vulnerability to cluster intersection in Dash results from the fact that users need to aggregate value in *PrivateSendTXs*. Thus, a *PrivateSendTXs* has several inputs from different *MixingTXs* in general that can be seen as such *on-chain* information.

To determine whether such *on-chain* information is also present in Bitcoin, we did the following. We checked for every *PostCoinJoinTXs* whether there are inputs from at least two different *CoinJoinTXs*. If this was the case, there was *on-chain* information as it is possible to intersect the anonymity sets of the different *CoinJoinTXs*. We found that out of the 7 228 843 *PostCoinJoinTXs*, 919 532 (12.7%) have inputs from at least two different *CoinJoinTXs*. This indicates that the coin aggregation problem is also present in Bitcoin.

5 Enhancing Privacy of Mixing

We show how to enhance the privacy of mixing and discuss direct countermeasures to mitigate the vulnerability to the Backlink attack. After discussing why fundamental changes to Dash seem unavoidable to prevent the DC attack, we propose a new mixing algorithm that removes the vulnerability to the *cluster-intersection attack*. This algorithm is of independent interest as it is not specific to Dash.

5.1 Preventing backlinks

The anonymity problems that come with backlinks are approachable within the design of Dash and Bitcoin. First, not all outputs of a *CreateDenomTX* must be input to *MixingTXs*. There may be *change*, such as discussed above in the example of Figure 4. Additionally, Dash allows for *CoinControl*, *i.e.* letting users in their wallet manually select inputs of a transaction [dasf]. While this is a useful feature, in the case of a user creating a BacklinkTX, we recommend explicitly warning them as backlinks remove the anonymity gained by mixing. A user's wallet should strictly separate any coins from *CreateDenomTXs* and those originating from a *MixingTX*. This idea is incorporated in the Bitcoin fungibility framework ZeroLink [zer] that distinguishes a *pre-mix* and a *post-mix* wallet. With version 0.16.0.1 [dasc] Dash improved its user interface following our recommendations after we disclosed our findings to them.

5.2 Cookie Monster Mixing

The vulnerability to the *cluster-intersection attack* results from the coin aggregation problem, that is the need to combine coins of different mixes. In Dash, this is a consequence of restricting the mixing to specific values. The logical solution would be to allow arbitrary values. However, arbitrary-value mixing suffers from privacy weaknesses caused by value analysis as discussed in Appendix B. Thus, we propose a new mixing algorithm, Cookie Monster Mixing. The basic principle behind Cookie Monster Mixing is to create a *MixingTX* where there are at least k outputs with the same value and k is the anonymity level the transaction should provide. This is related to the cookie monster problem [BK15]. Given a set of jars filled with various numbers of cookies, the cookie monster wants to eat all the cookies. However, the cookie monster has to proceed in rounds, select a subset of jars, and eat the same number of cookies from each jar in this subset. The goal is to eat the cookies in as few rounds as possible. In contrast, the objective in Cookie Monster Mixing is to maximize the number of cookies for a fixed number of rounds under constraints instead of minimizing rounds.

In Cookie Monster Mixing, a mixing service provider takes the role of the cookie monster, while the jars are inputs with a specific value of the cryptocurrency to be mixed. In Dash, the *masternodes* act as mixing service providers. Deviating from the cookie monster problem, let the number of rounds r be fixed. In each round $j \in \{1, \ldots, r\}$, the mixing service provider may choose a target value t_j and a subset of the input values from which t_j is subtracted. The subtracted value is added to the output set, while the objective is to maximize the total output value. Intuitively that relates to maximizing the total value of anonymous coins. Additionally, there are two constraints. First, the size of the subset of inputs selected per round needs to be at least k. Second, each selected input needs to have a value at least as large as t_j. Together the constraints ensure that the outputs determined via t_j have at least an anonymity set size of k as they have the same value and thus might have originated from any of the inputs selected in that round.

Integer Quadratic Problem The problem can be formulated as the following integer quadratic problem (IQP).

Constants
- $v_1, \ldots, v_n$: non-negative integers (values of n inputs)
- k: positive integer (minimum number of values to select per round)

Variables
- $x_1, \ldots, x_n$: $0-1$ vectors of length r (where x_i denotes the rounds in which value v_i has been selected)
- t: non-negative integer vector of length r (target values to be subtracted)

$$maximise \sum_{i=1}^{n} \langle x_i, t \rangle \tag{1}$$

$$s.t.\ \langle x_i, t \rangle \leq v_i, \text{for each } i \in \{1, \ldots, n\} \tag{2}$$

$$\sum_{i=1}^{n} x_{i,j} \geq k, \text{for each } j \in \{1, \ldots, r\} \tag{3}$$

The Objective (1) is to maximize the total anonymized value, while Constraints (2) and (3) ensure that it is actually possible to subtract t_j from the selected inputs and that at least k inputs are selected per round j respectively.

Greedy Algorithm We propose a greedy algorithm that approximates the integer quadratic problem. It is stated in Algorithm 2. The input to the algorithm is a list of input values *in_vals*, as well as k and r as defined above. *in_vals* can be obtained from the multiset of inputs I by replacing each input with its value. `LEN` returns the number of elements in a list and `EXTRACT_K_LARGEST` extracts the k^{th} largest element of a list. The algorithm returns *target_vals*, which is a list of values referring to the target values t_j of the integer quadratic problem.

Algorithm 2 Greedy solver

```
procedure SOLVER(in_vals, k, r)
    if LEN(in_vals) < k || r == 0 then
        return ∅
    target_vals = ∅
    tmp_vals = ∅
    r_val = EXTRACT_K_LARGEST(k, in_vals)
    target_vals.add(r_val)
    for val ∈ in_vals do
        if val > r_val then
            tmp_vals.add(val − r_val)
        else if val < r_val then
            tmp_vals.add(val)
    return target_vals.concat(SOLVER(
                tmp_vals, k, r − 1))
```

As long as the abort criterion (*i.e.*, `LEN`(*in_vals*) $< k \mid\mid r == 0$) is not fulfilled, the algorithm extracts the k^{th} largest element of *in_vals*, which is assigned to r_{val}. Since this element can be seen as the target value of the greedy algorithm in that round, it is added to *target_vals*. Then, the input values are updated as follows. In case a value is greater than r_{val}, their difference is added to *tmp_vals*. Otherwise, if the value is smaller than r_{val}, the value itself is added to *tmp_val*. If they are equal, the value is omitted. This behavior corresponds to inherently selecting all possible inputs per round in terms of the integer quadratic problem. The algorithm runs in polynomial time. There are at most r recursive calls and the runtime of each call is mainly determined by the time it takes to extract the k^{th} largest element of a list. If this is implemented by sorting, the algorithm runs in $\mathcal{O}(r \cdot n \log n)$.

Our greedy algorithm is not optimal, which can be seen by the following example. Let *in_vals* $= [2, 2, 1], k = 2$ and $r = 2$. In the first round, the algorithm extracts 2 as the second largest element and adds it to *target_vals*, while 1 is added to *tmp_vals*. In the recursive call, $\emptyset$ is returned, as the length *in_vals* is 1 and thus smaller than k which satisfies the abort criteria. Consequently, the output value achieved in terms of Objective (1) is only 4, as *target_vals* = [2] and the greedy algorithm inherently selects all possible inputs per round, which are the first two of *in_vals* in the first round. The optimum 5, however, is achieved by setting $a_1 = a_2 = x_{1,1} = x_{2,1} = x_{3,1} = x_{1,2} = x_{2,2}$ to 1 and $x_{3,2}$ to 0.

Evaluation of Algorithm 2 To measure the quality of our greedy algorithm in terms of maximizing Objective (1), we evaluated it against the optimal solution. Therefore we modelled the integer quadratic problem as given by Objective (1) and Constraints (2) and (3) in IBM's Optimization Programming Language (OPL) and used the mixed integer optimizer from IBM LOG CPLEX Optimization Studio V12.10.0 [ibm].

For the values of the inputs (*in_vals*), we first considered all clusters of *CreateDenomTXs* retrieved by applying the *multi-input heuristic* and Heuristic 41 as discussed in Section 4.2. For each cluster, we summed up the values of all outputs that were referenced by *MixingTXs*. This results in a distribution of Dash that users were mixing in the past. It is therefore better suited for evaluation than purely random values. We varied both, r and the number of inputs that we chose randomly from the distribution. We set k to 3 following the Dash whitepaper, which suggests at least three participants per mixing round [DD].The results are shown in Figure 7a averaged over 100 runs indicating that Algorithm 2 is nearly as good as the optimal solution. The average wall-clock time of the solver is reported in Figure 7b. In contrast, Algorithm 2 took less than a second for each choice of parameters. Therefore, particularly for multiple inputs and rounds, using a solver is infeasible, which is why Algorithm 2 should be used instead.

	r		
$\lvert in\_vals \rvert$	3	4	5
5	98.8%	98.5%	98.1%
10	98.2%	98.4%	98.3%
15	97.0%	97.2%	98.3%
20	96.1%	95.8%	97.7%

(a) Avg. performance of Algorithm 2 w.r.t Objective (1)

	r		
$\lvert in\_vals \rvert$	3	4	5
5	0.02	0.02	0.28
10	0.03	0.27	7.80
15	0.07	7.51	68.71
20	0.15	5.98	180.14

(b) Avg. wall-clock time of optimizer in minutes

Fig. 7: Evaluation of Algorithm 2 against optimizer

Using Cookie Monster Mixing removes the need for splitting and aggregating coins *before* and *after* mixing. Thus, the *on-chain* information that multiple coins of different mixes belong together is no longer present, preventing the DC attack. To prevent *cluster-intersection attacks* from *off-chain* information as well, Cookie Monster Mixing needs to be combined with privacy-aware wallets and browsers.

Acknowledgments

We would like to thank Christoph Egger, Paul Gahman, Viktoria Ronge and Kyle Soska for their helpful comments as well as all the reviewers of this work for their constructive feedback. Work was supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under reference number 442893093 and as part of the Research and Training Group 2475 "Cybercrime and Forensic Computing" (grant number 393541319/ GRK2475/1-2019).

References

AK18. Antti-Kaikkonen. Evaluating the privacy of privatesend, 2018.

AKR+13. Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In Sadeghi [Sad13], pages 34–51.

BCG+14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

bit. Bitcoin.

BK15. Leigh Marie Braswell and Tanya Khovanova. The cookie monster problem. *The Mathematics of Various Entertaining Subjects: Research in Recreational Math*, page 231, 2015.

bloa. Blocksci source code.

Blob. Blockstream. Bitcoin explorer source code.

btc. Bitcoin release v0.20.0.

dasa. Dash.

dasb. Dash core repository.

dasc. Dash core repository release history.

dasd. Dash core version 0.13.0.0.

dase. Dash developer documentation.

dasf. Dash wallet documentation - coin control.

dasg. Dash wallet documentation - privatesend and instantsend.

DD. Evan Duffield and Daniel Diaz. Dash: A payments-focused cryptocurrency.

Eur. European Cybercrime Center (EC3). Internet organised crime threat assessment 2020.

GKRN18. Steven Goldfeder, Harry Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *Proceedings on Privacy Enhancing Technologies*, 2018(4):179 – 199, 2018.

ibm. Ibm log cplex optimization studio v.12.10.0.

Ike. Scott Ikeda. South korea's new crypto aml law bans trading of "privacy coins" (monero, zcash).

joi. Joinmarket.

KFTS17. Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 153–173. Springer, Heidelberg, September 2017.

KGC+17. Harry Kalodner, Steven Goldfeder, Alishah Chator, Malte Möser, and Arvind Narayanan. Blocksci: Design and applications of a blockchain analysis platform. *arXiv preprint arXiv:1709.02489*, 2017.

KML+20. Harry A. Kalodner, Malte Möser, Kevin Lee, Steven Goldfeder, Martin Plattner, Alishah Chator, and Arvind Narayanan. BlockSci: Design and applications of a blockchain analysis platform. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2721–2738. USENIX Association, August 2020.

KYMM18. George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. An empirical analysis of anonymity in zcash. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 463–477. USENIX Association, August 2018.

Max13. Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world, 2013.

MM18. Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *PoPETs*, 2018(2):105–121, April 2018.

MNF17. F. K. Maurer, T. Neudecker, and M. Florian. Anonymous coinjoin transactions with arbitrary values. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 522–529, Aug 2017.

mon. Monero.

MPJ+13. 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.

MSH+18. Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. An empirical analysis of traceability in the monero blockchain. *PoPETs*, 2018(3):143–163, July 2018.

NML16. Shen Noether, Adam Mackenzie, and The Lab. Ring confidential transactions. *Ledger*, 1:1–18, 12 2016.

Que17. Jeffrey Quesnelle. On the linkability of Zcash transactions. *arXiv e-prints*, page arXiv:1712.01210, Dec 2017.

RMK14. Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical decentralized coin mixing for bitcoin. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 345–364. Springer, Heidelberg, September 2014.
RMK17. Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. P2P mixing and unlinkable bitcoin transactions. In *NDSS 2017*. The Internet Society, February / March 2017.
RS13. Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In Sadeghi [Sad13], pages 6–24.
Sad13. Ahmad-Reza Sadeghi, editor. *FC 2013*, volume 7859 of *LNCS*. Springer, Heidelberg, April 2013.
sam. Samourai wallet.
VS13. Nicolas Van Saberhagen. Cryptonote v 2.0, 2013.
was. Wasabi wallet.
zca. Zcash.
zer. Zerolink - the bitcoin fungibility framework.

A Differences in the Analysis in Bitcoin

This section highlights the general differences in the analysis of Bitcoin and Dash. For Bitcoin, we ran a full node, version 0.20.0 [btc] and used BlockSci [KML+20] with version 0.7.0. We retrieved the raw blockchain data corresponding to a chain of 612 793 blocks with 493 118 000 transactions.

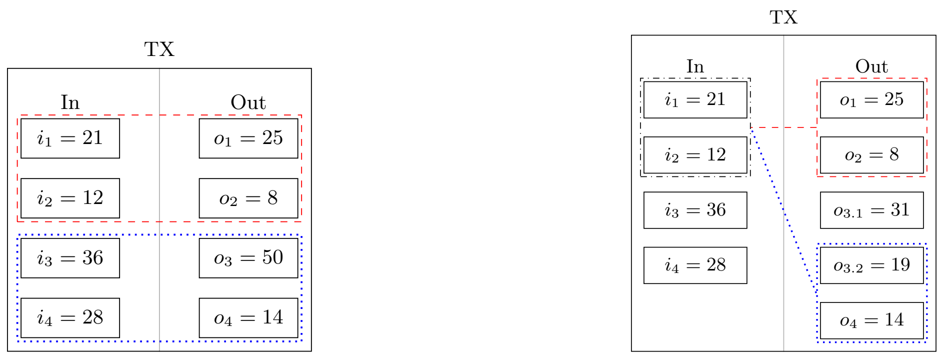
The main difference occurs in the transaction type detection as Bitcoin neither knows *CreateDenomTXs* nor *PrivateSendTXs*. Thus, besides the *CoinJoinTXs*, we also consider *PreCoinJoinTXs* and *PostCoinJoinTXs*. A *PreCoinJoinTX* is any transaction that has at least one output being referenced by a *CoinJoinTX*. Likewise, a *PostCoinJoinTX* is any transaction referencing at least one output of a *CoinJoinTX*. We further exclude all *CoinJoinTXs* from *PreCoinJoinTXs* and *PostCoinJoinTXs*. In comparison with Dash, the *CoinJoinTXs* would correspond to *MixingTXs*, the *PreCoinJoinTXs* to the *CreateDenomTXs*, and the *PostCoinJoinTXs* to the *PrivateSendTXs*. Detecting *PostCoinJoinTXs* and *PreCoinJoinTXs* is straightforward once *CoinJoinTXs* are detected. However, the detection of *CoinJoinTXs* is difficult as there are multiple different *CoinJoin* services in Bitcoin (*e.g.* [joi,was,sam]) that neither require the number of inputs and outputs to be the same nor restrict their inputs to specific denominations as is the case in Dash. BlockSci already implements a *CoinJoin* detection mechanism which, however, is tailored to JoinMarket [joi] transactions and therefore does not recognize the transactions of other *CoinJoin* services such as Wasabi Wallet [was] or Samurai Wallet [sam]. For this reason, we adapted the *CoinJoin* detection mechanism of the open-source Blockstream Bitcoin explorer [Blob] as it is capable of detecting *CoinJoinTXs* of several services. However, we also adopted the elements of BlockSci's algorithm [bloa] that were not specific to JoinMarket.

Algorithm 3 *CoinJoin* detection

```
procedure COJO_DETECTION(tx)
    if LEN(tx.inps) < 2 || LEN(tx.outs) < 3 then
        return False
    target = MIN(MAX(LEN(tx.outs)/2, 2), 5)
    found = False
    for out ∈ tx.outs do
        if out.val == 546 || out.val == 2730 then
            return False
        if OCC(val, tx.outs) >= target) then
            found = True
    if OCC_MOST(tx.outs) < OCC_UNIQ(tx.outs) then
        return False
    return found
```

Our algorithm is stated in Algorithm 3. The algorithm returns *True* if the provided transaction *tx* is (most likely) a *CoinJoin* transaction. The `LEN` method returns the number of elements of the passed argument, `MIN` and `MAX` work as expected. `OCC` computes the number of occurrences of the value *val* in the provided *outputs*. `OCC_MOST` returns the number of occurrences of the value that occurs the most while `OCC_UNIQ` returns the number of unique output values. The first thing the algorithm does is check whether the transaction has at least two inputs and three outputs. The reason for this is that mixing requires at least two participants. At least one participant generally receives some change, which is why there is always at least one additional output aside from the two mixed ones. Next, a *target* between two and five is computed. This is used to check whether there are at least *target* many outputs with the same value, where *target* corresponds to half the number of outputs but is kept between two and five as done by Blockstream [Blob]. As suggested by BlockSci, a transaction with so-called *dust* outputs is unlikely a *CoinJoin* transaction, which is why output values should not be equal to 546 or 2 730 [bloa]. These values are the smallest possible output values allowed by Bitcoin Core depending on the version. The last check is to prevent false positives as there needs to be at least as many equal-valued outputs as there are unique ones. The reason is that in a *CoinJoinTX* unique outputs should only be change outputs.

B Limitations to Arbitrary-Value Mixing



(a) original

(b) with output splitting

Fig. 8: *CoinJoin* transaction

We proposed Cookie Monster Mixing (see Section 5.2) as arbitrary-value mixing is not a suitable solution for the coin aggregation problem due to privacy weaknesses based on value analysis. When mixing coins with an arbitrary value, outputs can usually be linked to the corresponding inputs by inspecting the values, as discussed by Maurer *et al.* [MNF17]. Considering the *CoinJoin* transaction in Figure 8a taken from Maurer *et al.* [MNF17], the transaction can only consist of the two sub-transactions (dotted line and dashed line), such that it is possible to link inputs i_1 and i_2 to outputs o_1 and o_2, as well as i_3 and i_4 to o_3 and o_4, respectively. To prevent this linkage, Maurer *et al.* [MNF17] propose output-splitting algorithms. Given two transactions, their basic splitting algorithm works by calculating the difference between the sums of the corresponding output lists. Next, one of the outputs of the list with the larger sum is split to produce this difference [MNF17]. Thus, multiple input-output relations are possible. Applied to the two sub-transactions in Figure 8a, the algorithm results in the transaction depicted in Figure 8b. Output o_3 in Figure 8a has been split in $o_{3.1}$ and $o_{3.2}$ such that i_1 and i_2 belong to either o_1 and o_2 or $o_{3.2}$ and o_4. The reason is that the sum of the values of i_1 and i_2 equals 33, as do the sums of o_1 and o_2 as well as $o_{3.2}$ and o_4.

However, even if output-splitting results in multiple potential input-output relations, it is still possible to determine the actual input-output relation by inspecting the values. In Figure 8b, i_1 and i_2 are far more likely to result in o_1 and o_2 than in $o_{3.1}$ and o_4. The reason is that under the assumption that one of the outputs is *change*, input i_2 would not have been required as i_1 is larger than 19 ($o_{3.1}$) and 14 (o_4).

As their basic output-splitting algorithm does not affect the input linkability, that is i_1 and i_2 as well as i_3 and i_4 are linkable, Maurer *et al.* [MNF17] propose a version of the algorithm that implements input shuffling. Instead of using the difference between the sums of the corresponding output lists, the sum of a random subset of inputs is used to split the outputs. Thereby, the number of inputs is a parameter of the algorithm. In terms of Figure 8a, the input shuffling algorithm might employ the sum of i_1, i_2 and i_4. While this seems to be an improvement over the basic algorithm, it is especially dangerous if the inputs are linkable by heuristics. Intuitively, the gained privacy relies on an ambiguity on the input side, which is introduced by using the sum of a random subset in output splitting. However, if it is known which inputs belong together, there is no gain in privacy at all.