# Diffusion: Analysis of Many-to-Many Transactions in Bitcoin

1st Dylan Eck*
*Department of Mechanical Engineering*
*Colorado School of Mines*
dylaneck@mines.edu

2nd Adam Torek*
*Department of Computer Science*
*Boise State University*
adamtorek@u.boisestate.edu

3th Steven Cutchin
*Department of Computer Science*
*Boise State University*
stevencutchin@boisestate.edu

4rd Gaby G. Dagher
*Department of Computer Science*
*Boise State University*
gabydagher@boisestate.edu

***Abstract*—Bitcoin is a decentralized cryptocurrency that enables entities to transfer funds psuedo-anonymously. Investigative services like the FBI, however, have been able to subvert this using various techniques. To counter this, the Bitcoin community uses several methods to obscure transactions, including shared send transactions among others. We attempt to peer into these transactions through a pairing of address clustering and shared send untangling and test our experimental results through path finding between addresses. We then implement our methodology to test its effectiveness. Our findings show that while using clustering can improve path-finding results and shared send untangling, we recommend applying heuristics in combination to increase effectiveness.**

***Index Terms*—Blockchain; Bitcoin; Taint Analysis; CoinJoin**

## I. INTRODUCTION

Bitcoin is a decentralized, trust-less, and anonymous peer-to-peer cryptocurrency first proposed by Satoshi Nakamoto [1]. All transactions performed in Bitcoin are stored in the blockchain, a secure public ledger. Payments are made to addresses with no associated user-identifying information. This system provides psuedonymity, allowing particular users to be identified by their addresses, while not providing enough information to link addresses to real world actors.

Additionally, to further hide their activity, users may mix their coins with others using a variety of methods. In Bitcoin, these methods fall into two categories, defined by [2] as mixing services and shared send transactions. Our paper focuses on shared send transactions, which allow multiple users to send coins through a single transaction with many inputs and outputs.

These mixing services present a challenge for taint analysis as they obfuscate coin flows, making associating different entities difficult. The ability to untangle shared send transactions would enable more sophisticated analysis of Bitcoin network activity. In this paper, we seek to answer the question of whether a shared-send aware taint analysis tool combined with clustering heuristics can be used to more accurately track flow through the Bitcoin Network.

Our contributions include: Diffusion, a proposed novel heuristic augmented approach for clustering many-to-many Bitcoin transactions, methods for evaluating the accuracy of this approach, and the results of our own experimental evaluation of the approach.

*These authors contributed equally.

## II. RELATED WORK

Address clustering is a common topic of Bitcoin research. The study done by [3] uses a probabilistic model to determine the accuracy of their clustering. They also compare on-chain data with off chain information to validate their results. More studies, like the ones done by [4] show that it is possible to link IP addresses to Bitcoin addresses. Potential uses for this include risk scoring and address classification among others [5] [6]. Our experiments will focus on applying clusters to shared send transactions and will not analyze their validity or accuracy in any context but their effectiveness of transaction untangling.

Analysis has been done into mixing services as well. Methods of connecting inputs to outputs in mixers using a custom path-finding algorithm have been proven to be very effective at tracing funds in the mixing service Helix [7]. In addition, mixing services can be fingerprinted based on their behavior, either through inference based on behavior [8] or advanced machine learning algorithms like deep autoencoders trained to detect assumed features of mixers [9].

To our knowledge, few studies have been done on untangling shared send transactions. The most effective and thorough one we found was [2]. A consumer report with a less formalized and efficient algorithm called CoinJoinSudoku [10] finds groups of inputs and outputs where the total input is equal to the total output through a process of elimination. This algorithm does not have a formalized method of untangling transactions and took 30 hours to partially untangle a single transaction. Our experiments will use [2] as our untangling implementation for this reason.

Taint analysis methods for Bitcoin have also been researched. Besides work done by [11] for tracking illicit transaction activity, taint analysis has also been used to measure the anonymity of transactions and mixers [12].

## III. Background

Before we begin discussing the problem formulation, we must give some background information to lay a groundwork for understanding our problem.

### A. Blockchain

The blockchain is a distributed ledger that stores all transactions throughout Bitcoin's history. These transactions are stored on blocks which are generated from miners who solve cryptographic hash puzzles. Once the next block is mined, all of Bitcoin's peers vote in a consensus system to add it to the chain. If 51% of these peers vote to accept the block, it is appended to the blockchain and becomes the next block. Once added, the miner has complete control over what transactions are processed in their blocks.

### B. Transactions

Bitcoin transactions are publicly broadcast and stored in blocks on the blockchain. These transactions consist of unspent transaction outputs (UTXOs) which specify how much Bitcoin is to be transferred and the conditions required for execution. Once these transactions are created, they are broadcast to Bitcoin's network to be verified by miners. Once the transaction is put into a block, a script containing these conditions executes. This script generates outputs and moves the coins in the UTXOs to those outputs. Transactions may include a *transaction fee* which is paid to miners who append these transactions to their blocks.

Shared send transactions, shown 1, take advantage of multi-input, multi-output transactions in Bitcoin by using a similar numbers of input and output addresses and coin amounts. This is done to obfuscate who owns which inputs and output addresses and how much each party sent. Due to the complexity of these transactions, they are created and validated by third party services. This service can either be run on a server or in specialized software like Wasabi Wallet. Wasabi Wallet has a decentralized feature that lets users create, validate, and conduct a shared send transaction in Bitcoin [13].

## IV. SOLUTION: Diffusion

### A. Solution Overview

Our solution will use *path finding*, *untangling*, and *heuristic* techniques. The path-finding will consist of building an address graph and testing if any randomly selected addresses can connect to each other. We will run our untangling code on our data and see what many-to-many transactions we could untangle. If we were unable to untangle a transaction, we will remove it from the graph. After this, we will run a path-finding algorithm on the address graph. These will produce a matrix with the path lengths between each of our addresses and this will be our experimental result. We will divide our dataset into 7 day periods that are disjoint from each other to see the differences between each week in our data.

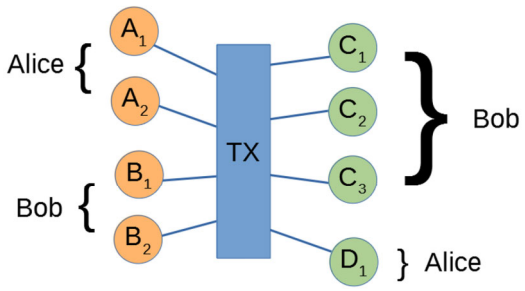


Figure 1: Example of a shared send transaction. The brackets represent which party, in this case Alice and Bob, own which input and output accounts.

This process will be our control trial. For each of the heuristics we use, we will apply them to many-to-many transactions before re-running our untangling and path-finding system. Once these trials are done, we will compare the resulting matrix with our baseline matrix and see if the average path length improved. We will then graph the average path length of each matrix, the number of path lengths, and the average length and number of connections for a few specific addresses of interest. We will also split this dataset into disjoint 7 day periods and compare results to the control trial for each of those periods to keep our heuristic results consistent with our baseline results.

### B. Diffusion *Components*

The first component of Diffusion is transaction untangling. If we detect a many-to-many transaction while parsing our data, we will attempt to untangle it. If the transaction has more than a specified number of inputs or outputs, our system will automatically label that transaction as impractical to unravel and move onto the next one. If our transaction can be untangled, we will label it as either simple if no untanglings are found, separable if a unique one is, and ambiguous if at least two partitions are found.

Once all transactions are labelled, we will construct an address graph with addresses as nodes and transactions as edges for each disjoint 7 day period in our dataset. If the transaction is ambiguous or impractical to untangle, we will only insert the addresses and not any edges to make the graph disjoint. Once our graph is constructed, we will then randomly select addresses that are 1 transaction away from a many-to-many transaction but not a part of one. We do not want to select addresses inside of many-to-many transactions because they will be overwritten when we apply our heuristics and untangling. If the next transaction is also a many-to-many transaction, we will remove it from our list as well.

Once we gather our possible addresses for each disjoint week, we will randomly select 2000 of them to perform path-finding. We will run Dijkstra's Single Source Shortest Path algorithm on all 2000 addresses to generate a 2000 x 2000

matrix containing the path lengths for all of our selected addresses. These randomly selected addresses will also be used to test the effectiveness of our clustering heuristics. This operation will be done for each disjoint week, producing a total of 4 2000 x 2000 matrices. This process will be our control trial, and one that we repeat for each of our heuristics.

For each of our three address clustering heuristics, we will run them on our dataset once for each disjoint week in our 1 month dataset. These three heuristics are as follows: multi-input single output transactions, change address clustering, and coinbase transaction clustering. These heuristics will be applied separately to see how effective each one was and enable comparisons of effectiveness between them.

In Bitcoin, multiple inputs flowing to a single output signals that the owner of the key that produced all those addresses signed all inputs and the output. This means that this transaction was authorized by a single entity. However, this clustering heuristic can be prone to error since shared send transactions have multiple input. To avoid this, we will not cluster any many-to-many transactions based on their inputs.

The second heuristic, change address detection, clusters addresses based on certain conditions. An OTC (one-time-change) address is an address created by a wallet to hold spare change after a transaction, usually spent in the next transaction it is used in. They are not meant to be used as a primary address, so this heuristic will not cluster an address if it appears in more than two transactions. Our conditions for change address detection are as follows:

- The transaction does not have the same number of inputs and outputs (so it is not a shared send transaction)
- Case 1: Transaction has two outputs
  - The change address must have 3 more decimal places than the main address
  - Address appears at most twice in our dataset, and is an output only once inside our data.
- Case 2: Transaction has more than two outputs:
  - The address is unique in the inputs (I.E. not a self-change transaction)
  - The address is unique in the outputs
  - This output appears at most twice in our dataset, this is its first appearance, and is an output once and at most an input once
  - The transaction address is in is not a coinbase transaction

This heuristic is more prone to error than any other we are testing since it is based on empirical observations and assumptions rather than how Bitcoin's system works. To account for this, we are adding conditions proposed by [14] to improve the accuracy and reduce the false negative rate of change address detection. This will reduce cluster size, but improve accuracy and enable us to have more confidence in our results.

Heuristic 3 groups the outputs of addresses if they are in a coinbase transaction. Coinbase transactions are rewarded to miners for finding the next block, and because they are a coin generation transaction, have no inputs. Mining is always done by entities with a shared goal in either single user mining or mining pools. Because of this, we can assume that all outputs of a coinbase transaction belong to the same entity and cluster them accordingly.

For heuristics 1 and 2, if we see that an input of the given transaction is already in a cluster and we have addresses that satisfy our conditions, those addresses will go to the same cluster. If not, we will create a new cluster for the given addresses. This same process will be done for heuristic 3 for the outputs of that coinbase transaction. If any of the outputs are found to belong to a cluster, the outputs will go to that cluster as well. This will enable us to aggregate clusters and build larger ones while still retaining accurate results.

### C. Methodology

With our solution described, we can now explain our methodology in detail, as shown in Figure 2. We start our method on step 1 through downloading block JSON files from Blockchain.com's Bitcoin data API. We downloaded 28 days worth of blocks from their API and saved them for parsing. In step 2, we parsed each of these JSON files and created a CSV file containing transaction data. Each CSV file has the transaction hash, input addresses, input amounts, output addresses and amounts, and the fee attached to that transaction. We created a CSV file for each day's worth of transactions to enable us to break apart our data by 7 day intervals. These CSV files form T, our primary data source for all of our experiments. We then split up T into four disjoint 7-day periods to perform analysis on individually.

Once we have created T, in step 3, we run our untangling code on each disjoint 7 day period to classify and untangle all of our transactions. In step 4, we discard our ambiguous and intractable transactions and keep the simple and separable ones. In step 5, we construct our address graph from our simple and separable transactions, drawing edges if the given addresses have contacted each other. If the transaction the addresses were involved in was impractical or ambiguous to solve, then we do not draw edges between those addresses. This will create our graph for path finding, $S_0$.

In step 6, we will randomly select addresses 1 jump away from addresses in many to many transactions, then filter out any from our selection that are also involved in many-to-many transactions. Once that is complete, we will randomly select 2000 addresses from that list, forming the address set A. This set is what we will use to perform path finding. In step 7, we will take our addresses A and our graph $S_0$ and perform Dijkstra's Single Source Shortest Path algorithm to get each path length between all 2000 of our addresses. This, in step 8, will produce our matrix M to perform analysis on. We will apply this process for each of our disjoint 7 day periods from our 28 days worth of transactions.

In step 9, we will run a selected heuristic on each disjoint week in T, H, to produce our resulting transaction data, T'. In step 10, T' will then be run through our untangler where we repeat steps 4 through 8 to get a different matrix, M'. We will compare this matrix to M and see if there are

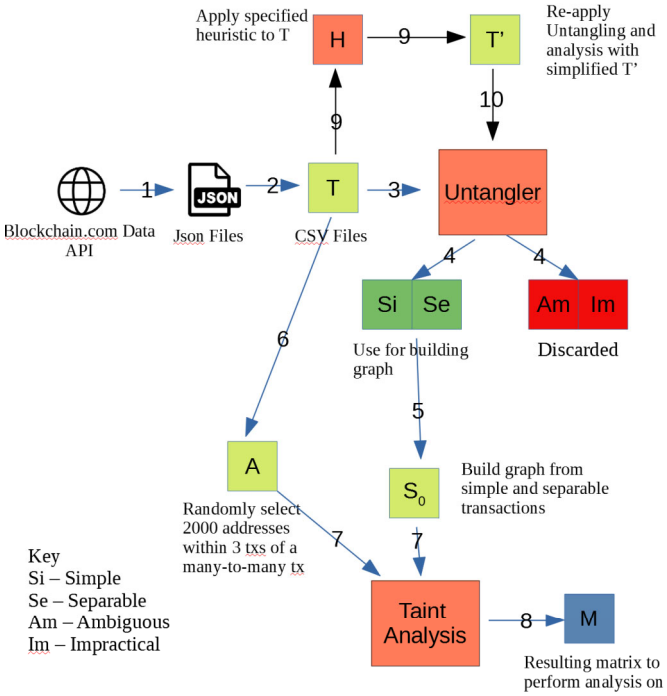


Figure 2: Our methodology

differences between the average path lengths, number of paths, and changes between specific addresses of interest. This will be repeated for each disjoint week and compared with other week's results to see the differences and similarities. Figure 12 (Appendix A) shows a flowchart of this methodology and the order of operations.

## V. Experiments

### A. Implementation and Setup

All of our code was written in Python version 3.8.5. Our graph implementations were completed using networkx, a graph analysis library available on Python. For our systems, we used compute nodes provided to us by Boise State University on their R2 High Performance Cluster. These nodes have 192 gigabytes of RAM and 2 Intel Xeon E5-2680 CPUs, each with 14 cores that can run up to 2.4 GHz. We ran our experiments on 28 days of transactions downloaded from Blockchain.info, totalling 4 7-day periods. Each of these was analyzed independently of each other to enable quick parallel computation and inter-week comparisons of results. For our experiments, we set the maximum number of inputs or outputs a transaction could have to 8. If the transaction was many-to-many and had more than 8 inputs or outputs, our code automatically labelled it impractical and moved into the next transaction. The source code for our implementation and experiments can be found here [1].

[1]Source Code: https://drive.google.com/drive/folders/1vjoWwkmeEPFDEH4xIzwxpIKCvmr4iO0P?usp=sharing

### B. Heuristics Results

We also performed scalability testing on our heuristics. We normalized our data into 1 million transaction intervals, from 1 million to 5 million transactions. We ran each heuristic on these normalized intervals to determine their scalability. The results from this experiment are shown in Figure 3 below.

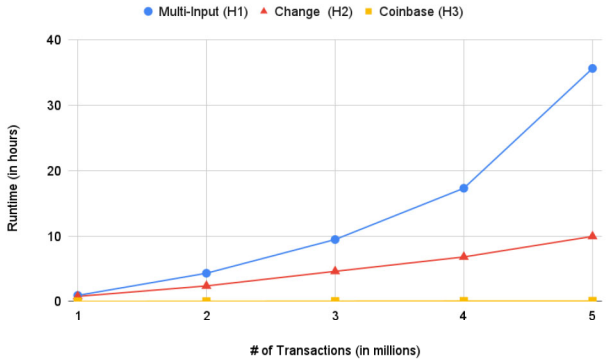


Figure 3: Heuristics Scalability Results

The runtimes above are due to the nature of each heuristic. Coinbase transactions only have to satisfy one condition before being grouped, making our clustering simple. Change addresses have to satisfy multiple conditions, making their clustering more computationally difficult, but they also produce smaller cluster sizes, making cluster aggregation easier for this heuristic than for multi-input clustering. Multi-input clustering takes the longest time to run and scales quadratically because of the large cluster sizes it generates, which are multiple times the size of the largest addresses for change address (H2) clustering. The supporting evidence for this can be found below in Table I. For example, on week 3, the max cluster size of H1 is 121139 addresses, approximately 12.58 times bigger than H2's largest cluster, which is 9633 addresses.

Table I: The average and max. cluster size for each heuristic with each week

| Heuristic | Type | W1 | W2 | w3 | W4 |
|---|---|---|---|---|---|
| H1 | Average | 13.0740 | 13.0610 | 12.3090 | 10.8514 |
| | Max | 48315 | 102154 | 121139 | 47027 |
| H2 | Average | 9.9597 | 10.6543 | 9.6055 | 10.0126 |
| | Max | 16521 | 4075 | 9633 | 5239 |
| H3 | Average | 1.0 | 1.0 | 1.0 | 1.0 |
| | Max | 1 | 1 | 1 | 1 |

### C. Untangling Scalability

To test the scalability of the untangling algorithms, our system was executed on inputs of linearly increasing sizes from one million to five million transactions. These experiments were run ten times and averaged to produce our scalability graph. For each test, we recorded the time it took to untangle transactions, perform address selection, and do pathfinding. The legend above the graph in Figure 5 shows the color and

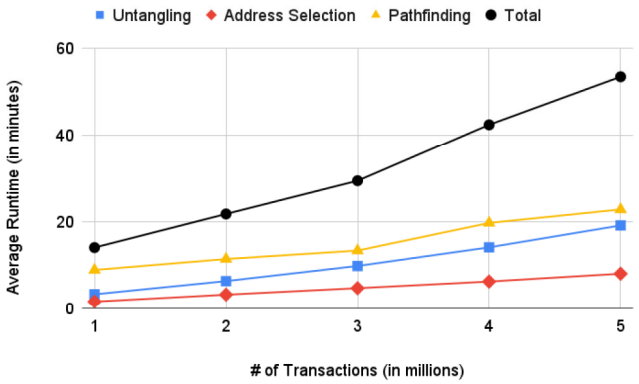


Figure 4: Untangling Scalability Results

shape codes for each statistic. We also include the total runtime to show how the whole system scales.

Diffusion's untangling system scales linearly since it processes on a transaction-wise basis. Because we multithreaded our solution and limited the number of inputs and outputs our untangling code attempts to process, it is able to grow linearly rather than super-linearly with our data. A slight dip can be seen on the 3 million transaction interval. This is due to the pathfinding step taking a only a marginally longer time to process transactions than it did on the 2 million transaction mark.

As expected, pathfinding takes the longest period of time because it finds a path in our address graph for each of the addresses in the 2000 x 2000 matrix. The untangling step takes the second-longest amount of time because it untangles, simplifies, and classifies all transactions it processes, which is less work than the pathfinding, but more than the address selection. Our fastest step is also the simplest one, only randomly selecting addresses from a weakly connected graph.

### D. Pathfinding Results

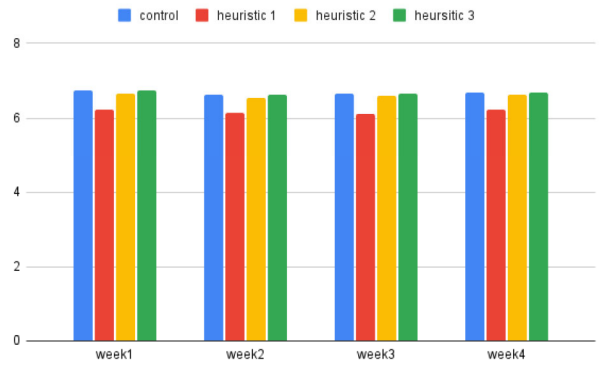


Figure 5: Average path length for each week.

Figure 5 above shows the average path length for each of the heuristics for each disjoint week we tested. The weeks are grouped next to each other to make the differences in the average path length between the control and each of the heuristic experiments. Figure 1 clearly shows that heuristic 1, multi-input clustering, had the greatest impact on shortening the average path length between connectable nodes. Heuristic 2, change address clustering, was significantly less effective than multi-input clustering, but still had a minor impact on the average path length. Heuristic 3, coinbase clustering, had no impact on the average path length compared to the control because it created small clusters that had no impact whatsoever on the final result.

Table II: The number of paths found for each week per each heuristic

| | Control | H1 | H2 | H3 |
|---|---|---|---|---|
| week 1 | 1,564,788 | 1,564,788 | 1,564,788 | 1,564,788 |
| week 2 | 1,661,526 | 1,661,526 | 1,661,526 | 1,661,526 |
| week 3 | 1,587,255 | 1,587,255 | 1,587,255 | 1,587,255 |
| week 4 | 1,633,426 | 1,633,426 | 1,633,426 | 1,633,426 |

We must note that the number of paths found and the percentage of connectable pairs did not change between trials and the control. This is because there are transactions in our data that were completely disjoint and did not interact with each other within our disjoint weeks. Clustering will not bring sets of transactions together that never interacted. Table II shows the number of paths found for each week for each heuristic, demonstrating that the heuristics had no effect the number of paths found in each week.

### E. Heuristics Interaction Visualizations

To show how the clusters we created connect to each other and differ between each heuristic, we made a visualization tool that shows transactions that happened between clusters from our data. This tool was built using D3.js force-graph and deployed on a web server for live viewing. The link can be found here [1]. Figures 6,7 below shows these visualizations for week 3 in our data between June 17th and June 23rd. Note that the closer the nodes are in the visualization, the more connections they have with each other and the more central they are. We also did not create visualizations for coinbase clustering because of how few connections it made.

Figure 6 shows why the multi-input address clustering was the most effective in shortening path lengths. This heuristic created only a few very central clusters that interacted with dozens of others, allowing for central points to form, shortening the paths between addresses, creating shorter average path lengths. These central clusters form a tight-knit community, with long trailing paths extending outward from them. The following visualization also shows disjoint communities where clusters did not transact with each other outside of that community.

Figure 7 shows the communities formed between clusters in change address clustering. This heuristic produced fewer important clusters, resulting in a larger, less centralized mass in the middle. This explains why the average path lengths

[1] http://mec402.boisestate.edu/blockchain/blockgraph.html

Figure 6: Interactions between multi-input clusters

Figure 7: Interactions between change address clusters

were only marginally better than the control's, as few central communities formed in our address graph. Long chains of clusters that interacted with each other also extend out from the larger central mass similar to multi-input address clustering. Many smaller disjoint communities outside of the large central mass also exist, similar to multi-input clusters.

## VI. Conclusion and Future Work

Shared send transactions are one of the methods for increasing anonymity in Bitcoin's network and enabling criminals to evade detection, mixing their dirty coins with others' clean coins. This presents a problem for taint analysis and pathfinding in Bitcoin since shared send transactions obfuscate who owns which coins, presenting a clear need for a method of using shared send untangling along with connecting addresses. We demonstrated that this is possible by performing shared send untangling, discarding ambiguous and impractical transactions, and building address graphs to perform taint analysis. To test our methodology, we implemented our system in Python and downloaded 1 months worth of transaction data, T, from Blockchain.info's API to perform analysis, separating it into four disjoint weeks.

To increase the effectiveness of shared send untangling, we applied clustering heuristics to our data set T. We found the multi-input, single-output clustering heuristics to increase the effectiveness of untangling. We also found that applying the coinbase clustering heuristic by itself to be ineffective and produce no clear results, so we recommend using it in tandem with multi-input clustering to make a larger and clearer picture of entities in Bitcoin. We have also shown that it is possible to apply these clusters on a weekly basis rather than against the whole blockchain.

Our work, Diffusion, could provide a foundation and methodology to perform shared send untangling, Bitcoin address graph construction, and address clustering to give more insights into shared send transactions and how to untangle them. Future systems could expand upon this foundation and build a comprehensive system for untangling shared send transactions in Bitcoin.

Potential future work could include: investigation into the identifying characteristics of the different classifications of transactions to aid in the creation of better informed transaction anlysis methods, and more in depth research into methods of visualizing shared send transactions and other mixing services to gain better insight into the intricacies of these transactions.

## References


[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *bitcoin.org*, 2009.

[2] Y. Yanovich, P. Mischenko, and A. Ostrovskiy, "Shared send untangling in bitcoin," *bitfury.com*, vol. 2016, 2016.

[3] D. Ermilov, M. Panov, and Y. Yanovich, "Automatic bitcoin address clustering," in *2017 16th IEEE ICMLA*, 2017.

[4] A. Biryukov and I. Pustogarov, "Bitcoin over tor isn't a good idea," in *2015 IEEE SP*, 2015.

[5] M. Spagnuolo, F. Maggi, and S. Zanero, "Bitiodine: Extracting intelligence from the bitcoin network," in *IFCA FC14*. Springer, 2014.

[6] J. Osterrieder and J. Lorenz, "A statistical risk assessment of bitcoin and its extreme tail behavior," *Annals of Financial Economics*, vol. 12, no. 01, 2017.

[7] Y. Hong, H. Kwon *et al.*, "A practical de-mixing algorithm for bitcoin mixing services," in *2nd ACM BCC*, 2018.

[8] Novetta, "Survey of bitcoin mixing services: Tracing anonymous bitcoins," 2015.

[9] L. Nan and D. Tao, "Bitcoin mixing detection using deep autoencoder," in *2018 Third IEEE DSC*, 2018.

[10] K. Atlas, "Weak privacy guarantees for sharedcoin mixing service," *CoinJoin Sudoku*, Sep 2014.

[11] T. Tironsakkul, M. Maarek *et al.*, "Probing the mystery of cryptocurrency theft: An investigation into methods for cryptocurrency tainting analysis," *arXiv*, 2019.

[12] M. Möser, R. Böhme, and D. Breuker, "An inquiry into money laundering tools in the bitcoin ecosystem," in *2013 APWG eCrime Researchers Summit*, 2013.

[13] *Wasabi Docs*, Sep 2020.

[14] H. Xi, H. Ketai *et al.*, "Bitcoin address clustering method based on multiple heuristic conditions," 2021.