CoinJoin Anonymity Analysis

November 30, 2018

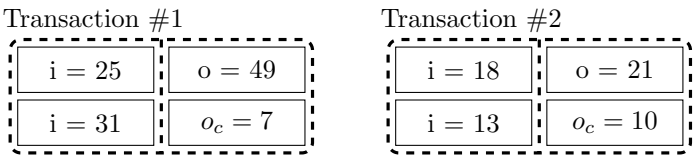
1 Introduction

CoinJoin is a technique for combining multiple Bitcoin transactions into a single one to make it more difficult for blockchain analyzers to determine which the original transactions are.

In this document we want to keep a record of our attempts to create a general approach for analyzing the pros and cons of the different mixing algorithms.

2 Mixing

Let's say we have these two transactions:



They can be combined in one transaction as follow:

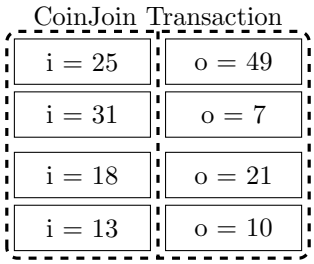


Figure 1 Simple CoinJoin transaction.

3 Analysis

We are going to focus specifically on blockchain analysis where the analysts goal is attach identities to coins (outputs) and tracking them using two well-known heuristics:

- H1: All inputs to a transaction are owned by the same entity
- H2: The one-time change address is controlled by the same entity as the input addresses

These heuristics cannot be applied to CoinJoin transactions but only to single payment transactions instead. For that reason the analysts have to unjoin the coinjoined transactions with the hope of finding the original payment transactions to apply the heuristics individually.

Side note: H1 is clearly a more robust assumption than H2.

3.1 Unjoining coinjoined transactions

Let's take the coinjoin transaction in figure 1. That transaction can be interpreted in many different ways but, *just for simplicity*, lets assume that we know this is a coinjoin transaction between two (no obvious) participants and that each transaction has no more than two outputs. This is enough for extracting the original transactions by grouping the outputs, the inputs and then match the sum of both sets:

Outputs:

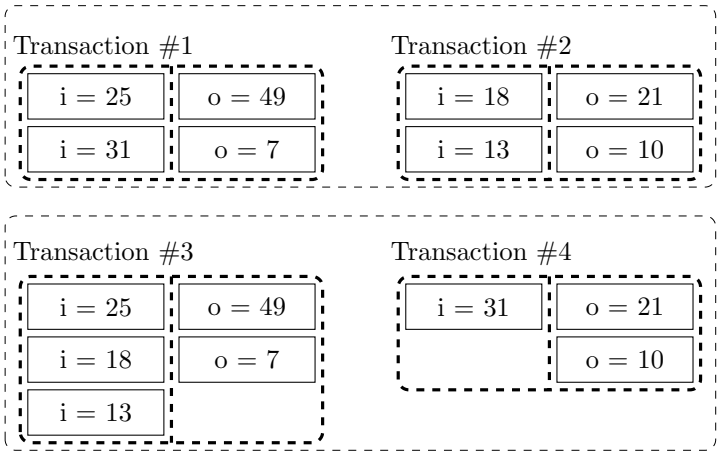
#	OutputSet	Sum(OSet)		#	OutputSet	Sum(OSet)
0	{49}	49		5	{49, 21}	70
1	{7}	7		6	{49, 10}	59
2	{21}	21		7	{7, 21}	28
3	{10}	10		8	{7, 10}	17
4	{49, 7}	**56**		9	{21, 10}	**31**

Inputs:

#	InputSet	Sum(ISet)		#	InputSet	Sum(ISet)
0	{25}	25		7	{31, 18}	49
1	{31}	**31**		8	{31, 13}	44
2	{18}	18		9	{18, 13}	**31**
3	{13}	13		10	{25, 31, 18}	74
4	{25, 31}	**56**		11	{25, 31, 13}	69
5	{25, 18}	43		12	{25, 18, 13}	**56**
6	{25, 13}	38		13	{31, 18, 13}	62

Before continuing with the example, it worth to note that given the coinjoin transaction has 4 inputs and that we know there are two sub-transaction then, the maximum number of inputs a sub-transaction can have is 3. These 3 inputs can be combined in 14 different ways. The number of ways a set with of n elements can be partitioned into disjoint subsets is called 'Bell number' and it grows rapidly as the value of n increases.

Having built this table it is possible now to extract all the possible transactions. In this case we have 4 sub-transactions (#1, #2, #3 and #4)



Also, we can see that if well all of these look perfectly valid, there are only two valid sets of transactions that could result in the original coinjoined transaction and in the image above we had wrapped them together in the dashed boxes. They are

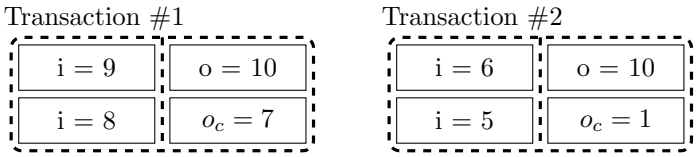
- Transaction #1 and Transaction #2
- Transaction #3 and Transaction #4

Clearly, other combinations are invalid because same inputs appears in more than one sub-transaction.

It is important to note that in this really simple example an analyzer has only 0.5 probability of choosing the right original sub-transactions set. In practice this does not use to happen but the good part is that we can make it happen simply by choosing the outputs values (that's what knapsack and traditional coinjoins do).

3.2 Equal-output coinjoin comparison

Lets say two people want to coinjoin together to obtain coins of the exact same value (for this example lets say 10). Both provide their transactions:



After join them, they get the following transaction:

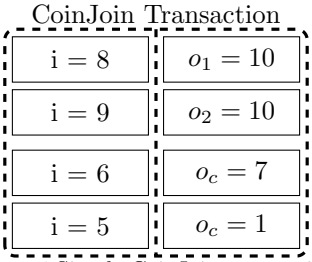
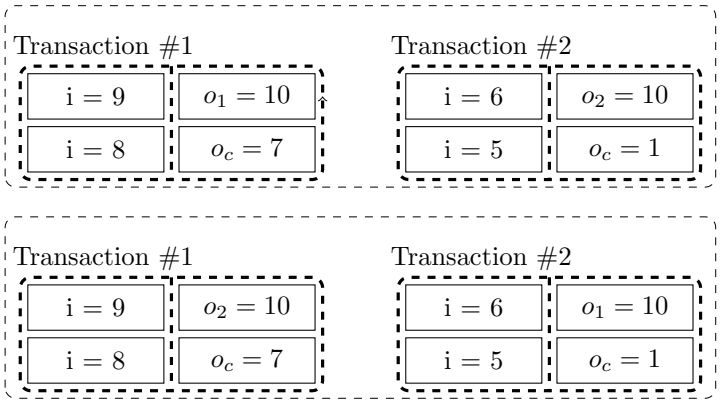


Figure 1 Simple CoinJoin transaction.

This coinjoin transaction can be analyzed in the same way that we did before but in this case we don't need some of the assumptions that we used then because unlike the nave coinjoin transaction we used initially, this transaction has a clear fingerprint and it is easy to know the number of participants involved simply by counting the number of indistinguishable outputs. Also, the change outputs are clearly identifiable. Finally, extracting the possible sub-transactions is trivial and doesn't require building big tables as before. Just take a change value, add it 10 and look what inputs sum gives us that result.



We can see in this example that the equal-outputs coinjoin transaction provides 0.5 probability to track the "anonymized" coin.