Anonymous CoinJoin Transactions with Arbitrary Values

Felix Konstantin Maurer
Communication and Distributed Systems
RWTH Aachen University
felix.maurer@comsys.rwth-aachen.de

Till Neudecker
Institute of Telematics
Karlsruhe Institute of Technology
till.neudecker@kit.edu

Martin Florian
Institute of Telematics
Karlsruhe Institute of Technology
florian@kit.edu

***Abstract*—Bitcoin, the arguably most popular cryptocurrency to date, allows users to perform transactions using freely chosen pseudonymous addresses. Previous research, however, suggests that these pseudonyms can easily be linked, implying a lower level of privacy than originally expected. To obfuscate the links between pseudonyms, different mixing methods have been proposed. One of the first approaches is the CoinJoin concept, where multiple users merge their transactions into one larger transaction. In theory, CoinJoin can be used to mix and transact bitcoins simultaneously, in one step. Yet, it is expected that differing bitcoin amounts would allow an attacker to derive the original single transactions. Solutions based on CoinJoin therefore prescribe the use of fixed bitcoin amounts and cannot be used to perform arbitrary transactions.**

In this paper, we define a model for CoinJoin transactions and metrics that allow conclusions about the provided anonymity. We generate and analyze CoinJoin transactions and show that with differing, representative amounts they generally do not provide any significant anonymity gains. As a solution to this problem, we present an output splitting approach that introduces sufficient ambiguity to effectively prevent linking in CoinJoin transactions. Furthermore, we discuss how this approach could be used in Bitcoin today.

1. Introduction

In the past years, Bitcoin [1] received a growing interest from academics and commercial entities [2]. Using a peer-to-peer (P2P) network, it requires no trust in a central authority and makes successful double spending unlikely, as long as most computation power belongs to honest peers. Even if an attacker amasses the majority of the computation power, no funds can be stolen without access to the wallets. Public keys are used as unlimited expendable pseudonyms, suggesting that it can be used anonymously.

However, research shows that the pseudonyms can be linked and if one pseudonym of a user leaks, the others can be inferred [3], [4], [5], [6], [7]. This is even worse since the record of all transactions that are used to perform the linking are public. Analyzing transactions is therefore possible for anybody and might endanger, e.g., human rights organizations in oppressive regimes. If Bitcoin is to be used as currency, this problem must be solved.

Many approaches to increase the anonymity in a process called mixing were proposed [8], [9], [10], [11], [12]. However, they generally have to be performed in addition to normal transactions, leading to extra delay and fees. *CoinJoin* is an concept that could be used to perform mixing and transactions in a single step. As Bitcoin does not require transactions to be issued by a single person, multiple users can combine their transactions into a larger CoinJoin transaction. Several implementations exist [12], [13], that facilitate the creation of these CoinJoin transactions. However, until recently they all required the users to transfer the same amount of bitcoins, for fear that otherwise the pseudonyms might be linked using the distinct coin values.

In this paper, we investigate the linkability of pseudonyms in CoinJoin transactions. **Our main contributions are:** 1) We show that naive CoinJoin transactions with arbitrary, commonly used amounts allow linking of pseudonyms. 2) Based on a concept called *knapsack mixing*[14], [15], we present novel output splitting algorithms that hinder linking. 3) We show that the computational difficulty of analyzing knapsack mixed CoinJoin transactions alone might provide enough anonymity, and that even if the analysis is performed, only a small number of pseudonyms can reliably be linked.

As our approach allows arbitrary values in CoinJoin transaction, Bitcoin mixing and coin transfer can now be performed in a single transaction. In contrast to previous approaches this reduces delay, fees, and allows to perform mixing on each transaction. Finally, we describe how our knapsack mixing algorithm could be used to build a full P2P mixing protocol, using existing techniques [16], [17].

2. Fundamentals

In this section, we provide a short overview of relevant aspects of Bitcoin, and briefly explain the heuristics used in Bitcoin deanonymization and the CoinJoin concept.

In Bitcoin, each transaction consists of input coins that are spent, and output coins that are created. Each input is a reference to an output, that was previously created in another transaction, recording that this output is now spent. This way, double spending is prevented. Through this referencing, the transactions and coins form a large graph. An example excerpt of this graph is shown in Figure 1.

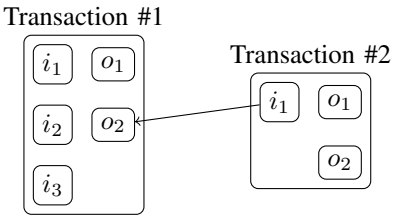

Figure 1: Example of two Bitcoin transactions. Input i_1 of transaction 2 references output o_2 of transaction 1, recording that it is spent.

The transactions are collected into blocks, by Bitcoin users called miners. Additionally to the transactions, each block contains the checksum of the previous block and a nonce. The nonce is randomly chosen so that the checksum of the block is lower than a specific target value in a process called mining. As each block references its precursor, all blocks form a linked list, the so called blockchain.

Each coin has a short program attached, called script. Whoever can provide a valid input to the script, can spent the coin. Usually, the program checks whether the spender possesses a specific private key, by including the corresponding public key. The public key, called Bitcoin address, can be thought of as the owner of the coin.

Once created, coins have a fixed value and can't be split or merged freely. This means, that to send a certain amount of bitcoins, possibly more than one coin must be spend. In turn, if coins of a higher value are spent then what should be transferred, the change must be returned in an extra output. Therefore, more than one coin as input and at least two coins as outputs is the norm. This led to heuristics that are used to analyze the transaction graph [3], [4]. Even though new Bitcoin addresses can be generated for each new coin, they can now be linked and attributed to the same user.

The first heuristic **H1** states that two addresses that appear in two inputs of the same transaction must belong to the same user. This mostly holds, as usually only a single user creates the transaction. Two additional heuristics try to determine the output coin that belongs to the payer, e.g., that returns the change to the payer. Heuristic **H2** applies if exactly one address of the outputs has never been previously used in other transactions. In this case, it can be assumed that this is a freshly generated address for receiving the change. Heuristic **H3** applies if exactly one output is smaller than any input. If the change output was larger than any input, the input would not need to be included and the transaction size could be reduced, leading to less fees.

To obfuscate the trail of a coin that is the result of applying these heuristics, mixing techniques were developed. In general, mixing is a process where multiple coins of different users enter a system and each user receives back a coin that he originally did not posses. The mixing is successful if nobody can learn which user received which coin by examining the blockchain.

One concept to realize mixing are CoinJoin transactions. In a CoinJoin transaction, multiple users contribute inputs and outputs to a single transaction, for example by using the same mixing service. Figure 2 shows an example. Alice would like to transfer 25 bitcoins using two coins of value 21 and 12 she owns, resulting in input i_1 and i_2, output o_1 and output o_2 that contains her change. Bob would like to transfer 50 bitcoins using two coins of value 36 and 28, resulting in input i_3 and i_4, output o_3 and output o_4 that contains his change. Mapping 1 shows the sub-transactions of Alice and Bob, whereas Mapping 2 shows the complete CoinJoin transaction. Our definition of mapping will be explained in Section 4 and is not important in this context. Technically, CoinJoin transactions are no different from normal Bitcoin transactions.

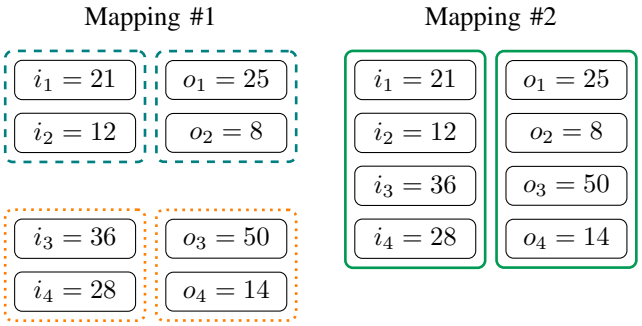

Figure 2: Example of a CoinJoin transaction with two sub-transactions. Each sub-transaction has two inputs and two outputs. There are two valid mappings, the second mapping is a derived mapping and can be created from the first, a non-derived mapping.

3. Related Work

As a context for this paper, we will discuss related work in this section. We present findings on the anonymity of Bitcoin, as well as approaches on how to improve it.

To deanonymize users, information can be collected from the blockchain itself and from the P2P network of Bitcoin clients. While information gathered from the P2P network requires an active attacker, there is already a lot of implicit information on the blockchain, which can be retrieved by anybody any time. Reid and Harrigan [3] construct the graph of transactions and use heuristic $H1$ to link addresses. Subsequent works additionally use heuristics $H2$ and/or $H3$, include information found online [4], [5], as well as global properties of the graph [6], and combine these approaches into an automated tool [7].

To mitigate these attacks, the community developed methods of *mixing* coins of multiple users. Initial approaches were centralized services, which were improved by adding accountability [8] and preventing the service from learning address connections [9]. Newer approaches are often realized as P2P mixing, based on concepts like *CoinJoin* [18] or *CoinSwap* [19]. Example include *CoinShuffle* [12], *CoinShuffle++* that uses DiceMix [16], *CoinParty* [11] and *Xim* [10]. However, all approaches require the users to mix a fixed amount of bitcoins. As users want to transfer arbitrary amounts of coins, mixing and transferring coins have to

be separate steps. Therefore, current mixing approaches all require extra transactions to prepare coins for mixing and to mix the coins, which introduces additional delay and fees.

New cryptocurrencies solve these problems by design [20], [21], [22], [23], [24]. They can hide sender, receiver and sometimes even the amount of a transaction while still allowing anyone to verify its correctness. Some approaches include similar mechanisms as CoinJoin [14], [15] and propose splitting outputs to obfuscate links between coins in a scheme they call *knapsack mixing*. However, they are not backwards compatible, rely on new cryptographic methods and introduce overhead compared to Bitcoin. It is therefore unlikely that they will replace Bitcoin anytime soon.

Some of their concepts are discussed in the Bitcoin community and may be backported if compatibility allows it. One example is *Confidential Transactions*, that enable transactions with hidden amounts. *ValueShuffle* [25] is an improved version of CoinShuffle++ that uses Confidential Transactions in CoinJoin transactions. The authors highlight that this allows to perform transactions and mixing in a single step, eliminating the separate coin mixing step. However, as they rely on Confidential Transactions, their approach is currently not Bitcoin-compatible.

We argue that being able to transfer and mix coins in a single step is a crucial requirement for a mixing scheme that can be enabled and used by default for all transactions. In this paper, we analyze how output splitting algorithms that implement knapsack mixing and are compatible with Bitcoin today, can be an alternative to Confidential Transactions in mixing schemes like ValueShuffle.

4. Analysis

In this section, we analyze the anonymity provided by naive CoinJoin transactions with arbitrary amounts. In order to find the linked inputs and outputs belonging to the sub-transaction of one user, we search for a set of outputs with the same sum as a given set of inputs. This is equal to solving the subset sum problem and a special case of the knapsack problem. Saxena et al. [14] as well as Noether et al. [15] also discuss this issue and call it *knapsack mixing*. Both propose splitting the outputs into smaller parts to create a difficult instance of this problem but do not expand on the idea. We examine such output splitting in Section 5, using the model and the methods introduced in the following.

4.1. Model

For analyzing the anonymity provided by knapsack mixing, we developed a model that describes CoinJoin transactions. It omits the fees to simplify the model and the analysis. Our results will therefore be a lower bound to the complexity of analyzing real CoinJoin transactions. In practice, the analysis of CoinJoin transactions by an attacker will be even more difficult.

A CoinJoin transaction $T = (I, O, v)$ consists of inputs $I = \{i_1, \ldots, i_n\}$ and outputs $O = \{o_1, \ldots, o_m\}$. All coins $C = I \cup O$ are assigned an value $v : C \to [0..2^{64} - 1]$. As we do not consider fees, the summed value of all input coins must be equal to the summed value of the output coins $\sum_{i \in I} v(i) = \sum_{o \in O} v(o)$.

A CoinJoin transaction consists of *sub-transactions* t_k, each consisting of inputs I_k and outputs O_k. To use existing heuristics for transaction graph analysis, we need to deconstruct it into its sub-transactions. However, we do not know how many sub-transactions there are in a given CoinJoin transaction. We only know that the sum of the inputs of a sub-transaction must be equal to the sum of its outputs and that no input or output can appear in more than one sub-transaction. Therefore, there must be at least one way of partitioning all inputs and outputs, so that each subset of inputs has exactly one corresponding subset of outputs with which it forms a sub-transaction. We call this a *mapping*, as it maps inputs to outputs.

To denote the set of all possible partitions of a set S we write $\Phi(S)$. A mapping $M = (\mathcal{I}, \mathcal{O}, m)$ consists of a partitioning of the inputs $\mathcal{I} \in \Phi(I)$ and a partitioning of the outputs $\mathcal{O} \in \Phi(O)$. It furthermore consists of a bijective projection $m : \mathcal{I} \to \mathcal{O}$ such that $\forall \mathtt{I} \in \mathcal{I}, \sum_{i \in \mathtt{I}} v(i) = \sum_{o \in m(\mathtt{I})} v(o)$, i.e., the sum of one input set is equal to the sum of its related output set. We denote the set of all mappings by $\mathcal{M}$.

Each CoinJoin transaction has at least one valid mapping where $\mathcal{I} = I$ and $\mathcal{O} = O$, i.e., it consists of only one sub-transaction. Intuitively, this mapping is unlikely to the represent the original sub-transactions, except if it actually was not a CoinJoin transaction but a normal transaction. Each mapping that contains more than one sub-transaction can always be used to construct a new mapping by merging two sub-transactions. We call a mapping that can be created from another mapping a *derived mapping* and a mapping that can't be created from another mapping a *non-derived mapping*. Figure 2 depicts an example CoinJoin transaction and its two possible mappings.

Based on the definition of a mapping, we define the probability $p_{ii}(i_1, i_2)$ that two inputs belong to the same sub-transaction as the number of mappings where the input i_1 and i_2 are elements of the same subset in the input partition, divided by the total number of mappings.

$$i_1, i_2 \in I, \quad p_{II}(i_1, i_2) = \frac{|\{M = (\mathcal{I}, \mathcal{O}, m) \in \mathcal{M} \mid \exists \mathtt{I} \in \mathcal{I} : i_1, i_2 \in \mathtt{I}\}|}{|\mathcal{M}|}$$

The probability p_{OO} that two outputs belong to the same sub-transaction can be defined correspondingly.

We define the probability $p_{IO}(i, o)$ that an input coin and an output coin belong to the same transaction as follows. It is defined as the number of mappings where the input i is in a subset of the input partition that is assigned to the subset of the output partition of which o is an element, divided by the total number of mappings.

$$i \in I, o \in O, \quad p_{IO}(i, o) = \frac{|\{M = (\mathcal{I}, \mathcal{O}, m) \in \mathcal{M} \mid \exists \mathtt{I} \in \mathcal{I} : i \in \mathtt{I} \wedge o \in m(\mathtt{I})\}|}{|\mathcal{M}|}$$

Listing 1: Algorithm for finding valid partitions

```
def find_partitions(S, sums):
    results = []
    for subset in power_set(S):
        if not sum(subset) in sums:
            next
        partitions = find_partitions(S - subset, sums)
        for partition in partitions:
            results.add([subset] + partition)
    return results
```

4.2. Complexity

The complexity of finding all mappings depends on two already known problems, the problem of enumerating possible partitions of a set, and the *subset sum* problem.

The number of possible partitions of a set of size n is the Bell number B_n which is recursively defined as $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$. An upper bound for Bell numbers has been provided [26] with $B_n < (\frac{0.729n}{ln(n+1)})^n$. To find all matching input partitions and output partitions, a brute-force algorithm would have to iterate all input partitions and for each of these all output partitions. For each input-output partition combination it must furthermore compare the size of the sets. For n inputs and m outputs, this would result in an asymptotic run time of $O((\frac{n}{ln(n)})^n \times (\frac{m}{ln(m)})^m \times n \times m)$.

However, in practice the search for matching partitions can be optimized. Not all partitions are candidates for a mapping. For example, input partitions that include a set with a sum that is not a sub sum of the outputs cannot be part of a mapping. Therefore, instead of iterating all possible partitions, we perform a depth first search with abort criterion. The algorithm iterates all possible subsets and for each subset that has a sum which is a sub sum of a given set, it recursively calls itself with the remaining of the set. A pseudo code example is shown in Listing 1.

This still means that we have to at least iterate all possible subsets, e.g., the power set, which is of size 2^n for a set of size n. Therefore, we assume that the lower bound for complexity of finding all mappings in the best case where no partitions are valid is at least $O(2^n * m)$ where m is the time it takes to solve the subset sum problem for each set. As can be seen in Figure 3, the runtime of our evaluation program does indeed grow exponentially, which supports our assumption.

4.3. Evaluation

To evaluate the anonymity provided by CoinJoin transactions, we generated sample transactions and analyzed them. The coins used in the CoinJoin transactions were chosen at random following an approximated distribution of the bitcoin values used in actually observed transactions. We generated the distribution by reading all output values from the blockchain up until March 2017. The values where sorted into buckets of size 1000, to arrive at a discrete cumulative distribution function as show in Figure 4.

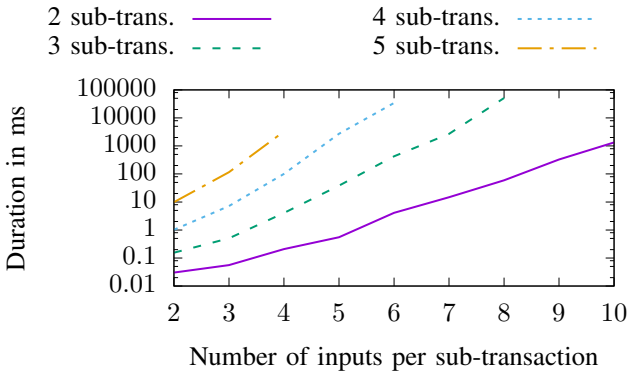

Figure 3: The processing time for calculating all mappings for a CoinJoin transaction. For each number of sub-transactions there is one line and on the x-axis the number of inputs per sub-transaction are marked.

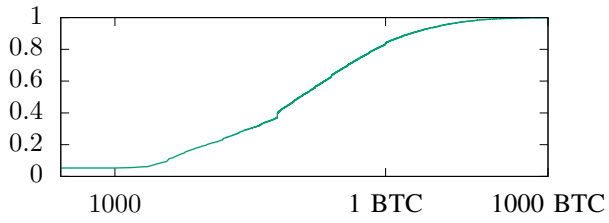

Figure 4: The discrete cumulative distribution function of the coin values in the Bitcoin blockchain.

CoinJoin transactions are generated by generating sub-transactions based on this distribution. For each sub-transaction, a parametrized number of inputs is generated and two outputs, as is in regular transactions. The first output's value is chosen randomly to be smaller than the sum of the inputs and the second output's value is the remaining difference. The sub-transactions are then merged. In Section 5 we discuss how we add mixing to it.

After generating the CoinJoin transaction, all possible mappings are calculated. Our program implements the algorithm shown in Listing 1 to search valid partitions. We collected all possible input partitions using this search algorithm filtered by the possible subset sums of the output set. Respectively we collect all possible output partitions, filtered by the subset sums of the input partitions. As the set of all possible subset sums is too large to fit in memory, we use a search algorithm that does not precompute them. To further speed up the search, we initially store all possible subset sums in a Bloom filter that we query first and only if it returns a (possibly false positive) match we do an search for the sum. Finally, we iterate the found input and output partitions, to find matching partitions, e.g., an input partition of the same size as the output partition where each set in the input partition also has a unique matching set in the output partition. These matching partitions then constitute the possible mappings of the given CoinJoin transaction.

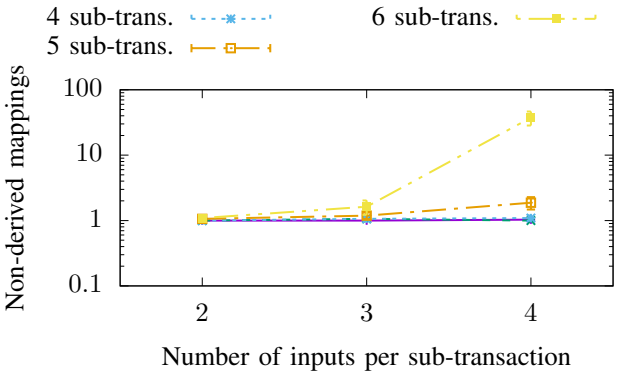

Figure 5: The number of non-derived mappings for CoinJoin transactions that have not been mixed.

As derived mappings can be constructed from the non-derived mappings, by merging sub-transactions they do not provide additional information. We assume that including derived mappings will diffuse the results, as for example the mapping where all inputs and outputs form one sub-transaction will always be contained and will always link all inputs and outputs. Therefore, in the rest of the evaluation, we only use the set of non-derived mappings.

To compensate the random coin values, we performed 32 runs for each parameter combination. The probabilities are then calculated for all input-output, and input-input pairs as defined previously. We aggregate the probabilities for each run by counting all pairs where it is 1 or 0, and by calculating the mean over all pairs. Furthermore, we aggregate the results for all runs with the same parameters by calculating the mean and the 95% confidence interval.

Figure 3 shows the processing time for calculating the mappings dependent on the number of sub-transactions and inputs per sub-transaction. As it grows exponentially with the number of sub-transactions and inputs, we only generate CoinJoin transactions up to a size of about 25 inputs.

Figure 5 shows the number of non-derived mappings for CoinJoin transactions, depending on the number of sub-transactions and the number of inputs per sub-transaction. For all number of sub-transactions with two inputs, there is generally only one non-derived mapping. With increasing numbers of inputs, the average number of non-derived mappings and the confidence interval grows.

The exponential complexity of determining possible mappings is already improving anonymity, as attackers would have to spend a high amount of computing power when analyzing the transaction graph. This approach can be used to mix any transaction and therefore used on every transaction. If implemented in the standard clients, the number of CoinJoin transactions would increase continuously. As a result, the required computation power to analyze all CoinJoin transactions could outpace the available computation power. In a sense, this would be similar to the way Bitcoin is protected against manipulation of the transaction history by requiring increasing amount of hashing power.

As each non-derived mapping is a possible candidate for the original composition of sub-transactions, a low number of non-derived mappings indicates a low level of privacy. For small unmixed CoinJoin transactions there generally exists only one non-derived mapping and therefore the original composition of sub-transactions can be unambiguously recovered. Larger CoinJoin transactions seem to have more non-derived mappings, however this is only a mean value and the confidence interval is quite large. In our results, we see many of these transactions with only one possible non-derived mapping. An attacker can then assume that this non-derived mapping represents the original composition of sub-transactions and apply heuristics $H1$, $H2$, and $H3$ to the sub-transactions. Users might therefore be deanonymized as if they were not mixing at all.

5. Knapsack mixing

As we show in the previous section, CoinJoin transactions with arbitrary values will generally only produce one non-derived mapping. This mapping likely represents the original composition of sub-transactions and can be used to apply existing heuristics for transaction graph analysis. In this section, we investigated how outputs can be split to mitigate this situation, based on the knapsack mixing idea.

5.1. Mixing algorithm

Our algorithm tries to split outputs so that different output sets can be constructed for the same input set. It achieves this by splitting a single output when two transactions are merged. The difference between the sum of the two sets is calculated and one coin of the larger set is split to produce this difference. The new coin with the value of the difference can then be added to the smaller set to produce a new set with the same sum as the originally larger set.

Figure 6 depicts the result of applying this algorithm to the example CoinJoin transaction from Section 2. The output o_3 of the original CoinJoin transaction has been split into outputs $o_{3.1}$ and $o_{3.2}$. Output $o_{3.1}$ has the value 31 which is the difference of the input sets and $o_{3.2}$ contains the remaining Bitcoins. Two non-derived mappings were created, and it is not possible to tell which represents the original sub-transactions. A generalized algorithm that handles edge cases is shown in Listing 2. This algorithm can be applied recursively while merging multiple transactions.

However, we notice, that it only decreases the linkability of input-output pairs and not the linkability of input-input pairs. The reason for this is, that our algorithm only produces new output sets with the same sum than existing output sets. Hence, only input sets with the same sum than without mixing will be found. It can be easily fixed by randomizing the input sets before applying the output splitting algorithm. The algorithm of splitting outputs must then be slightly adjusted to handle more edge cases. Listing 3 shows the adjusted `mix_transaction` function. The function to realize the sub sum in one output set stays the same.

Listing 2: Simple output splitting algorithm

```
def mix_transactions(t_a, t_b):
    if sum(t_a.inputs) == sum(t_b.inputs):
        return Transaction(
            inputs = t_a.inputs + t_b.inputs,
            outputs = t_a.outputs + t_b.outputs)
    new_inputs = t_a.inputs + t_b.inputs
    if sum(t_a.inputs) > sum(t_b.inputs):
        diff = sum(t_a.inputs) - sum(t_b.inputs)
        new_outputs = realize_subsum(t_a.inputs,
                                     diff)
        new_outputs += t_b.inputs
    if sum(t_b.inputs) > sum(t_a.inputs):
        diff = sum(t_b.inputs) - sum(t_a.inputs)
        new_outputs = realize_subsum(t_b.inputs,
                                     diff)
        new_outputs += t_a.inputs
    return Transaction(inputs = new_inputs,
                       outputs = new_outputs)

def realize_subsum(coins, subsum):
    result = []
    for coin in coins:
        if subsum == 0:
            result.add(coin)
        elif coin <= subsum:
            result.add(coin)          2 sub-trans.
            subsum -= coin            3 sub-trans.
        elif coin > subsum:
            result.add(subsum)
            result.add(coin - subsum)
            subsum = 0
    return result
```

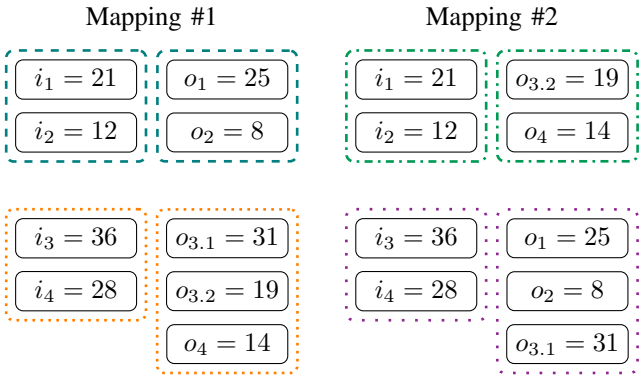

Figure 6: Example of a CoinJoin transaction with two sub-transactions after splitting the outputs. It is the same CoinJoin transaction as shown in Figure 2. The original output 3 has been splitted into two new outputs 3.1 and 3.2. Mapping 1 and mapping 2 are now both non-derived, the trivial, derived mapping that can be constructed from each has been omitted.

Listing 3: Output splitting algorithm with input shuffling

```
def mix_transactions(t_a, t_b, num_inputs):
    if sum(t_a.inputs) == sum(t_b.inputs):
        return Transaction(
            inputs = t_a.inputs + t_b.inputs,
            outputs = t_a.outputs + t_b.outputs)
    new_inputs = t_a.inputs + t_b.inputs
    random.shuffle(new_inputs)
    random_sum = sum(new_inputs[:num_inputs])
    while (random_sum >= sum(t_a.outputs)
           && random_sum >= sum(t_b.outpus)):
        random.shuffle(new_inputs)
        random_sum = sum(new_inputs[:num_inputs])
    if sum(t_a.outputs) > random_sum:
        new_outputs = realize_subsum(t_a.outputs,
                                     random_sum)
        new_outputs += t_b.outputs
    else if sum(t_b.outputs) > random_sum:
        new_outputs = realize_subsum(t_b.outputs,
                                     random_sum)
        new_outputs += t_a.outputs
    return Transaction(inputs = new_inputs,
                       outputs = new_outputs)

def realize_subsum(coins, subsum):
    ...
```

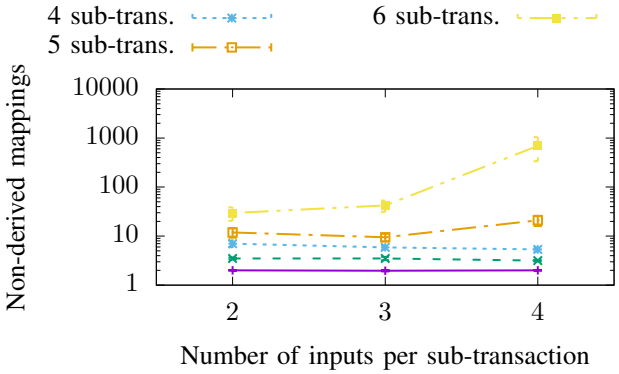

Figure 7: The number of non-derived mappings for CoinJoin transactions that have been mixed with our simple input shuffling algorithm.

5.2. Evaluation

As in Section 4.3, we generate CoinJoin transactions. This time we use our simple output mixing algorithm and the input shuffling algorithm to mix sub-transactions while they are merged. Examining the results, we notice that the average probability of inputs and outputs being linked increased. However, it is unclear what the significance of the average probability for the anonymity provided by CoinJoin transactions is. Therefore, we focused on the average number of input-input and input-output pairs that can be linked with probability $p = 1$. They provide reliable information to an attacker and could be used to apply heuristic $H1$, $H2$ or $H3$.

Figure 7 shows the number of non-derived mappings after applying our input shuffling output splitting algo-

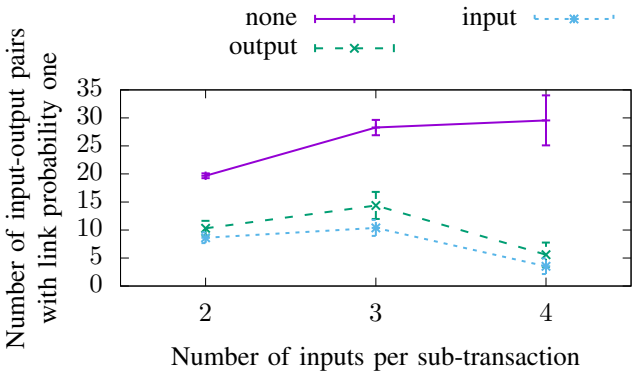

Figure 8: The number of input-output-pairs with link probability $p_{IO}(i, o) = 1$ in CoinJoin transactions with 5 sub-transactions, depending on the mixing algorithm used.

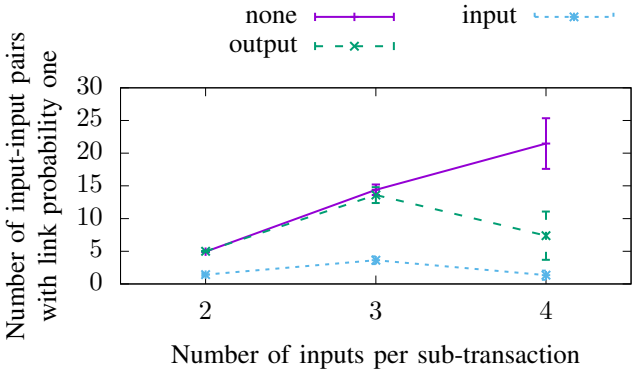

Figure 9: The number of input-input-pairs with link probability $p_{II}(i_1, i_2) = 1$ in CoinJoin transactions with 5 sub-transactions, depending on the mixing algorithm used.

rithm, equivalent to Figure 5 from Section 4.3. Even for mixed CoinJoin transactions with a low number of sub-transactions, the number of non-derived mappings is reliably greater than one. With growing number of sub-transactions, the number of non-derived mappings increases.

Figure 8 depicts the number of input-output-pairs linked with probability $p_{IO}(i, o) = 1$ in CoinJoin transactions with 5 sub-transactions, depending on the mixing algorithm used. Compared to unmixed transactions, the simple output mixing algorithm produces a significantly lower number of pair with probability $p_{IO}(i, o) = 1$ and the input shuffling does not improve this much further. Similar, in Figure 9 the number of input-input-pairs linked with probability $p_{II}(i_1, i_2) = 1$ is shown. Here, the simple output mixing algorithm does not decrease the number of input-input-pairs with probability $p_{II}(i_1, i_2) = 1$ significantly for low numbers of inputs, whereas the input shuffling does.

As our results show, the input shuffling mixing algorithm produces multiple non-derived mappings and only a small number of coin pairs with linking probability $p = 1$. Therefore, an attacker can no longer unambiguously reconstruct the sub-transactions. Heuristic $H1$ might still be used for input-input pairs that are linked with a probability $p_{II}(i_1, i_2) = 1$. However, as the full sub-transactions cannot be known, heuristics $H2$ or $H3$ can no longer be used. This means that an attacker might be able to link a small number of pseudonyms in one transaction, but not across transactions. Therefore, we have high confidence that it succeeds in generating CoinJoin transactions which provide anonymity despite using arbitrary valued coins.

6. Discussion

As shown by our evaluation, in naive CoinJoin transactions with arbitrary values the original sub-transactions can usually be recovered. Therefore, they only provide some form of *computational anonymity* given a large-enough number of inputs and outputs. Using our simple mixing algorithms, enough ambiguity can be introduced to make linking highly unlikely for most of the coin pairs, regardless of available computation power. While some can still be linked, not knowing the complete sub-transaction prevents the use of heuristic $H2$ and $H3$. Therefore, it is not possible to tell which outputs have the same owner as certain inputs. Assuming no Bitcoin addresses are reused, this prevents linking coins to the same owner across multiple transactions and renders information gained by $H1$ less useful.

Our analysis and evaluation shows an exponentially growing time required to analyze CoinJoin transactions. While speedups probably can be achieved, our model constitutes a simplification. We expect analyzing real CoinJoin transactions to be even more expensive for two reasons. First, we only look at non-derived mappings while it might be possible that the original composition is one of the derived mappings. Second, real CoinJoin transactions would include fees and analyzing them would require comparing set sums not by equality but by their difference. This will increase the runtime and also will introduce a difficult trade-off when choosing the threshold when comparing sums. A small threshold will result in less possible mappings but might not produce the original composition of sub-transactions. A large threshold will result in additional mappings, which we assume will decrease the probability of two coins being linked.

Our work only presents algorithms to implement knapsack mixing for CoinJoin transactions. To employ our approach, it must be used in an CoinJoin-based mixing protocol. Similar to ValueShuffle, the DiceMix [16] protocol could be used to for the P2P communication. As we are splitting outputs, we need to be able to generate new Bitcoin addresses for the payee. To facilitate this without requiring interaction, stealth addresses [17] could be used.

The resulting P2P mixing protocol would be compatible with Bitcoin today. As transactions and mixing now happen in a single step, all overhead of Bitcoin transactions compared to previous approaches is reduced. This translates into lower costs and faster mixing. Furthermore, it can be used for any transactions and could be implemented as default in standard Bitcoin clients.

However, currently, our output splitting algorithm requires knowledge of all sub-transactions. Using it in a P2P network would likely leak information to all participants. We are already researching a new output splitting algorithm, that mitigates this problem. First findings show that it provides at least as much anonymity as the input shuffling algorithm. Alternatively, this problem could be avoided by employing secure multiparty computing.

Our analysis also only considers CoinJoin transactions in isolation. It is currently unclear whether the linkability can increase when the analysis is extended to the whole transaction graph, which is left open for future work.

7. Conclusion

In this paper, we show that regular CoinJoin transactions do not provide anonymity if participants want to transact arbitrary values. Using our model, we are able to deconstruct them into the original sub-transactions, based on the unique coin values.

As a solution to this problem, we propose a novel output splitting algorithm that implements the knapsack mixing idea. Furthermore, we define metrics for estimating the anonymity of CoinJoin transactions and evaluate the mixing algorithm by generating and analyzing transactions. The results show that our algorithm reliably prevents unambiguously recovering the original sub-transactions. Only a small number of coins remain linkable. Existing transaction graph analysis heuristics can therefore no longer be used.

Our algorithm allows mixing with arbitrary values in a way that is compatible with Bitcoin today. This enables peer-to-peer mixing protocols that can be used by default for all transactions. In contrast to previous approaches, such a mixing protocol would not require additional transactions and thus reduce delay and fees.

Acknowledgments

The authors would like to thank Jan Henrik Ziegeldorf and Roman Matzutt for their helpful feedback. This work has been funded by the German Research Foundation (DFG) as part of project A2 within the Collaborative Research Center (SFB) 1053 "MAKI Multi-Mechanism-Adaptation for the Future Internet".

References

[1] S. Nakamoto. (2008) Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: http://www.cryptovest.co.uk/resources/BitcoinpaperOriginal.pdf

[2] blockchain.info. Bitcoin market capitalization. [Online]. Available: https://blockchain.info/charts/market-cap

[3] F. Reid and M. Harrigan, *An analysis of anonymity in the bitcoin system*. Springer, 2013, pp. 197–223.

[4] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. New York, NY, USA: ACM, 2013, pp. 127–140.

[5] M. Fleder, M. S. Kester, and S. Pillai, "Bitcoin transaction graph analysis," *arXiv preprint arXiv:1502.01657*, 2015.

[6] M. Ober, S. Katzenbeisser, and K. Hamacher, "Structure and anonymity of the bitcoin transaction graph," *Future internet*, vol. 5, no. 2, pp. 237–250, 2013.

[7] M. Spagnuolo, F. Maggi, and S. Zanero, "Bitiodine: Extracting intelligence from the bitcoin network," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 457–468.

[8] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for bitcoin with accountable mixes," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 486–504.

[9] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for bitcoin," in *Financial Cryptography and Data Security*. Springer, 2015, pp. 112–126.

[10] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-resistant mixing for bitcoin," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*. ACM, 2014, pp. 149–158.

[11] J. H. Ziegeldorf, R. Matzutt, M. Henze, F. Grossmann, and K. Wehrle, "Secure and anonymous decentralized bitcoin mixing," *Future Generation Computer Systems*, 2016.

[12] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "Coinshuffle: Practical decentralized coin mixing for bitcoin," in *Computer Security-ESORICS 2014*. Springer, 2014.

[13] Shared coin. [Online]. Available: https://en.bitcoin.it/wiki/Shared_coin

[14] A. Saxena, J. Misra, and A. Dhar, *Financial Cryptography and Data Security: FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ch. Increasing Anonymity in Bitcoin, pp. 122–139.

[15] S. Noether, A. Mackenzie, and M. C. Team, "Ring confidential transactions," https://lab.getmonero.org/pubs/MRL-0005.pdf, 2016, accessed: 27.09.2016.

[16] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "P2p mixing and unlinkable bitcoin transactions," 2017.

[17] P. Todd. Stealth addresses. [Online]. Available: https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg03613.html

[18] G. Maxwell. (2013) Coinjoin: Bitcoin privacy for the real world. [Online]. Available: https://bitcointalk.org/index.php?topic=279249

[19] ——. (2013) Coinswap: transaction graph disjoint trustless trading. [Online]. Available: https://bitcointalk.org/index.php?topic=321228.0

[20] N. van Saberhagen. (2013) Cryptonote v 2.0. [Online]. Available: https://downloads.getmonero.org/whitepaper_annotated.pdf

[21] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 2013, pp. 397–411.

[22] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 2014.

[23] E. Androulaki and G. O. Karame, "Hiding transaction amounts and balances in bitcoin," in *International Conference on Trust and Trustworthy Computing*. Springer, 2014, pp. 161–178.

[24] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno, "Pinocchio coin: building zerocoin from a succinct pairing-based proof system," in *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*. ACM, 2013, pp. 27–30.

[25] T. Ruffing and P. Moreno-Sanchez. Mixing confidential transactions: Comprehensive transaction privacy for bitcoin. [Online]. Available: https://people.mmci.uni-saarland.de/~truffing/papers/valueshuffle.pdf

[26] D. Berend and T. Tassa, "Improved bounds on bell numbers and on moments of sums of random variables," *Probability and Mathematical Statistics*, vol. 30, no. 2, pp. 185–205, 2010.