Dash cryptocurrency deanonymization

Foteini Baldimtsi, Joao Brandao, Panagiotis Chatzigiannis, and Ioanna Karantaidou

George Mason University

Abstract. Cryptocurrency payments form a transaction graph. The goal of anonymity techniques is to obfuscate this graph in a way that the money flow cannot be traced. The basic suggested practice for users in order to gain some privacy is to generate fresh addresses to store their coins all the time. Heuristics on Bitcoin, a cryptocurrency that only offers this type of pseudonymity, allow for address clustering. If a cluster ends up uniquely identifying a user, this corresponds to a successful deanonymization attack.
In this report, we examine such clustering techniques on Dash, an extension of Bitcoin that applies mixing to increase anonymity. We implement the well-known Bitcoin clustering techniques on Dash and we extend these techniques to attack any mixing protocol as long as it does not hide the payment value.

1 Introduction

1.1 Cryptocurrency basics

The blockchain. A blockchain is an append-only distributed ledger that is maintained through a peer-to-peer network. It consists of sequential blocks, each containing the hash of the previous block such that they form a verifiable chain and any alteration becomes immediately apparent.

Transactions. Coins are stored in addresses. An address is associated with a spending secret and a value. In order for a transaction to be valid, the following have to hold: the spender has to prove knowledge of the spending secret and the input address(es) must hold the necessary value. If a transaction is valid, it ends up in a future block.

Account-based vs UTXO model As mentioned above, in order for a transaction to be valid, the address must hold the necessary value that the spender wishes to transfer. There are two models to enforce this requirement: the account-based and the Unspent Transaction Output (UTXO). The account-based model uses accounts as addresses and keeps track of each account's balance as a global state. The requirement is satisfied as long as the sending value is less or equal to the balance. The moment of the transaction, the spent value is deducted from the account and the global state is updated. The most popular account-based cryptocurrency is Ethereum.

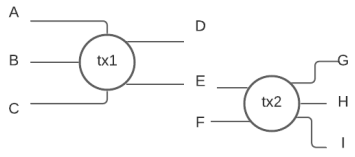
In the UTXO model, each address is associated with a fixed value (in some cases, it can be increasing but never decreasing). An address is valid and associated with a value if it is the output of a previous valid transaction that appears on the chain. Since the value never decreases, the address is only used once and then is removed from the UTXO set. The requirement "the input address(es) must hold the necessary value" is satisfied as long as the address has not been used as an input to a past transaction, or in other words, double-spending has not occurred. Some popular cryptocurrencies that work in the UTXO model are Bitcoin, ZCash, Monero and Dash.

Since blockchain transactions can be publicly verified, it means they do not need to happen under the supervision of a trusted entity (e.g. a bank). While decentralization is the main motivation for users to transact using cryptocurrencies, some users also prefer them as they give them a sense of anonymity (especially in cryptocurrencies that work in a complete permissionless, decentralized model). However it is well known that user anonymity is not guaranteed - in fact, given the public nature of the blockchain, there are several techniques for analysis and user deanonymization. In the next sections we review these issues. We use first cryptocurrency (Bitcoin) as the starting point to illustrate how user anonymity can be degraded in several cases.

1.2 Bitcoin and pseudonymous addresses

As Bitcoin emerged, it was widely believed that it offered complete anonymity. As opposed to traditional banking systems, users could trivially create an account and transact without needing to provide any personal information or identification. A Bitcoin's address was a pseudonym, completely de-associated from one's physical identity. However in practice, a user might conduct an in-person payment in BTC. In order to minimize this privacy leakage, Bitcoin users are encouraged to use a new address every time they need to store coin. This way, even if one of these addresses gets linked to their identity, the rest of the transactions remain private.

Fig. 1: A transaction-centered graph



However, even if the addresses of a user are randomly picked and seemingly unrelated to each other, it has been observed that Bitcoin (as well as other cryptocurrencies, even anonymous ones like Monero mostly in their early stages) is susceptible to coin tracing.

In the next paragraph, we describe how a simple case of coin tracing can be conducted on a transaction graph. In Section 2.1, we are going to revisit this example and point out what coin tracing ability means even for cryptocurrencies that are anonymous, as opposed to just pseudonymous like Bitcoin.

1.3 An example of coin tracing via graph analysis

Graph with transactions as nodes. An example of immediate data representation with graphs can be found in Figure 1. It is immediate because blocks are already organized as sequence of transactions with input, output addresses. The graph nodes here are the transactions (each represented with a transaction id on the block explorer) and each transaction node follows after all nodes that include its inputs as their outputs.

Graph with addresses as nodes. As a meta-analysis, one can derive a graph with addresses as nodes using the blockchain information. An example is given in Figure 2 where one can see the initial coin graph that corresponds to the transaction graph of Figure 1. An edge (X, Y) denotes there is a possible movement of money from address X to address Y or else, X is an input to the same transaction where Y is an output.

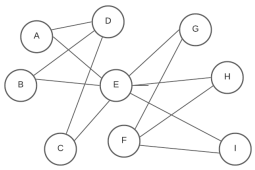


Fig. 2: Probabilistic coin tracing

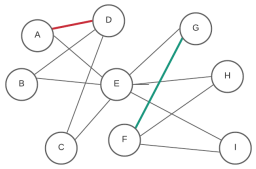


Fig. 3: Probabilistic coin tracing with side information

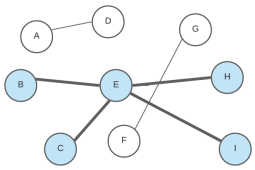


Fig. 4: Deterministic coin tracing

Fig. 5: A coin-centered graph related to graph of Figure 1

Often, by using available side information such as physical identity reveal (such as a public service or an exchange website) or the address values (if publicly visible), one can decide a coin's route. For example, let us assume that an outside observer can see that address A transferred all coins to address B and address F to address G (Figure 3). That means that any other edge (A, Y), (X, B) and (F, Y'), (X', G) are connected with probability 0 and can be removed from the graph. In Figure 4 the route of coins in addresses B, C is deterministically defined. One can see that although addresses are randomly generated, relationships between them can be found by analyzing the transactions graphs.

In the following sections we discuss mitigation techniques that have been suggested in order to prevent coin tracing. Pseudonymity and successful coin route obfuscation are believed to lead to user anonymity.

Network level attacks Another way to correlate random pseudonymous addresses to each other is via a network level attack [9, 14]. A peer-to-peer network participator has been found to be able to link pseudonyms to IP addresses and, as

a result, to each other. Attacks have been designed to even overcome the protection offered by the use of Tor [8]. Since this is a stronger adversarial model attack, we will only focus to de-anonymization attacks that only exploit publicly available blockchain information.

1.4 Anonymity mechanisms

We define *Anonymity* as the property of breaking the linkage between transactions and hiding the sender's and receiver's identity [4]. We note that this is a requirement in order to avoid the coin tracing technique explained in Section 1.3.

The first step for anonymity is pseudonymity, i.e. the disassociation between address and physical identity that is offered by Bitcoin. On top of that, anonymity involves techniques to hide the receiver's identity (meaning that every transaction output is often enforced to be a fresh, unique address), the sender's identity (divided in set anonymity and full anonymity) and finally, to hide the transferred amount (Confidential Transactions).

In order to implement the above techniques, the following cryptographic primitives are widely used on top of pseudonymous addressing: ring signatures, mixers, cryptographic commitments, Zero-Knowledge Proofs (ZKPs) and stealth addresses. We briefly explain each primitive.

- A *Ring Signature* is a special type of digital signature that consists of a ring of public keys, a message and the signing algorithm output denoted by σ. The signing algorithm takes as input a secret key that corresponds to one of the public keys of the list(called the ring). If σ verifies to 1, a verifier is convinced that such a secret was used during the signing, but cannot distinguish which public key it corresponds to.
- A *Commitment scheme* is a primitive that allows one to commit to an element in a hiding way, meaning that no information can be inferred about the committed element by the commitment alone. The committing entity uses a secret, revealed during the opening phase. The scheme must be binding, i.e. the committing party must be unable to find a secret that opens to an element different that the one that was initially used.
- A *ZKP* is a protocol by which a prover can prove to a verifier that they hold a witness that satisfies a public statement (often called a relationship $\mathcal{R}$). The goal is that no other information must be reveal to the verifier other than the prover's knowledge of a witness (and not the witness itself no matter the protocol's repetitions).
- A *stealth address* is a way to generate receiver's addresses. There are two ways to generate a fresh, random-looking address: one is when the receiver generates and communicates the address to the sending party. A stealth address solves this communication overhead and also keeps the two parties unlinked on the network level. A sender can generate a stealth address using the recipient's public key such that: stealth addresses are unlinkable to each other and they can be spent with the receiver's unique spending secret.

Stealth addresses are widely used to the anonymous cryptocurrency Monero to protect receiver's privacy.

The final anonymity-enhancing technique is the so called *mixers*. A mixer or a tumbler is a service that takes a set of identifiable coins as an input, mixes them with each other and outputs a set of coins with the following goal: to obfuscate the connection between each mixing's output to the original source coin. Mixers can be categorized into centralized that make use of a central entity to assist in the process such as Tumblebit [12] and to decentralized such as CoinJoin [1].

Although attempts have been made to apply anonymity techniques on account-based cryptocurrencies such as Ethereum [6], multiple concurrency issues arise and attempts to overcome them lead to performance degradation.

Therefore, the most widely used anonymous cryptocurrencies are on the UTXO setting. In the next section, we present such cryptocurrencies with their main properties.

1.5 Popular anonymous cryptocurrencies

In this subsection, we give a brief overview of the most popular and widely used anonymous cryptocurrencies. Although third party mixing services can be used on top of other cryptocurrencies as well, here we include currencies that with built-in, inherent anonymity properties. The reason behind this distinction is because their privacy level can be easier quantified as it is less strictly based on user behavior. The top anonymous cryptocurrencies in terms of usage, ordered by popularity are the following:

Monero (XMR). It is the most popular anonymous cryptocurrency with market capitalization ranking 24. In terms of anonymity, it uses the set anonymity technique by applying the following primitives: stealth addresses for recipient privacy, ring signatures for sender anonymity and confidential transactions for transfer value hiding. A confidential transaction can be further down analyzed to the following primitives: a commitment to the value associated with a public address and a proof of equality between the sum of the input committed values and the sum of the output committed values. Double-spending avoidance is enforced by a unique key image value generated for every spent input.

Dash (DASH). Dash is the second most popular anonymous cryptocurrency with market capitalization value ranking 48. It uses the Bitcoin core protocol and applies centralized coinjoin mixing with the same denomination (meaning coin value). A detailed description of how Dash works is given in Section 3.

ZCash (ZEC). ZCash is the third most popular anonymous cryptocurrency with market capitalization value ranking 52. It is the evolution of the Zerocoin protocol [17] that was designed on top of Bitcoin. It consists of a pool of non-private transparent addresses and a pool of shielded addresses. In order to spend

from a shielded address anonymously, one has to construct a ZKP for the following statement: there is an address in the pool for which the spender knows the spending secret associated with it, the balance is maintained and the value of the input is transferred to the value of a freshly generated output. The pool consists of all addresses every existed on the ledger, therefore ZCash offers full anonymity. The ZKP is instantiated with zk-snarks, a protocol that offers succinct proofs with short verification time (independent of the pool size) with the cost of expensive setup and long public parameters and expensive proving time (linear to the pool size). Double-spending avoidance is enforced with the use of nullifiers, a unique value that is public and corresponds to one spent coin. The spending ZKP includes a proof of correctness for the computation of the corresponding nullifier. Any third party can verify the proof and perform a lookup to the nullifier set (represented as a Merkle tree) to make sure that the address used has not been used as an input to a previous transaction.

1.6 Related work

We present published work on deanonymization techniques on plain Bitcoin, anonymization techniques offered as services on top of Bitcoin or cryptocurrencies that extend Bitcoin, such as ZCash.

[16] designs attacks on Bitcoin privacy. The authors focus on the publicly visible flow of money. As a firs step, they identify addresses controlled by some known real-world user. Such users are mining pools, exchange banks, vendors, etc. They then apply well-known heuristics to form clusters of addresses that belong to the same user. If such clusters overlap with any of the known addresses, the whole cluster gets mapped to its physical identity. To emphasize the contribution on pseudonymous cryptocurrency clustering, the others apply the attack on criminal activity discovery. If clusters overlap with any address from the list of Bitcoin thefts, their transaction activity can be traced.

[15] defines the notion of "taint resistance" as an anonymity measurement. The authors analyze Coinjoin's taint resistance criteria satisfaction and explore the weaknesses that come from Coinjoin misuse. They explore a passive adversarial behavior in the case where transferred values are distinct and in the clear also an active adversary that influences the participants distribution. They run an experimental measurement of taint resistance indicators that can be achieved under these adversarial models.

[19] performs another Bitcoin anonymity analysis. The authors define the two kinds of networks that appear in a pseudonymous cryptocurrency, the transaction/address network and the user network. They then integrate off-network side channel information in their analysis. Behaviors such as donations, voluntary identity disclosure, TCP/IP packet pushing order, etc, are used with the goal to integrate the two networks and derive money flow graphs. They present their passive analysis deanonymization results as flow graphs.

A quantitative analysis on Bitcoin is conducted in [20]. A static analysis is run on the whole chain to extrapolate statistics such as the number of addresses

per entity (derived by transaction input heuristics), incoming BTC amount distributions, balance distribution, transaction numbers etc. Such works are used as a base for attacks that use user behavior analysis as side information to conduct deanonymization.

[5] formalizes Bitcoin-based clustering techniques and the notion of "address unlinkability" as an anonymity measurement. More specifically, clustering heuristics such as common input addresses and change address identification are explored. The authors also define the notion of "User Profile Indistinguishability" as another anonymity metric. In order to evaluate privacy according to their second metric, the authors conduct a user-behavior analysis where they incorporate information such as geographical location and time zone. Finally, they evaluate their metrics by using ground truth data and simulated data.

In [18], traceability of Bitcoin transactions under the use of an anonymization service is investigated. Their attack applies to a high level idea of a mixing service. They exploit the fact that distinct coin values are used during the mixing and the fact that such values are posted in the clear, i.e. no confidential transactions are used. Throughout the experiments, no inside information possessed by any central mixer is used, except public blockchain data.

[10] exploits the side information of web cookies to cluster addresses of the same user. They extend their attack to work on top of transactions that take place after a multi-round mixing protocols run. Specifically, Coinjoin is assumed as the mixing protocol instantiation. The side information of their attack is not available through the public ledger, their adversary is rather third companies such as shopping sites, the advertisement industry, etc. Examples of such information are the payment time, the payment address, the price (transferred value) and in cases of intrusive trackers, even the user's real identity.

[23] revisits all clustering heuristics suggested on Bitcoin. They present a clustering implementation, starting with a ledger crawler and clustering implementation. On top of clusters, the authors perform an entity relationship analysis: a heuristic algorithm for identifying relationships between Bitcoin entities, for example group activities between communities on the network.

[11] is reasoning on the effectiveness of clustering algorithms towards deanonymization. The authors derive statistics about how often privacy recommendations are ignored by users. Privacy-harmful actions are indentified from real data. Such actions are: address reuse instead of fresh address generation per transaction, super-clusters created by major services and predicted individual user behavior that leads to an expected slow incremental growth of address clusters.

[7] extends clustering techniques to also include the so called "unknown transactions". Such transactions include the null value as an input or output address and although disregarded by known clustering techniques, they often end up as part of the chain. The authors approach clustering by using single-source directed unlabeled acyclic weakly connected graphs. They then connect graphs through an isomorphism to link them to a unique user behavior.

In [13], the anonymity guarantees of ZCash are being investigated. A large portion of ZCash does not use shielded pool address, therefore it behaves like

plain Bitcoin. The authors repeat clustering attacks on the non-private part of ZCash chain. Finally, they study the interaction of such non-private coins with the shielded pool and more specifically, ZEC withdrawal and deposit. They assign large spikes in withdrawl and deposit activities to the founders and miners of ZCash by applying user behavior analysis. For transactions solely inside the shielded pool, they probabilistically assign transactions to hacker activity according to the number of inputs and outputs which is the only information leaked by a cryptocurrency that offers full set anonymity for both sender and receiver.

Panos: [?] also: https://link.springer.com/chapter/10.1007/978-981-16-6554-7_31 https://www.inderscienceonline.com/doi/abs/10.1504/IJSN.2021.119391

2 Coinjoin

Coinjoin [1] is a mixing technique that takes place in one transaction. The high level idea is the following: n addresses with their corresponding values are given as input and n addresses with their values as outputs. The sum of inputs must be the same as the sum of outputs. Digital signatures must be included signed with the secret key that corresponds to each input (each participant proves that they know the spending secret). The signed message is the whole transaction information.

The idea is rather simple, the challenges of Coinjoin are with concern to how each participant will communicate their pair of input-output addresses without revealing the mapping to other users and how the amounts used are not going to leak any information to the users and an outside observer [15].

Solutions have been suggested in literature but have not yet been implemented. [22] suggests the use of DC nets as an anonymous broadcast technique that can be used for decentralized implementation of Coinjoin. [21] suggests the use of Confidential Transactions on top of Bitcoin and Coinjoin in order for users to be able to mix arbitrary amounts. All these solutions regarding decentralization and multiple denominations come with extra computation and communication cost.

2.1 Graph node Clustering and Mixers

In Section 1.3, we show how one can apply side information (using the public ledger or the network level) to derive a coin's flow by iteratively removing graph edges. A coin's flow in a system with sequential rounds of mixing corresponds to mixed coin deanonymization. Note here that the coin's flow corresponds to the same user creating fresh addresses and moving the funds. Their goal is to break the link between where they acquired the coins and where they spent them.

In the following sections we show how to use heuristics in order to iteratively apply clustering techniques on Dash's coin graph, a cryptocurrency with inherent mixing. If such clustering extends to different blocks, we view it as a first step to mixer-based cryptocurrencies deanonymization.

3 Dash cryptocurrency

As already mentioned, Dash is the second most widely used cryptocurrency with built-in anonymity mechanisms. More specifically, it incorporates the idea of efficient mixing with various levels of security guarantees according to the number of mixing rounds. For a privacy-concerned user, there is a trade-off between anonymity level and the amount of fees paid.

Dash evolved from Darkcoin [2], a fork on Bitcoin and got rebranded as Dash [3] on 2015. Compared to Bitcoin, it introduces one more entity: the *Masternodes*. The pre-existing entities that are shared with Bitcoin are the users, i.e. the senders and receivers that appear in a transaction and the miners, i.e. the entities that take care of the maintenance and development of the blockchain ledger by verifying transactions, forming blocks, run a consensus protocol and agree on the ledger's content. In a Proof-of-Work ledger like the one run by Bitcoin, the trust model about miners is the one that assumes an honest majority.

Masternodes. A Masternode is a trusted entity that in addition to relaying messages on the peer-to-peer network and verifying transactions, offers the following additional services: building and approving the two different types of transactions that are named InstantSend and PrivateSend and vote on the governance of the Dash ecosystem.

Profits and Governance. Cryptocurrency profits are collected in the form of transaction fees. They are alocated to fund honestly behaving Masternodes, miners and future project ideas.

Being a Masternode has the following requirements: to submit 1000 DASH as a collateral and maintain that amount as unspent funds in the system and to provide a proof of service to the network. Masternodes that fail to meet the requirements are marked as inactive.

The proof of service involves the Masternode's assistance with Dash's two types of transactions: InstantSend and PrivateSend.

InstantSend transactions are typical Bitcoin transactions but with a shorter validation type. Typically, a transactions is tagged as valid after a number of blocks after the block it was mined. This holds because there is a non-negligible probability of block reordering. A reordering can lead to a double-spending attack.

A user has the option to ask for a transaction of type InstantSend. This corresponds to the following sequence of actions: Masternodes accept in real-time and mark the transaction as valid. If takes at least 60% of the Masternode quorum to approve the transaction, sign and broadcast it through the network. An approved InstantSend transaction guarantees that it will be mined into a future block and also that any future transaction is going to use the updated UTXO set and double-spending attacks are not possible.

PrivateSend. A PrivateSend transaction is initiated by a privacy-concerned user and it gets implemented as multiple Coinjoin rounds. In order to mitigate Coin-

join privacy and performance weaknesses we explained in Section 2, Dash's Coinjoin implementation has the following characteristics: it is centralized, meaning that there is a trusted entity (here the Masternode) that communicates with the participants and forms the Coinjoin transaction and it uses fixed denominations, meaning that each Coinjoin mixing allows for values in the clear but enforces the same denomination in order to preserve privacy. We note that the mixing output is an address that belongs to the sender, i.e. a user sends coins back to themselves through a mixing process before they perform an actual payment.

We now present a PrivateSend transaction on one user's side (we use the notion of wallet to denote one user). On a high level, it follows the following steps:

1. The user initiates a PrivateSend transaction and chooses the amount to be mixed. They also pick the number of Coinjoin rounds they wish. More rounds correspond to higher privacy guarantee.
2. The wallet checks if there are enough funds, picks the required addresses that sum to the amount and initiates a denomination creation transaction. This transaction breaks the coins into the following denominations: 0.0100001, 0.100001, 1.00001, 10.0001 and 100.001 DASH until no more partition is possible.
3. At the same time, during the denomination creation transaction, the wallet allocates a collateral amount. The computation of the collateral amount is probabilistic and is also a function of the number of requested rounds.
4. For each denomination value, the wallet repeats the following process: picks at random one of the available Masternodes sends a private message with the number of inputs and the denomination to be mixed.
5. The Masternode waits until mixing requests from at least 3 different users are collected. Once this step is complete, they message back the corresponding wallets with a confirmation.
6. The wallet waits until they receive a confirmation message by the Masternode. It generates fresh output addresses (as many as the inputs it is going to include in this mixing) and relays a message back with the input and output addresses.
7. Once the Masternode collects all the data from all participating users, it forms the transaction and sends it back to the them.
8. Once the user's wallet receives the Coinjoin transaction data, it signs using each of the inputs' secret keys and sends back the signatures to the Masternode.
9. When all participating wallets complete their signatures, the Masternode submits the transaction to the pool in order to get posted in a future block. A separate collateral reduction transaction might get posted after a number of Coinjoin transactions. It is not immediately correlated to the Coinjoin transaction itself.
10. The wallet repeats the process for the rest of the coins. For each denomination and each round, it picks a new Masternode at random.

The collateral. For all participants, one in every 10 rounds, their collateral address will appear on the blockchain as a transaction input with a new transaction output that has reduced value. The value difference will be published as fees. This works an incentive for users to behave honestly. If they abort, there is a high chance for their collateral to be reduced even if they do not receive back any complete mixing service.

4 Standard clustering algorithms and heuristics

4.1 Clustering heuristics

- Common-input-ownership heuristic: if a transaction has more than one input then all those inputs are owned by the same entity.
- Address reuse: Using same address for multiple transactions (opposite of change address).
- Change transaction/change address: Spending input in its entirety.
- Heuristic: If an output is the first to send value to an address, it is potentially the change.
- Heuristic: A round number is typically NOT the change address. In other words, when users transfer bitcoins between their own wallets, they are likely to choose values that are powers of ten. On the other hand, it is extremely unlikely that you receive power of ten change due to a wallet's coin selection.
- Unnecessary input heuristic: if a transaction has multiple inputs, the payment (i.e. not the change) address shouldn't be less than one input alone. In other words, if there exists an output that is smaller than any of the inputs it is likely the change. If a change output was larger than the smallest input, then the coin selection algorithm wouldn't need to add the input in the first place.
- Heuristic: Merging two large clusters is rare (i.e. large cluster-slow growth property)
- Heuristic: No change address (i.e. exact payment amount) indicates that both addresses belong to same user.
- Heuristic: Batch payments (i.e. many inputs-outputs in the same transaction) likely indicates an exchange or other large organization.
- Peeling chain: Usage pattern from change address. Single address begins with relatively large amount of coins, and small amount is "peeled off" to one address, and remainder is transferred to another address.
- In a peeling chain, the change output is the output that continues the chain. Note: This heuristic depends on the outputs being spent to detect change. If an output has not been spent, it is considered a potential change output.

References:

- `https://github.com/citp/BlockSci/blob/master/src/heuristics/change_address.cpp`
- `https://en.bitcoin.it/wiki/Privacy#Change_address_detection`

mixing value clustering $C_S \leftrightarrow C_M$

1. C_S = the input of a denomination creating tx
2. C_M = the set of input addresses in a merging tx
3. assign the max sum of C_M values to the max C_S address, remove both and repeat

Algorithm 1 Mixing value clustering with threshold value v_t.

$C_M, C_S \leftarrow \emptyset$
$Cl \leftarrow \emptyset$ {Holds clustered addresses}
for tx_i in B **do**
  if $v_i \geq v_t$ **then**
    if $v_i.type =$ `CreateDenom` **then**
      $C_S \leftarrow C_S \cup \mathsf{tx}_i$
    end if
    if $v_i.type =$ `Merge` **then**
      $C_M \leftarrow C_M \cup \mathsf{tx}_i$
    end if
  end if
end for
while $C_M \neq \emptyset \wedge C_S \neq \emptyset$ **do**
  $m \leftarrow$ FindMaxValAndRemove(C_M)
  $s \leftarrow$ FindMaxValAndRemove(C_S)
  $Cl \leftarrow Cl \cup (m, s)$
end while

Algorithm 2 Splitting tx clustering

$j = 0$
for tx_i in $B \wedge tx_i$ not in $Cl \wedge tx_i.type =$ `CreateDenom` **do**
  $c_{S,j} = \mathsf{tx.in} \cup \mathsf{tx.out}$
  $j++$
end for

general clustering **Splitting txs** (outputs with specific values)

1. $C_S = \{in.addr[i]\} \cup \{out.addr[i]\}$, for $out.addr[i]$ with value in the CJ denomination set

Merging txs (multiple inputs, under the assumption that multisigs are not common)

1. $\mathsf{ch} = \emptyset$, try to identify the change address ch

2. if there is one *out.addr* and the inputs are in CJ denominations, then (this probably was just merging, no payment) ch $\leftarrow$ *out.addr*
3. if there are 2 out addresses and one is identified as public address that belongs to a merchant/ exchange company etc, mark the other one as ch
4. if exists $out.addr[i]$ has value smaller than all inputs, ch $\leftarrow$ $out.addr[i]$
5. else if ....??
6. $C_M = \{in.addr[i]\} \cup$ ch

if one of C_S, C_M intersects with any address in the merged sets of **mixing value clustering**, take the union of all sets

Panos: Find if exchanges do mixing. Same for masternodes. **Panos:** Cluster based on mixing freqency.

Removing 2-round data look for collateral 0.004 $\rightarrow$ 0.003 $\rightarrow$ 0.002(U) or 0.002 spent otherwise (not reduced by 0.001)

- look for merging and splitting of the same value (round 1 inputs, round 2 outputs)
- to trace round 1 outputs (or round 2 inputs), exclude from your search: (U) addresses,output addresses that are not inputs in a CJ, output address that go into a CJ but no 0.003 $\rightarrow$ 0.002 tx follows in the same or next X blocks
- Remove all associated addresses from the space before moving to 4-round clustering

Dash-based clustering, rounds 1,4

- if there is a collateral tx 0.004 $\rightarrow$ 0.003 in the same block or the next ?? blocks **Ioanna:** *how many blocks does one round usually take* , assign inputs to their C_S cluster
- if there is a collateral tx 0.001 $\rightarrow$ 0 in the same block or the next ?? blocks and there are spent (S) outputs, assign them to their C_M cluster

Dash-based clustering, rounds 2,3

- trace $addr1(0.004) \rightarrow addr2(0.003) \rightarrow addr3(0.002) \rightarrow addr4(0.001) \rightarrow addr5(0)$ collateral
- assign $addr1$ to a C_S cluster and mark a set of blocks for each user
- next steps????

Dash-based clustering, aborts and not available masternodes ? **Ioanna:** *identify behavior*

5 Conclusion and Future work

We conclude this report with some interesting future directions for further deanonymizing dash.

Collateral Heuristic: In Section 3, we briefly mentioned the use of a collateral amount, i.e. an amount that is withheld from the user's initial funds before the mixing happens. Although the exact initial collateral amount is a function of multiple parameters and an amount reduction happens only probabilistically once every 10 Coinjoins, we believe we can use the collateral initialization and reduction to derive the number of chosen rounds with high confidence for each user (note that this information is not publicly observable in the Dash blockchain). By inferring the number of mixing rounds with good probability, a probabilistic analysis of transaction inputs/outputs along the transaction graph is made possible (i.e. without learning the number of mixing rounds, any analysis without a known ending point would run indefinitely).

As an extension of this work, we would like to apply our clustering methods to the whole blockchain data. In order to achieve that in a reasonable time frame, we also intend to optimize our code (found in Appendix), starting with an optimized version of the union find algorithm that is repeatedly called throughout the attack.

Finally, we would like to further explore attacks on mixing protocols. More specifically, in [10], the authors describe the "Cluster Intersection Attack" over mixing protocols that takes as input: a set of mixed coins C known to be controlled by the same user and an integer r, representing the adversary's (possibly incorrect) assumption that the victim did at most r rounds of mixing. They design a simulation to implement this attack. We believe that by applying the collateral heuristic of Section 5, we can derive r with high confidence and implement the attack on real blockchain data, instead of simulated.

References

1. Coinjoin. `https://bitcointalk.org/?topic=139581`, accessed: 05/02/2021
2. Darkcoin-fork. `https://github.com/dashpay/dash/commit/c1d622110cc2053de028a7929f99be570bbdca3a`, accessed: 05/02/2021
3. Dash-rebranding. `https://github.com/dashpay/dash/commit/35bb210c6b8a782bdccf0dc997b924e6b9130bac`, accessed: 05/02/2021
4. Alsalami, N., Zhang, B.: Sok: A systematic study of anonymity in cryptocurrencies. In: 2019 IEEE Conference on Dependable and Secure Computing (DSC). pp. 1–9. IEEE (2019)
5. Androulaki, E., Karame, G.O., Roeschlin, M., Scherer, T., Capkun, S.: Evaluating user privacy in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 34–51. Springer (2013)
6. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
7. Caprolu, M., Pontecorvi, M., Signorini, M., Segarra, C., Di Pietro, R.: A novel framework for the analysis of unknown transactions in bitcoin: Theory, model, and experimental results. arXiv preprint arXiv:2103.09459 (2021)
8. Dingledine, R., Mathewson, N., Syverson, P.: Tor: The second-generation onion router. Tech. rep., Naval Research Lab Washington DC (2004)

9. Fanti, G., Viswanath, P.: Anonymity properties of the bitcoin p2p network. arXiv preprint arXiv:1703.08761 (2017)
10. Goldfeder, S., Kalodner, H., Reisman, D., Narayanan, A.: When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. arXiv preprint arXiv:1708.04748 (2017)
11. Harrigan, M., Fretter, C.: The unreasonable effectiveness of address clustering. In: 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld). pp. 368–373. IEEE (2016)
12. Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., Goldberg, S.: Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In: Network and Distributed System Security Symposium (2017)
13. Kappos, G., Yousaf, H., Maller, M., Meiklejohn, S.: An empirical analysis of anonymity in zcash. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 463–477 (2018)
14. Mastan, I.D., Paul, S.: A new approach to deanonymization of unreachable bitcoin nodes. In: International Conference on Cryptology and Network Security. pp. 277–298. Springer (2017)
15. Meiklejohn, S., Orlandi, C.: Privacy-enhancing overlays in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 127–141. Springer (2015)
16. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the 2013 conference on Internet measurement conference. pp. 127–140 (2013)
17. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: 2013 IEEE Symposium on Security and Privacy. pp. 397–411. IEEE (2013)
18. Möser, M., Böhme, R., Breuker, D.: An inquiry into money laundering tools in the bitcoin ecosystem. In: 2013 APWG eCrime researchers summit. pp. 1–14. Ieee (2013)
19. Reid, F., Harrigan, M.: An analysis of anonymity in the bitcoin system. In: Security and privacy in social networks, pp. 197–223. Springer (2013)
20. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: International Conference on Financial Cryptography and Data Security. pp. 6–24. Springer (2013)
21. Ruffing, T., Moreno-Sanchez, P.: Valueshuffle: Mixing confidential transactions for comprehensive transaction privacy in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 133–154. Springer (2017)
22. Ruffing, T., Moreno-Sanchez, P., Kate, A.: P2p mixing and unlinkable bitcoin transactions. In: NDSS. pp. 1–15 (2017)
23. Xi, H., Ketai, H., Shenwen, L., Jinglin, Y., Hongliang, M.: Bitcoin address clustering method based on multiple heuristic conditions. arXiv preprint arXiv:2104.09979 (2021)

A Clustering implementation

In this section we provide the code of our Python implementation for the clustering techniques discussed in this report. Our code also requires setting up a Dash cryptocurrency full-node endpoint to reply the necessary queries.

```
import requests
import time

EP = "http://127.0.0.1:3001/insight-api/" #network endpoint for Dash full node

ps_denoms = [100001000,10000100,1000010,100001,10000.1]
#denomination values used in mixing

def getBlockHash(blocknum):
'''
Outputs a block hash given a block number
'''
url = EP + 'block-index/' + str(blocknum)
return requests.get(url).json()['blockHash']

def getBlockData(blocknum,page=0):
'''
Fetches a block's data given a block number.
'''
url = EP + 'txs/?block=' + getBlockHash(blocknum)+'&pageNum=' + str(page)
return requests.get(url).json()

def getTxfromBlock(blocknum):
'''
Outputs a list of transactions given a block number.
'''
pages = getBlockData(blocknum)['pagesTotal']
listoftxs = []
for i in range(0,pages):
listoftxs += getBlockData(blocknum,i)['txs']
return listoftxs

def countAddrinTxlist(txlist,addrset = set()):
'''
Parses a list of transactions in a block and adds all unique addresses
to a set.
'''
for tx in txlist:
for intx in tx['vin']:
if 'addr' in intx:
```

```
addrset.add(intx['addr'])
for outtx in tx['vout']:
if 'addresses' in outtx['scriptPubKey']:
addrset.add(outtx['scriptPubKey']['addresses'][0])
return addrset

def getIOfromTxlist(txlist,idx,io):
'''
Returns a list of input or output transcations given a transaction list.
'''
if io == 'in':
return txlist[idx]['vin']
elif io == 'out':
return txlist[idx]['vout']


def largeValueDenom(txlist,clusterlist=[]):
'''
Given a transaction list, prints large values (>100) that are denominated.
Returns True if such a transaction was found.
'''
for tx in txlist: #iterate through all transactions in block
if (len(tx['vin']) == 1) and ('value' in tx['vout'][0]):
#if a transaction is splitting into denoms
#print(int(float(tx['vout'][0]['value'])*100000000) in ps_denoms)
if ('value' in tx['vin'][0]):
#print(float(tx['vin'][0]['value']))
if ((int(float(tx['vout'][0]['value'])*100000000) in ps_denoms)
and (float(tx['vin'][0]['value']) > 100)):
print("")
print("")
print("Large value denominated: "
+ str(float(tx['vin'][0]['value'])))
print("Block id: " + str(tx['vin'][0]['txid']))
return True
#print(float(tx['vin'][0]['value']))
#print(float(tx['vout'][0]['value']))
return False

def largeValueMerge(txlist,clusterlist=[]):
'''
Given a transaction list, prints large values (>100) that are merged
from denominations. Returns True if such a transaction was found.
'''
for tx in txlist: #iterate through all transactions in block
```

```
if ((len(tx['vout']) == 1) and ('value' in tx['vout'][0])
and ('value' in tx['vin'][0])):
#if a transaction is splitting into denoms
if ((int(float(tx['vin'][0]['value'])*100000000) in ps_denoms)
and (float(tx['vout'][0]['value']) > 10)):
print("")
print("")
print("Large value merged: " +str(float(tx['vout'][0]['value'])))
print("Block id: " + str(tx['vin'][0]['txid']))
return True
return False

def largetxsearch(startblock,endblock):
'''
Searches for large value merging or denomination within a block range.
'''
#largetxsearch(1460260,1460500)
starttime = time.time()
for blocknum in range(startblock,endblock):
print("Parsing block " + str(blocknum) + '/' + str(endblock)+
" Time elapsed: " + str(time.strftime('%H:%M:%S',
time.gmtime(time.time()-starttime))) + ' , ETA:' +
str( time.strftime('%H:%M:%S',
time.gmtime((time.time()-starttime)*
(endblock-startblock)/(blocknum+1-startblock)))) +
' sec', end='\r', flush=True)
txlist = getTxfromBlock(blocknum)
if largeValueDenom(txlist) or largeValueMerge(txlist):
print(blocknum)

def cmInputClustering(txlist,clusterlist=[]):
'''
Given a transaction list and a list of clustered addresses,
runs the common input heuristic and adds them to the cluster.
'''
for tx in txlist: #iterate through all transactions in block
if (len(tx['vin']) > 1) and (tx['vin'][0]['valueSat'] not
in ps_denoms): #ignore mixing txs
cluster = []
for addr in tx['vin']:
cluster.append(addr['addr'])
cluster = list(set(cluster)) #remove duplicates
if len(cluster)>1: clusterlist.append(cluster)
return clusterlist
```

```
def singleOutputClustering(txlist,clusterlist=[]):
'''
Given a transaction list and a list of clustered addresses, runs
the single output heuristic and adds them to the cluster.
'''
for tx in txlist: #iterate through all transactions in block
if (len(tx['vout']) == 1) and ('addresses' in
tx['vout'][0]['scriptPubKey']): #if single output
cluster = [tx['vin'][0]['addr'],
tx['vout'][0]['scriptPubKey']['addresses'][0]]
cluster = list(set(cluster)) #remove duplicates
if len(cluster)>1: clusterlist.append(cluster)
return clusterlist

def changeAddressClustering(txlist,clusterlist=[]):
'''
Given a transaction list and a list of clustered addresses,
runs change address heuristics and adds them to the cluster.
'''
for tx in txlist: #iterate through all transactions in block
if (len(tx['vout']) == 2) and ('addr' in tx['vin'][0]):
#if two outputs in a tx
#always clusters with first input, but if multiple inputs
# they will be clustered as well in the end
if tx['vout'][0]['value'][-3:] == '000': #first output has
#round value (last three digits are zero),
#so second is change address
#so cluster second output with input
cluster = [tx['vin'][0]['addr'],
tx['vout'][1]['scriptPubKey']['addresses'][0]]
elif tx['vout'][1]['value'][-3:] == '000':
cluster = [tx['vin'][0]['addr'],
tx['vout'][0]['scriptPubKey']['addresses'][0]]
elif tx['vout'][0]['value'] > tx['vout'][1]['value']:
#if first output is larger, it is change address
#so cluster first output with input
#maybe introduce a factor?
cluster = [tx['vin'][0]['addr'],
tx['vout'][0]['scriptPubKey']['addresses'][0]]
else:
cluster = [tx['vin'][0]['addr'],
tx['vout'][1]['scriptPubKey']['addresses'][0]]
cluster = list(set(cluster)) #remove duplicates
if len(cluster)>1: clusterlist.append(cluster)
return clusterlist
```

```
def findclusters(startblock,finishblock):
'''
Given a range of blocks, parses the blockchain, runs all available
heuristics and outputs the resulting clusters and all parsed addresses.
'''
clusters = []
alladdresses = set()
starttime = time.time()
for blocknum in range(startblock,finishblock):
print("Parsing block " + str(blocknum) + '/' + str(finishblock)+
" Time elapsed: " + str(time.strftime('%H:%M:%S',
time.gmtime(time.time()-starttime))) +
' , ETA:' +str( time.strftime('%H:%M:%S',
time.gmtime((time.time()-starttime)*(finishblock-startblock)
/(blocknum+1-startblock)))) + ' sec', end='\r', flush=True)
txlist = getTxfromBlock(blocknum)
countAddrinTxlist(txlist,alladdresses)
clusters = cmInputClustering(txlist,clusters)
clusters = singleOutputClustering(txlist,clusters)
clusters = changeAddressClustering(txlist,clusters)
return [clusters,alladdresses]

def mergeclusters(clusterlist):
'''
Given a list of clusters, finds intersecting elements among those clusters
and outputs the unioned clusters  when such intersections occur.
'''
# Make a list of all values, then put them into a separate set
vals = sum(clusterlist, [])
vals = list(map(lambda x: {x}, set(vals)))
for item in map(set, clusterlist):
# Find sets that intersect with at least one value with sublist
vals_in = [x for x in vals if x & item]
# Output all dosjoint sets with sublist
vals_out = [x for x in vals if not x & item]
# Find union of non-disjoint sets and output them into one set
vals_in = set([]).union(*vals_in)
if vals_in:
vals = vals_out + [vals_in]
return [list(x) for x in vals]

def statistics(startblock,finishblock):
'''
Outputs statistical information for clustered addresses
```

```
given a range of blocks.
'''
res = findclusters(startblock,finishblock)
print("")
print("")
print("Total clusters: " + str(len(res[0])) )
print("Total clustered addresses: " +
str(len(list(map(lambda x: {x}, set(sum(res[0],[])))))) )
print("Total addresses: " + str(len(res[1])) )
```