A Traceability Analysis of Monero's Blockchain

April 17, 2017

Amrit Kumar
National University of Singapore
amrit@comp.nus.edu.sg

Clément Fischer
National University of Singapore
cfischer@comp.nus.edu.sg

Shruti Tople
National University of Singapore
shruti90@comp.nus.edu.sg

Prateek Saxena
National University of Singapore
prateeks@comp.nus.edu.sg

ABSTRACT

Monero is a cryptocurrency that has rapidly gained popularity since its launch in April 2014. The source of its growth can be mainly attributed to its unique privacy properties that go well beyond the pseudonymity property of cryptocurrencies such as Bitcoin.

In this work, we conduct a forensic analysis of the Monero blockchain. Our main goal is to investigate Monero's *untraceability* guarantee, which essentially means that given a transaction input, the real output being redeemed in it should be anonymous among a set of other outputs. To this end, we develop three heuristics that lead to simple-to-implement attack routines.

We evaluate our attacks on the Monero blockchain and show that in 87% of cases, the real output being redeemed can be easily identified with certainty. Moreover, we have compelling evidence that two of our attacks also extend to Monero RingCTs — the second generation Monero that even hides the transaction value.

Furthermore, we observe that for over 98% of the inputs that we have been able to trace, the real output being redeemed in it is the one that has been on the blockchain for the shortest period of time. This result shows that the mitigation measures currently employed in Monero fall short of preventing temporal analysis. Motivated by our findings, we also propose a new mitigation strategy against temporal analysis. Our mitigation strategy leverages the real spending habit of Monero users.

1 INTRODUCTION

Distributed e-cash systems such as Bitcoin have seen widespread adoption and popularity. Both professionals and the general public have come to appreciate the convenient combination of public transactions and proof-of-work as a trust model.

However, as outlined by Chaum in the first e-cash proposal [3], privacy and anonymity properties remain important desiderata for any e-cash system. In fact, Bitcoin, the most popular cryptocurrency fairs poorly in terms of privacy and anonymity as evidenced by several analyses in the past [6, 11, 14, 16]. These works show that it is possible to build a large part of the Bitcoin transaction graph that would show which entity pays what amount to whom. These works have further provided an impetus to the industry to develop automated tools to perform a forensic analysis of the Bitcoin's public *blockchain* [5].

Considering the privacy issues in Bitcoin, a new cryptocurrency called *Monero* was launched on April 18^{th} 2014. Monero addresses the privacy issues by requiring the currency to ensure the following two properties (definitions are informal):

(1) **Unlinkability:** For any two transactions, it should be impossible to prove that they were sent to the same person.
(2) **Untraceability:** Given a transaction input, the real output being redeemed in it should be anonymous among a set of other outputs.

In order to guarantee unlinkability, Monero by design introduces the notion of *one-time random addresses*. The idea is that each sender of a transaction generates a new one-time random address for the recipient in a way that only the recipient can spend it using a long-term secret key. If each address is generated using fresh randomness and is used only once, then it should be hard for an adversary to link two addresses. Monero enforces untraceability using a cryptographic primitive called *ring signatures* [7, 15]. The primitive allows a sender (also the signer) to anonymously sign the transaction (the message) on behalf of a set of other users. As a result, the real output being redeemed remains anonymous amongst the chosen set of outputs belonging to other users. The guarantee that Monero aims to achieve is the privacy *à la anonymity-set size*.

While there are several other cryptocurrencies that have implemented privacy solutions (*e.g.*, Zcash [1, 20], and Dash [4]) and also a few proposals for implementations of privacy features on top of the largest blockchains (*e.g.*, Mimblewimble [8] and Confidential transactions [10]), Monero currently has the most momentum of all the live privacy projects[1]. The value of Monero in terms of USD shot up by around 27 fold in the year 2016 from its value in 2015. Its rise to popularity is mainly due to its strong privacy properties and its design simplicity. Since January 2017, Monero has further strengthened its privacy guarantees by incorporating *ring confidential transaction* (RingCTs) [13]. In addition to hiding the real output being redeemed, RingCTs also hide the transaction value.

In this work, we present a forensic analysis of the Monero blockchain to test the adverted untraceability guarantees. We assume throughout this work a global passive adversary who has access to the public blockchain data. Our study takes a two dimensional approach, where, we first design our attack routines and then evaluate it on Monero blockchain data.

Contributions & Findings. We summarize our contributions and findings below:

(1) In order to set the ground for our traceability analysis, we first conduct a study of the network and usage statistics

[1] https://thecontrol.co/meet-the-best-performing-digital-currency-of-2016-monero-e6010768e54a

on Monero. This gives us several interesting insights into how Monero is used and its ensuing privacy impact. One of the most important observations that we make here is that over 65% of inputs have an anonymity-set size of one. This means that these inputs are traceable by default.

(2) Next, we leverage our statistical findings to develop three heuristics that allow us to mount traceability attacks. Our first heuristic (Heuristic I) leverages the fact that over 65% of inputs are traceable due to zero mix-ins. We show that such inputs in fact lead to a *cascade effect*, where they affect the untraceability of other inputs with which they have a non-empty anonymity-set intersection. Our evaluation shows that the cascade effect renders another 22% of input traceable. Moreover, results from Heuristic I serve as a ground truth as it only yields true positives. Unfortunately, we do not obtain any ground truth on RingCTs.

(3) Our second heuristic (Heuristic II) exploits the fact that several outputs from a previous transaction are often merged to aggregate funds when creating a new transaction. Such idioms of use leak information on the real outputs being redeemed. We employ Heuristic II on both RingCTs and non RingCTs. On non RingCTs, Heuristic II has a true positive rate of 95%. Due to its high true positive rate, we believe that it should extrapolate well even to RingCTs. Moreover, Heuristic II also identifies merging of outputs in 1% of RingCTs.

(4) Our third heuristic (Heuristic III) considers an attack based on the temporal analysis of transaction outputs. In fact, Heuristic III considers the most recent output (in terms of block height) in the anonymity-set as the real one being redeemed. Surprisingly, Heuristic III has a true positive rate of 98.5% on non RingCTS. We believe that it should also extrapolate well even to RingCTs.

(5) Motivated by the results of Heuristic III, we also propose a better method to choose outputs to include in the anonymity-set. Our method takes into account the actual spending habit of users.

It is important to note that Monero designers and developers are aware of the theoretical possibility of our attacks [9] and have put in place some measures to mitigate the risks. In light of [9], our work has three goals:

(1) To show that the attacks entail huge risks in practice and are not merely theoretical.
(2) The existing measures often fall short in mitigating the risks.
(3) Propose better mitigation strategies.

Roadmap. This paper takes the following roadmap. In Section 2, we present the essential background on Monero. In Section 3, we present network and usage statistics on Monero. This allows us to understand how Monero is currently used and gives an insight into the ensuing privacy implications. Section 4 presents the general methodology to our traceability analysis and in Section 5, we develop heuristics to break the untraceability guarantees in Monero. Lastly, in Section 6, we present the related work.

2 MONERO BACKGROUND

Monero (XMR) is a privacy-by-design cryptocurrency. It is based on the CryptoNote protocol [19] that attempts to solve the traceability and linkability issues in Bitcoin. Monero incorporates a mixing protocol that is *autonomous* and *spontaneous*. The system is autonomous as a user can mix his coins on its own and spontaneous since the mixing process does not incur any latency.

In this section, we present the essential background on Monero. The material provided here is largely a presentation of the CryptoNote protocol. Interested readers may refer to [19] for further details. In the following, we first list the Monero system parameters and then present the details on how Monero achieves unlinkability and untraceability.

2.1 System Parameters

As in Bitcoin, transactions in Monero are signed by the sender using a digital signature scheme, thereby, authorizing the transfer of funds to the receiver. Monero's digital signature algorithm employs the elliptic curve Ed25519, popularized by Bernstein *et al.* [2]. Table 1 abstractly presents the curve parameters. The table also includes two special hash functions $\mathcal{H}_s$ and $\mathcal{H}_p$ employed in Monero. We refer the interested readers to the CryptoNote whitepaper [19] for a detailed construction of the hash functions and the instantiations of the parameters. In the rest of the paper, we use the parameters in the notational form while abstracting their actual instantiations.

Table 1: Monero system parameters.

q:	a large prime number;
$\mathbb{F}_q$:	a finite field of order q;
$E(\mathbb{F}_q)$:	an elliptic curve on $\mathbb{F}_q$;
G:	a base point on $E(\mathbb{F}_q)$;
ℓ:	a prime order of G;
$\mathcal{H}_s$:	a cryptographic hash function $\{0,1\}^* \rightarrow \mathbb{F}_q$;
$\mathcal{H}_p$:	a cryptographic hash function $E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_q)$.

Each Monero user has a long-term public, private key pair that we denote by $(pk_{\text{LT}}^{\text{user}}, sk_{\text{LT}}^{\text{user}})$. A long-term public (resp. private) key is a pair of classical public (resp. private) keys of an elliptic curve cryptosystem:

$$sk_{\text{LT}}^{\text{user}} = (a, b), \text{ where, } a, b \xleftarrow{\$} [1, \ell - 1],$$
$$pk_{\text{LT}}^{\text{user}} = (A, B), \text{ where, } A = aG \text{ and } B = bG.$$

The long-term public key can be made public without the risk of sacrificing privacy.

2.2 Ensuring Unlinkability

Contrary to the Bitcoin's model, where a user possesses a unique public, private key pair (corresponding to an address), in Monero, a sender generates a *one-time public key* based on the recipient's address and some randomness. In this sense, transactions destined to the same recipient are in fact sent to different one-time public keys (not directly to a unique address) and only the recipient can recover the one-time private keys to redeem the funds.

More concretely, when a user say Alice wishes to pay to another user Bob, she first asks for Bob's long-term public key $pk_{\text{LT}}^{\text{Bob}} =$

(A, B). She then generates a one-time public key $pk_{\text{OT}}^{\text{Bob}}$ for Bob in the following manner:

$$\begin{bmatrix} \text{Choose } r \xleftarrow{\$} [1, \ell - 1], \\ \text{Set } R \leftarrow rG \text{ and } P \leftarrow \mathcal{H}_s(rA)G + B, \\ \textbf{return } pk_{\text{OT}}^{\text{Bob}} = P. \end{bmatrix}$$

The value R referred to as the *transaction public key* is made public by including it in the transaction. Note that:

$$\begin{aligned} P &= \mathcal{H}_s(rA)G + B = \mathcal{H}_s(raG) + B \\ &= \mathcal{H}_s(aR)G + bG = (\mathcal{H}_s(aR) + b)G. \end{aligned} \tag{1}$$

Hence, the corresponding one-time secret key $sk_{\text{OT}}^{\text{Bob}} = \mathcal{H}_s(aR) + b$ (Cf. Equation 1). Since, only Bob knows (a, b) as $sk_{\text{LT}}^{\text{Bob}} = (a, b)$, only he can derive $sk_{\text{OT}}^{\text{Bob}}$. Alice then creates a transaction, where, she pays to $pk_{\text{OT}}^{\text{Bob}}$. The one-time key serves as a single-use payment address for Bob. Since, an address is always generated using fresh randomness and is used only once, it is hard for anyone other than the sender and the receiver to link two addresses to the same user.

In case there are multiple outputs (in the transaction) paying to the same recipient, a new one-time public key is generated per output. As a result, each transaction output (TXO) can then be identified by its corresponding one-time public key.

Bob can easily identify whether a TXO belongs to him by using a, B and R. It suffices for Bob to check whether a, B and R when combined generate P (Cf. Equation 1). If so, the TXO (identified by P) belongs to him. To spend the funds received, Bob first recovers R and derives $sk_{\text{OT}}^{\text{Bob}}$. He can then use these data to redeem funds in a later transaction using a procedure presented in the next section.

In order to lighten the notation, we will henceforth refrain from using pk_{OT} to denote a one-time public key (identifying a TXO), and instead used P with an eventual subscript. Similarly, the corresponding secret key sk_{OT} will henceforth be denoted by s with an eventual subscript. Under this shorthand, $P = sG$.

2.3 Ensuring Untraceability

Monero aims to ensure untraceability using a cryptographic primitive called *ring signatures*. Monero uses a modified version of the ring signature proposed by Fujisaki *et al.* [7]. A ring signature allows a user to sign a message on behalf of a "ring" of users. In order to do so, the signer only needs to know his own secret signing key. After signing the message, the user provides (for verification purposes) not his own public key but the public keys of all the other users in the ring. A verifier is convinced that the real signer is a member of the ring but cannot ascertain its identity. The signer is hence anonymous in the ring that forms its *anonymity-set*.

In order to see how Monero employs ring signatures, let us consider an example where a user Alice wishes to send 10 XMR to another user Bob. She first asks for $pk_{\text{LT}}^{\text{Bob}}$ and then creates a one-time public key from it using some randomness (as discussed in Section 2.2). Now, instead of signing the transaction as in Bitcoin, Alice takes some other TXOs available on the blockchain of value 10 XMR. Recall that these TXOs are identified by their corresponding one-time public keys. Let this set be $\mathcal{S} = \{P_1, \ldots, P_m\}$; it also includes Alice's own input $P_t = s_t G$, for $t \in [1, m]$. Her input key P_t will then be anonymous among the keys in $\mathcal{S}$. Since, Alice knows s_t, she can create a ring signature and sign the transaction.

Now, when an adversary (possibly the recipient or a miner) sees the transaction, she may only conclude that a TXO corresponding to one of the input keys from $\mathcal{S}$ was spent but cannot ascertain the real one. The extra input keys that Alice includes in $\mathcal{S}$ are referred to as the *mix-ins*. Since Alice's real input key is anonymous among the keys in $\mathcal{S}$, larger is the number of mix-ins used, the better is the anonymity achieved. Clearly, the ring signature provides an in-built mixing service within Monero, where each user can mix his coins in an autonomous manner.

However, with only a ring signature, it becomes possible for Alice to carry out a double spend. It suffices to chose another set of mix-ins and create a new transaction with a new ring signature. Miners cannot detect the double spend since they do not know the TXO being redeemed in a transaction. In order to prevent this, Monero requires the signer to include a special data in the transaction called the *key image* which is $\mathfrak{I} = s\mathcal{H}_p(P)$, where $P = sG$ is the signer's one-time public key. The key image creates a deterministic tag for the signer's input key. Signing using the same key twice, will generate the same tag and hence double spending can be detected. Note that this requires modifying the ring signature to prove in NIZK that one of the keys in $\mathcal{S}$ indeed generates $\mathfrak{I}$. The key image is is made available to miners by including it in the transaction. Linking a key image to an input key in $\mathcal{S}$ is not feasible since the key image uses the secret one-time key s known only to the signer.

In case there are multiple inputs in the transaction, a ring signature and the corresponding key image are required for each input. Figure 1 schematically presents a typical Monero transaction. Each TXO also includes a transaction public key, R (not shown).

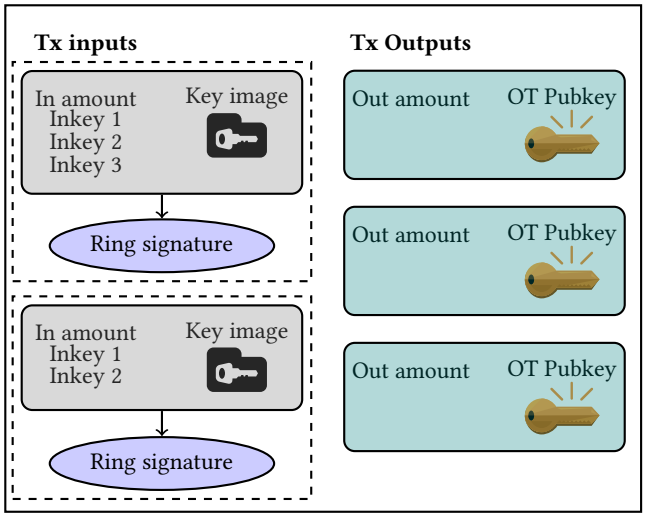

Figure 1: A schematic representation of a typical Monero transaction. It has two inputs and three outputs. The first input uses two mix-ins (hence three input keys), while the second one uses one. The sum of all output amounts must be equal to the sum of all input amounts. Ring signatures hide the real input key corresponding to the TXO being spent.

2.4 RingCTs

Recently, on January 10^{th} 2017, Monero launched a special type of transaction called *ring confidential transactions* (RingCTs). As in a regular transaction, a RingCT hides the real TXO being spent, but additionally, it also hides the amount in a TXO. In fact, every TXO in a RingCT is of value '0' (meaning unknown). The construction uses commitments to hide the output amount. Moreover, the commitment scheme is homomorphic and allows miners to verify whether the sum of input commitments is equal to the sum of output commitments [13]. This prevents the creation of new coins. It is important to note that with the advent of RingCTs, a TXO can be mixed with any other output (for creating the ring signature) on the blockchain as all TXOs now have the same value.

RingCT is currently in its experimental phase and has been incorporated in Monero as a soft fork. It is slated to be introduced as a hard fork in September 2017.

3 MONERO NETWORK STATISTICS

In this section, we present network and usage statistics on Monero. As we show, the statistics reveal interesting insights into how Monero is currently used and its ensuing privacy impact. The results shown here are important as they form the basis of our traceability attacks presented in Section 5.

Below, we present the dataset for our statistical analysis. The same dataset will also be used to evaluate the impact of our traceability attacks later in Section 5.

3.1 Dataset

We acquired the entire Monero history from the first transaction on April 18^{th} 2014 up to and including the last transaction on February 6^{th} 2017. The last transaction is included in the block with height 1,240,503. The resulting dataset comprises of a total of 961,463 non-coinbase transactions included in 418,910 blocks. The first non-coinbase transaction appears in the block with height 110. It is important to note that our dataset also includes 47,428 RingCTs. The first RingCT appears in the block with height 1,220,517 (on January 10^{th} 2017). We purposefully decide to include RingCTs so as to have a more representative dataset.

Throughout our analysis, we maintain the raw data in the form of a blockchain and use Monero daemon (`monerod`) to access it. The daemon handles interactions with the Monero blockchain through Remote Procedure Calls (RPC). The RPC interface provides two main methods: `getheight` and `gettransactions`, both returning output in the JSON format [12]. The former returns information on a given block height, while the latter on a list of transaction hashes.

In the remainder of the section, we present results of our statistical analysis on the dataset. Unless otherwise stated, we do not include coinbase transactions. Our analysis focuses on three important aspects: 1) *available liquidity*, where the goal is to know the different output amounts currently available on the blockchain and their frequency. Note that low liquidity implies that it may not always be possible to find sufficient number of mix-ins for a given amount; 2) the number of mix-ins used. Clearly, the larger it is the higher is the anonymity; 3) number of input and outputs in a transaction. Roughly speaking larger is the number of inputs and outputs, the higher is the probability that one of our attacks would work.

3.2 Available Liquidity

There are 1,339,733 different output amounts in the dataset. The largest being 500,000 XMR. We also observe that approximately 85% of all outputs have an amount less than 0.01 XMR ($\approx$ 0.13 USD[2]). We refer to them as *dust values*.

Figure 2 presents the cumulative frequency of the available liquidity. We observe that values less than 1 XMR cover 87% of all the outputs. This shows that lower denominations dominate the blockchain, and users often transact with small amounts.

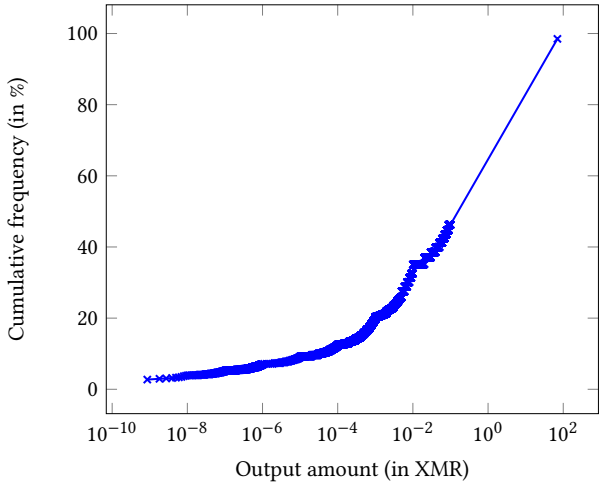

Figure 2: The plot presents the cumulative frequency of output amounts. Only 1 out of every 100 data points are shown.

We further observe that a total of 1,244,165 (93%) output values have a frequency of 1. This means that when these outputs have to be redeemed, they cannot be mixed with any other output. The only possible way to create an untraceable transaction redeeming these outputs is to create a RingCT. These output values sum to a total of 1,012,231.3 XMR. Moreover, 84.5% of these outputs correspond to dust values. The dust values themselves total to 1,723.36 XMR. Lastly, there are five outputs: i) with value greater than 1 XMR, ii) and that appear only once on the blockchain: 1.67 XMR, 3.45 XMR, 200,000 XMR and 300,000 XMR and 500,000 XMR.

Another important observation is regarding the number of output values that do not respect the usual denomination format in Monero. A usual Monero output value is of the form: $A \times 10^B$, where $1 \leq A \leq 9$ and $B \geq -12$ [9]. We found that 99.98% of output values are not in the usual format. Moreover, 92.8% of these values appear only once on the blockchain. The total value of these outputs that appear only once and are non-denomination compliant is 12231.27 XMR. The large number of non-denomination compliant output values can be attributed to the simple fact that denominations per se cannot be enforced in Monero. This is essentially due to the

[2]Source https://www.cryptonator.com/rates/XMR-USD, Accessed on February 23^{rd} 2017.

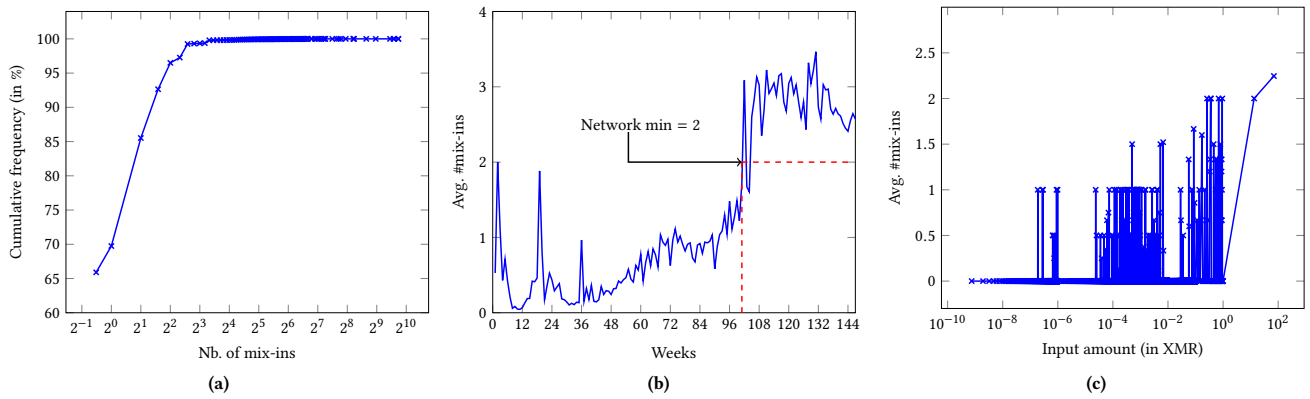

Figure 3: Results on the number of mix-ins. (a): The plot presents the cumulative frequency of the number of mix-ins used in an input of a transaction. (b): The plot presents the variation (with time) of the average number of mix-ins used in a transaction. The x-axis represents the number of weeks after the launch of Monero. (c): The plot presents the average number of mix-ins used for an input amount. Only 1 out of every 100 data points are shown.

underlying design and parameter choices. For instance, the block reward in Monero is often not denomination compliant by the way it evolves over time. The same is true for transaction fee that depends on the transaction size.

3.3 Number of Mix-ins

We first note that different inputs in a transaction may have different number of mix-ins. In fact, a total of 115 different number of mix-ins have been used until now. The minimum and the maximum numbers being 0 and 851 respectively. Figure 3a presents the cumulative frequency of the number of mix-ins used in an input. One may observe that lower number of mix-ins, *i.e.*, 0, 1, 2, 3 and 4 correspond to roughly 96% of all mix-ins. Moreover, 65.9% of all inputs have zero mix-ins.

Figure 3b presents the evolution of the average number of mix-ins used per input over time. The curve shows three distinct regions with noticeable characteristics. The first region starts from the first week and ends roughly in the 46^{th} week. The average number of mix-ins used per input during this period is less than 1. The second region starts from the 47^{th} week and ends in the 101^{st} week. Note that it was in the 101^{st} week (on March 23^{rd} 2016) that Monero developers enforced a network-wide minimum mix-in of 2. As a result, miners reject transactions with number of mix-ins lower than 2 with the exception where a higher mix-in is impossible due to insufficient liquidity. During this period, the average number of mix-ins continued to increase but remained less than 1. The third region starts from the 106^{th} week. It was in this period that the network-wide minimum could actually come into effect. This means that starting from week number 101 and until week number 106, users could not find enough suitable outputs to mix with.

We also observe that there are 9 different mix-in values (7.8%) that are unique in the sense that these many mix-ins are used in only one input (across the entire dataset). Such transactions can be attributed to unique users. In other words, the number of mix-ins used may become an identifying trait of Monero users.

In Figure 3c, we present the average number mix-ins used per input amount. The goal is to know whether users employ a larger number of mix-ins when they spend a larger amount. We observe that this is true to some extent since as the input amount increases, the plot shows some spikes. However, the phenomenon is not consistent as mix-in numbers of 0, and 1 tend to dominate all through the plot.

As we have seen in Figure 3a, smaller number of mix-ins dominate the blockchain. There are two possible explanations to this phenomenon. First, it could be possible that at the time a user creates a transaction, he may not find enough suitable outputs to mix with and hence is forced to choose a lower number of mix-ins. This may indeed happen since majority of the output amounts are non-denomination compliant. Second, enough outputs are indeed available at any given time, but users deliberately choose a low number of mix-ins. The incentive here being a lower transaction fee. Indeed, a larger number of mix-ins implies a larger transaction size and hence a larger transaction fee.

In order to investigate further, we provide in Table 2, the cumulative frequency of the first 11 values from 0 to 10. The table also presents the percentage of cases when it was possible to choose a higher number of mix-ins. In order to compute these data, we include coinbase transactions. It is to note that a coinbase transaction output cannot be used as a mix-in for 60 blocks. We take this fact into our account when computing the data for the last column.

We first observe that the second most frequent number of mix-ins used after 0 is 2. This is mainly due to the network-wide minimum imposed since March 23^{rd} 2016. We now focus on the last column of the table. In the case of inputs with zero mix-ins, 85.9% of them could have been spent using a higher number of mix-ins. As for the rest, over 99% of the inputs could have been spent using a higher number of mix-ins. These results clearly show that users deliberately use a lower mix-in value to avoid paying a larger transaction fee.

Table 2: Cumulative frequency of the number of mix-ins (only the first 11 are shown). The last column counts the number of instances where it was possible to choose a higher number of mix-ins.

Mix-ins	Freq	Cumul. freq. (in %)	Higher #mix-ins possib. (in %)
0	12148623	65.9	10434988 (85.9)
1	707788	69.7	701252 (99.1)
2	2908304	85.5	2902246 (99.8)
3	1313596	92.6	1313530 (99.9)
4	709686	96.5	709681 (99.9)
5	141800	97.3	141797 (99.9)
6	365720	99.2	365718 (99.9)
7	9616	99.3	9614 (99.9)
8	8593	99.3	8593 (100)
9	5369	99.4	5366 (99.9)
10	76524	99.8	76523 (99.9)

3.4 Number of Inputs and Outputs

Our last result is on the number of inputs and outputs in a transaction. It is important to note that the existence of denominations in Monero has a direct impact on the number of outputs that a transaction can have. In order to understand this, let us consider the following example, where, a user Alice has 1000 XMR in one of her addresses and wishes to pay 11 XMR to another user Bob. For the sake of simplicity, let us assume that there is no transaction fee. Since 11 XMR is not denomination compliant, Alice cannot create a transaction with just two outputs where the first output pays 11 XMR to Bob, while the second output sends back a sum of 989 XMR as a change to Alice. Rather, the very best that Alice can do is to create two outputs for Bob for 10 XMR and 1 XMR respectively. Similarly, for the change she can (at the very best) create three change outputs for 900 XMR, 80 XMR and 9 XMR respectively. As a consequence, the number of outputs can become large. This also leads to a large number of inputs per transaction. To see this, consider the situation when Bob wishes to spend 11 XMR that he previously received from Alice. Bob is now forced to create a transaction with two inputs.

Figure 4 presents the evolution of the number of inputs and outputs per transaction. We first observe that the average number of inputs and outputs per transaction are 19 and 17 respectively. These large numbers can be seen as a direct impact of denominations. The general observation is that the number of inputs and outputs were relatively large in the first few weeks and they gradually decreased as the currency matured over time. The number of inputs reaches a maximum of 104 in the fourth week and gradually reached to 3 in the last week. Similarly, the number of outputs has a maximum value of 76 in the sixth week and gradually reached to 2 in the last week. Smaller number of inputs and outputs at the tail can be attributed to RingCTs, which made denominations redundant. With RingCTs, the number of outputs in a transaction can be limited to two: one for the payment to the recipient, the other for the change.

We now focus on the impact of RingCTs on the number of inputs and outputs. On an average, a RingCT has 3.7 inputs and only 1.2 output. Figure 5 shows the evolution of these values over time.

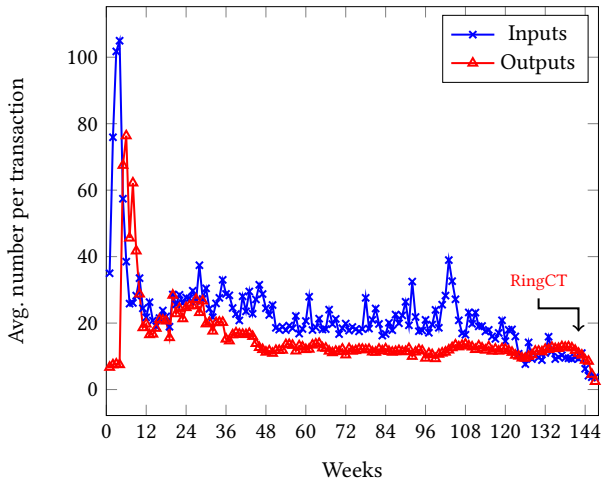

Figure 4: The plot presents the average number of inputs and outputs in a transaction. The x-axis represents the number of weeks after the launch of Monero.

Clearly, with the advent of RingCTs, the number of outputs per transaction was consistently around 2. The number of inputs however does not show a stable and consistent pattern. The peaks in the curve do however tend to lower. It could be due to the existence of denominations from non RingCTs. As a result, users were still forced to merge a large number of inputs to reach a desired transaction amount.

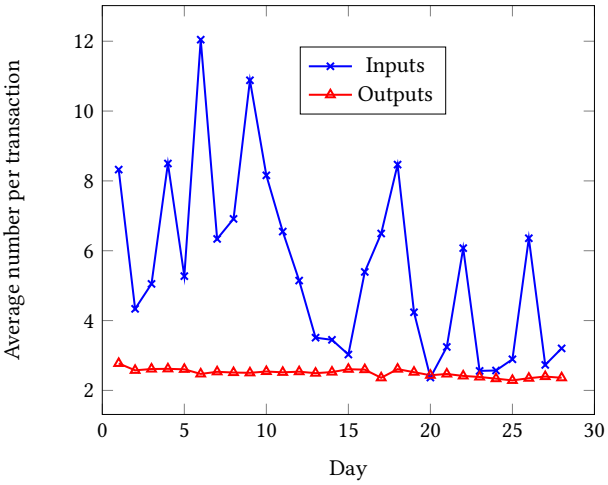

Figure 5: The plot presents the average number of inputs and outputs in a RingCT. The x-axis represents the number of days after the launch of RingCTs.

3.5 Summary of Our Findings

We summarize the main results of this section in Table 3.

Table 3: Monero network and usage statistics summary.

Observable	Key findings
Liquidity	1. 93% of output values appear only once. 2. 99.98% of output values are not denomination compliant.
Nb. mix-ins	1. 96% of inputs mix-ins in [0, 4]. 2. 65.9% use zero mix-ins. 3. In at least 85% of cases, a higher number of mix-ins could have been chosen.
Nb. inputs/outputs	1. Non RingCTs: Average number of inputs and outputs per transaction is 19 and 17 resp. 2. RingCTs: Average number of inputs and outputs per transaction is 3.7 and 1.2 resp.

4 TRACEABILITY ANALYSIS: METHODOLOGY AND DEFINITIONS

Our goal in the rest of the paper is to exploit the findings of the previous section to mount traceability attacks on Monero. In this section, we set the ground for our attacks by presenting the general methodology and the underlying reason for their potential success.

We first recall that if an input key in a transaction is mixed with m mix-ins, then the input key should be untraceable among the $m+1$ keys being used to create the ring signature. In other words, the *anonymity-set size* of the real input key is expected to be $m+1$.

A critical hypothesis in the anonymity-set size argument is that all the $m+1$ input keys that form an input are equally likely to be the real input key being spent[3]. If this condition does not hold, the anonymity-set size gets reduced and it becomes possible to distinguish a mix-in key from the real input key. This argument is reminiscent of a similar criticism previously made by Serjantov and Danezis in [17] in the context of mix-nets.

To see this, let us consider the following scenario where a user Alice pays to another user Bob using one mix-in. The corresponding transaction has one input composed of two input keys: P_1 (belonging to Alice) and P_2 belonging to some unknown user (Cf. Table 4a). The corresponding output key will be P_{Bob}. Now, consider a situation where another user with key P_3 decides to spend his output using two mix-ins by making a payment to Charlie (Cf. Table 4b). The user choses P_1 and P_2 as the mix-ins.

Table 4: View of a global passive adversary. Each key in Tx-2 does not have the same probability of being spent. P_3 has a higher probability than P_1 and P_2 as they also appear in a previous transaction Tx-1.

(a) Tx-1.

Input keys	Output keys
P_1 P_2	P_{Bob}

(b) Tx-2.

Input keys	Output keys
P_1 P_2 P_3	P_{Charlie}

[3]We abusively use the term 'a key being spent' rather than the more conventional 'a TXO being spent'. This is only to ease the presentation as each TXO is identified by a one-time public key.

Any global passive adversary looking at the blockchain will observe the two transactions as given in Table 4 and conclude the following:

(1) With only Tx-1, the probability that funds in P_1 was spent is 1/2, the same for P_2.
(2) With only Tx-2, the probability that funds in P_1 was spent is 1/3, the same for P_2 and P_3. This is the expected behavior in Monero.
(3) Looking at both Tx-1 and Tx-2, the adversary may conclude that the probability that P_1 was spent in the second transaction is 1/4, the same for P_2. But, the probability that P_3 was spent is 1/2. Hence, each input key may not have the same probability of being spent in a transaction given that one of them has a priori (non-zero) probability of being already spent.

While the above example does show that all input keys in a transaction input may not necessarily have the same probability of being spent. It however does not show how to trace (identify) the real input key. One of our heuristics (developed in the next section) extends this idea by analyzing whether an input key has been previously spent in a transaction with a probability 1. If such a key is identified and used later as a mix-in in another transaction, then the anonymity-set size of the input can be reduced by 1.

In the rest of this paper, we use the following definitions to define what it means to be traceable at different granularity.

Definition 4.1. [**Effective anonymity-set size**] Consider a transaction Tx and one of its inputs I that uses m mix-ins keys to create the ring signature for I. If it is possible to identify any k mix-in keys out of m, then the *effective anonymity-set size* of I is $m+1-k$.

For instance, consider a scenario where an input is spent using say two mix-ins, and the adversary is able to identify only one mix-in (because it was previously spent in an another transaction). In this scenario, the effective anonymity-set size is two instead of the expected size of three. In the best case (from an adversarial perspective), it may be possible to identify all mix-ins used in an input. We refer to such inputs as *traceable inputs*.

Definition 4.2. [**Traceable input**] Given a transaction Tx and one of its inputs I, I is *traceable* if its effective anonymity-set size is one.

Extending the definition of traceability to transactions, we have the following definition.

Definition 4.3. [**Traceable transaction**] A transaction Tx is said to be traceable if each of its inputs is traceable.

The untraceability guarantee in Monero relies on the assumption that it is hard for an adversary to distinguish a spent TXO from an unspent TXO. In the next section, we develop three heuristics that allow us to test this hypothesis and reduce the anonymity-set size for an input.

5 TRACEABILITY ATTACKS

We present three heuristics that define our attack strategies on existing Monero transactions. For each of the heuristic, we present the attack routine and then evaluate its impact on the Monero blockchain.

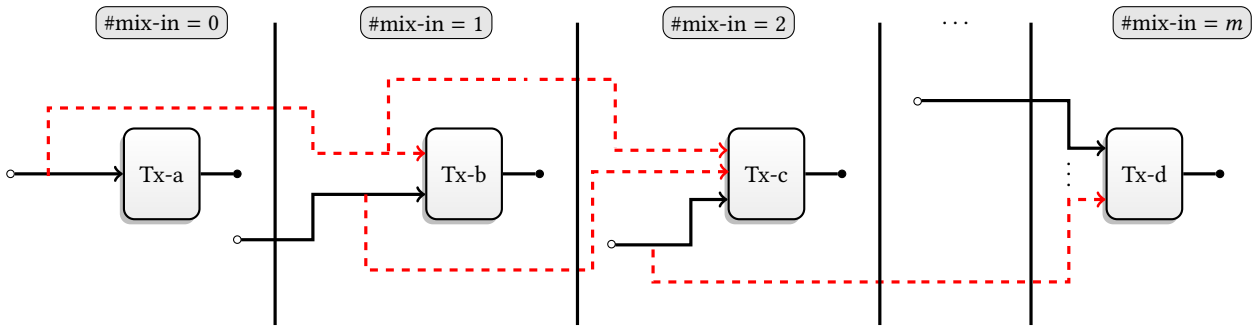

Figure 6: Heuristic I: Cascade effect due to zero mixins. Each transaction has only one input (left of the transaction) and one output (on the right). The number of mix-ins used increases from left to right. Dashed lines represent the input keys identified as a mix-in. Lines in bold are the real input keys being spent.

5.1 Heuristic I: Leveraging Zero Mix-ins

We first make the following observation. Consider the case, where, a user Alice creates a transaction Tx-a with an input that she spends without using any mix-in. Alice's input key can clearly be identified as spent. Now, consider a later transaction Tx-b created by another user Bob who uses Alice's input key as a mix-in for his input. If the number of mix-ins used by Bob is one. Then, any adversary looking at the blockchain can identify the real input key being spent in Bob's transaction and his input becomes traceable too. More generally, if the number of mix-ins used by Bob is $m > 1$, then the effective anonymity-set size now becomes m (instead of the expected value of $m + 1$).

A closer look reveals that use of zero mix-in leads to a *cascade effect* where the traceability of an input affects the traceability of another input in a later transaction. Figure 6 schematically presents this cascade effect triggered by Tx-a that uses no mix-in. Input in Tx-a is clearly traceable. Tx-b uses one mix-in in the form of the previously traced input and hence is also traceable. Tx-c now uses two mix-ins as the real inputs of Tx-a and Tx-b respectively. Since, these inputs have been previously identified as spent, Tx-c is also traceable.

One must note that the cascade effect may not always make an input traceable. In fact, it is also possible that it may only reduce the effective anonymity-set size but still less than the expected value. For instance, Tx-d uses m mix-ins but only one of those have been previously identified as spent. Hence, Tx-d remains untraceable but its effective anonymity-set size is m (instead of the expected value of $m + 1$).

5.1.1 Attack routine. Implementing Heuristic I as a full fledged attack routine is straightforward and a pseudocode is presented in Algorithm 1. For the sake of flexibility and future deployability in robust applications, the algorithm takes two parameters T and η. T is the highest block height to analyze and η is the maximum number of iterations.

The algorithm runs in three steps. In the first step (Lines 1-2), it initializes two data structures: `spentKeys` and `keysToAnalyze`. The former stores a set of input keys that have been identified as already spent. This set would grow with the number of iterations and will reach a stable point when no new key can be marked as

Algorithm 1: Heuristic I

Data: η: the number of iterations and T: the maximum block height to analyze.
Result: A set spentKeys of spent output keys.

```
1  spentKeys ← ∅
   // Each entry of keysToAnalyze is a list of keys.
2  keysToAnalyze ← [ ]
3  foreach height ≤ T do
       // Retrieve block with given height
4      block ← getBlock(height)
       // Retrieve non-coinbase transactions
5      transactions ← getTransactions(block)
6      foreach tx ∈ transactions do
           // Retrieve inputs in the transaction
7          inputs ← getInputs(tx)
8          foreach input ∈ inputs do
               // Retrieve input keys in this input
9              inKeys ← getInputKeys(input)
               // Add all keys as a list
10             keysToAnalyze.add(inKeys)
11 repeat η times or until spentKeys reaches stable point
12     foreach inKeys ∈ keysToAnalyze do
           // Store keys that are note spent.
13         untracedKeys ← ∅
14         foreach inKey ∈ inKeys do
15             if inKey ∉ spentKeys then
16                 untracedKeys ← untracedKeys ∪ inKey
17         if |untracedKeys| = 1 then
18             spentKeys ← spentKeys ∪ untracedKeys
                   /* Remove newly identified spent key
                   from each entry of keysToAnalyze     */
19             keysToAnalyze.removeAll(untracedKeys)
20 return spentKeys
```

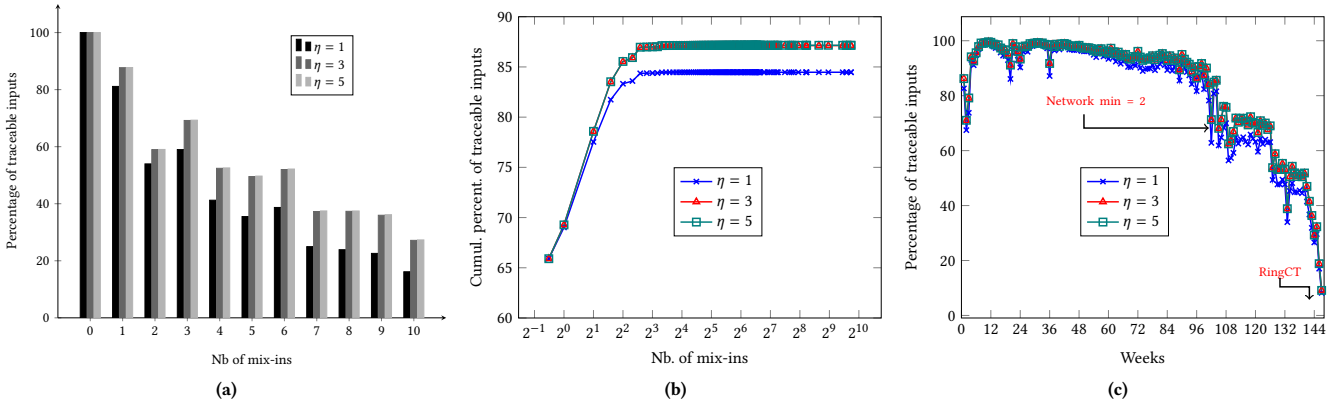

Figure 7: Results on traceability using Heuristic I. In (a), we present the percentage of traceable inputs as a function of the number of mix-ins. In (b), we plot the cumulative percentage of traceable inputs as a function of the number of mix-ins. In (c), we present the evolution of the percentage of traceable inputs over time. In the x-axis, we have the number of weeks after the launch of Monero.

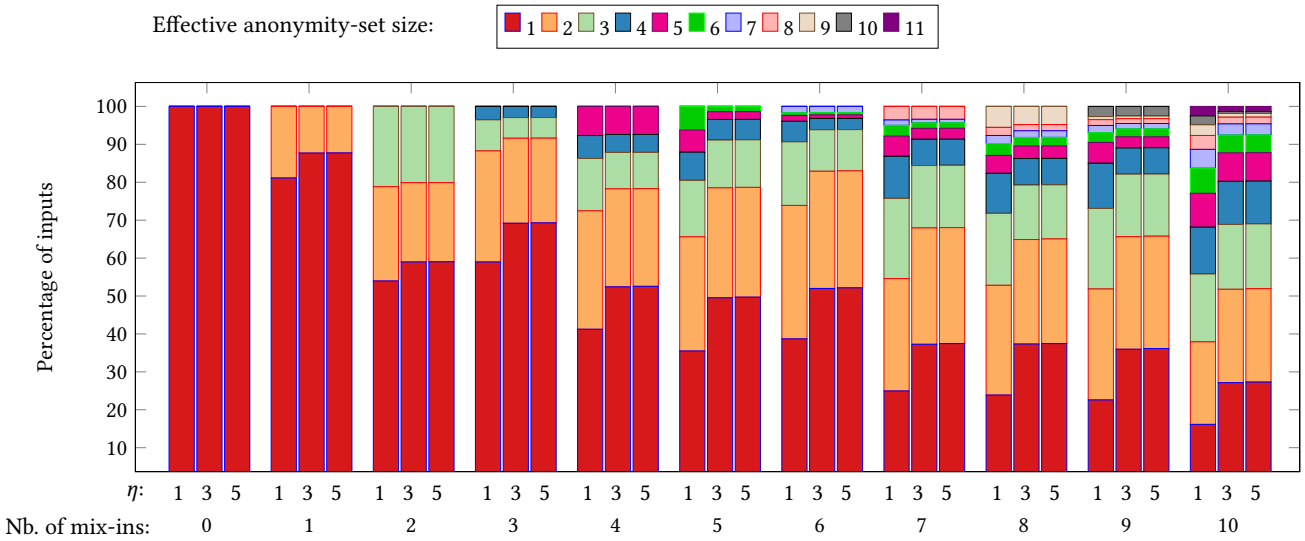

Figure 8: Results on effective anonymity-set size using Heuristic I. In x-axis, we have the number of mix-ins from 0 to 10. For each of these mix-ins we plot three stacked bars corresponding to $\eta = 1, 3, 5$. Each stacked bar for a fixed η represents the percentage of inputs that have an effective anonymity-set size between 1 to 11. For instance, for inputs using only 2 mix-ins, and for $\eta = 5$, the percentage of inputs with efffective anonymity-set size of 1 is 59%, the percentage of inputs with effective anonymity-set size of 2 is 21% and the percentage of inputs with effective anonymity-set size of 3 is 20%.

spent. The second data structure `keysToAnalyze` stores yet-to-be-analyzed keys. It can very simply be seen as a list of items, where each item is itself a list of input keys. The data structure provides two interfaces: `add()` and `removeAll()`. The former adds a list of input keys corresponding to an input to the data structure, while the latter removes an input key from every entry of the data structure. The number of keys in this data structure will decrease with the number of iterations until no new key can be marked as spent.

In step 2 (Lines 3-10), the algorithm uses Monero's JSON interface to retrieve all blocks with height less than or equal to T, the coinbase transactions therein, the inputs and all the corresponding input keys. These input keys are then used to populate `keysToAnalyze`.

In step 3, the heuristic is applied on every set of input keys (corresponding to an input) in `keysToAnalyze`. The algorithm is bootstrapped by all inputs which use no mix-in. An input key k_{in} from an input is marked as spent if all other input keys for that input have been previously marked as spent. In fact, the key k_{in} is the real input being spent here. Once, such a key is identified (Lines 14-18), it is added to the list of keys identified as already spent. Finally, the newly identified spent key is removed from `keysToAnalyze` to

avoid analyzing the same key multiple times. This step is repeated until no more spent keys can be identified or when the number of iterations reaches its maximum value η (whichever is earlier). The parameter η is used in the algorithm only for the sake of flexibility where an early stop is desired.

It is also possible to run step 3 in a more optimal manner. The optimization exploits the cascade effect of Figure 6. In fact, before analyzing the inputs of a block, they can first be sorted by their number of mix-ins, which is a greedy attempt to identifying traceable inputs as soon as possible. It is not developed in Algorithm 1 to ease presentation.

5.1.2 Impact. The impact of Heuristic I on the traceability of inputs is collectively shown in Figure 7. We first note that the success of our attack based on Heuristic I is mainly due to the fact that over 65% of inputs do not use any mix-ins. It impacts the traceability of another 22% of the inputs, leading to a total of 87% of traceable inputs.

In Figure 7a, we present the percentage of traceable inputs for number of mix-ins less than or equal to eleven. Note that these mix-in sizes together cover 99.8% of all inputs in our dataset. We present histograms for three values of η (1,3 and 5). With $\eta = 5$, we observe that the set of spent keys almost reaches a fixed point. Just after the first iteration ($\eta = 1$), the number of traceable inputs using one mix-in reaches as high as 81%. For $\eta = 5$, this percentage of traceable inputs using one mix-in becomes 87%. The plot also shows the cascade effect as inputs using a high number of mix-ins such as 10 also have a considerable percentage of traceable inputs (27% for $\eta = 5$).

Figure 7b presents the cumulative percentage of traceable inputs as a function of the number of mix-ins. Clearly, as the number of mix-ins increases, the cascade effect deteriorates and roughly 87% of all inputs become traceable. It is interesting to note that the cascade effect leads to one traceable input that uses 153 mix-ins. This is the largest number of mix-ins that gets affected by the cascade effect.

Figure 7c presents the evolution of the percentage of traceable inputs over time grouped by week. Since, the initial transactions did not use any mix-in, a large majority (over 95%) of the inputs were traceable. The maximum percentage of traceable inputs per week is 98.9% (in the 10^{th} week). In fact, the percentage dropped to roughly 62% in the 105^{th} week when the network-wide minimum mix-in of 2 could come into effect. Since then, the percentage of traceable inputs has seen a consistent decline. For the last week, only 8% of all inputs were found to be traceable. Note that Heuristic I could not find at any result on RingCT inputs.

We found several instances where Heuristic I could not identify a traceable input, but it did nevertheless succeeded in reducing the effective anonymity-set size. Figure 8 presents these findings. For the sake of completeness, it also includes the results presented in Figure 7a. We observe that for $\eta = 5$, roughly 24% of inputs that use 10 mix-ins have an effective anonymity-set size of two. This shows how close Heuristic I can be in identifying the real input. Moreover, the plot shows that as the number of mix-in increases, the percentage of inputs on which Heuristic I does not work at all tends to decrease. In fact, for inputs using 10 mix-ins, Heuristic I does not affect the effective anonymity-set size for only 0.9% of all such inputs.

5.2 Heuristic II: Leveraging Output Merging

We develop Heuristic II to mainly trace RingCTs which Heuristic I failed to do. However, it naturally extends to non RingCTs just as well. In order to present the underlying idea, let us consider a scenario where a user creates a transaction Tx-a having one input and two outputs O_1, O_2. (Cf. Figure 9). Without loss of generality, let us assume that only 1 mix-in is used. At a later time, another user creates a transaction Tx-b with two inputs I_1, I_2 and one output. Let us suppose that both the inputs use one mix-in each. The first input I_1 uses one of the outputs O_1 of Tx-a as an input key. Similarly, the second input I_2 uses the other output O_2 of Tx-a as an input key. Heuristic II then identifies O_1 and O_2 as the real input keys being spent in Tx-b.

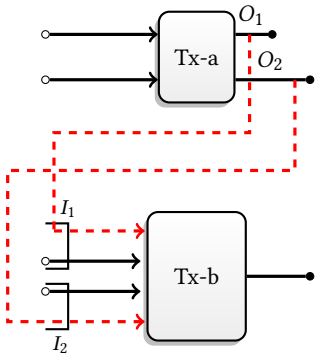

Figure 9: Heuristic II. Tx-a is a transaction with one input that uses one mix-in. It has two outputs O_1 and O_2. Tx-b is another transaction that has two inputs denoted by I_1 and I_2. Each input again has one mix-in. Both I_1 and I_2 include outputs of Tx-a. According to Heuristic II, the input keys O_1 and O_2 represented using the dashed line are the real keys being spent in Tx-b.

Heuristic II functions under the assumption that while creating a transaction, it is less likely to choose several mix-ins that are outputs of a single previous transaction. Hence, if a transaction includes keys (possibly across several inputs) that are outputs of a single previous transaction, then they are likely to be the real ones being spent. Due to the underlying assumption, the results obtained from Heuristic II cannot always be conclusive and in fact may admit false positives. In this sense, Heuristic II is weaker than Heuristic I as the latter does not admit any false positive and hence yields the ground truth.

5.2.1 Attack routine. In order to simplify the further discussion of Heuristic II, we refer to transactions of type Tx-a (as in Figure 9) as a *source* transaction, while, those of type Tx-b as a *destination* transaction. Hence, a destination transaction uses two or more outputs of a source transaction across its inputs. The minimum number of two or more outputs is needed so as to capture the merging of outputs.

In Algorithm 2, we present the general attack routine. The algorithm takes a parameter T for the maximum block height to analyze and returns a set of candidate output keys. The set of candidate

keys aggregates all output keys of all source transactions for which we find at least one destination.

The core of the algorithm is the function `analyzeOutput(·, ·)` defined in Lines 9-24. It takes two parameters: $height_1$ and $outputKeys$. The second parameter is a set of output keys for a source transaction that appears in block height $height_1$. The function determines whether there exists a destination that uses at least two of the keys in $outputKeys$ across its inputs (Lines 18-23). If a destination exists, the corresponding keys are added in the output set of candidate keys (Lines 22-23).

Algorithm 2: Heuristic II

```
   Data: T: the maximum block height to analyze.
   Result: A list candidateSets of linked inputs/outputs.
1  candidateSets ← ∅
2  foreach height ≤ T do
3      block ← getBlock(height)
       // Retrieve non-coinbase transactions
4      transactions ← getTransactions(block)
5      foreach tx ∈ transactions do
           // Retrieve outputs in the transaction
6          outputKeys ← getOutputKeys(tx)
           // Invoke analyzeOutput(·, ·) defined below
7          candidateSets ← candidateSets ∪
             analyzeOutput(height, outputKeys)
8  return candidateSets
9  define analyzeOutput(height_1, outputKeys) as
10     candidateSets ← ∅
11     foreach height_2 ∈ [height_1 + 1, T] do
12         block ← getBlock(height_2)
           // Retrieve non-coinbase transactions
13         transactions ← getTransactions(block)
14         foreach tx ∈ transactions do
15             nonEmptyIntersections ← 0
16             candidateKeys ← ∅
17             foreach input of tx do
                   // Retrieve input keys in this input
18                 inKeys ← getInputKeys(input)
19                 if inKeys ∩ outputKeys ≠ ∅ then
                       /* increment increases
                          nonEmptyIntersections by
                          inKeys ∩ outputKeys          */
20                     increment(nonEmptyIntersections)
21                     candidateKeys ←
                         candidateKeys ∪ (inKeys ∩ outputKeys)
22             if 2 ≤ nonEmptyIntersections and
                 2 ≤ |candidateKeys| then
23                 candidateSets.add(candidateKeys)
24     return candidateSets
```

Due to the probabilistic nature of how mix-ins are chosen, Algorithm 2 may encounter the following scenarios:

- **S1:** It may not find any destination for a given source.
- **S2:** It may find several destinations for a given source.
- **S3:** It may find one (or more) destination for a given source, where the same source output appears in more than one destination input.
- **S4:** It may find one (or more) destination for a given source, where more than one source outputs appear in a single destination input.

S1 essentially means that Heuristic II failed to yield any result on the given source instance. While, S2 means that the heuristic has false positives at the transaction level, hence it is hard to ascertain the real destination for a given source. S3 presents the worst case for the heuristic. It means that the heuristic has false positives even at the input level. Hence, it is hard to even ascertain the input where the source output was indeed spent. S4 essentially yields a set of candidate keys being spent in the input.

In the following section, we conduct experiments to estimate how well Heuristic II performs. To this end, we take a two dimensional approach. We first measure how frequently the above scenarios occur and then estimate the false positive rate by relying on the ground truth data obtained from Heuristic I.

Remark 1. *Heuristic II can also be used to break the unlinkability guarantee of Monero. To see this, one may observe through the example of Figure 9 that since O_1 and O_2 are the real input keys being spent in I_1 and I_2, they must belong to the same Monero user. This means that with the help of Heuristic 2, it becomes possible to link two outputs to the same user. As this unlinkability attack is based on Heuristic II, it also entails false positives. Since, the focus of this work is on traceability, we do not develop this any further and leave it as a future work.*

5.2.2 Impact. Heuristics II found results on 410,237 different source transactions, which is roughly 43% of all transactions in our dataset. These source transactions also include 636 RingCTs, which is 1% of all RingCTs in our dataset. The low fraction of RingCTs is essentially due to the fact that the average number of inputs and outputs per RingCT is only 3.7 and 1.2 respectively (Cf. Figure 5). Recall that Heuristic II essentially exploits the use of outputs of source transactions in a destination transaction. Hence, a low number of inputs and outputs directly affects the applicability of the heuristic.

In Figure 10a, we present the results obtained on 409,601 non RingCT sources. Around 60% of all source transactions have only 1 matching destination. The maximum number of destinations found for a source was 146. However, the percentage of source drops exponentially as the number of destinations increases. Similarly, Figure 10b presents the results obtained on 636 RingCT sources. We observe that a source has at most 3 destinations and a majority of the source transactions (95.1%) have only one destination. The small number of destinations in case of RingCT sources is essentially due to the fact that RingCTs inherently have a low number of inputs.

As we discussed earlier, Heuristic II can generate false positives. In order to estimate the accuracy of the results of Heuristic II, we compare its results with those obtained by Heuristic I that provides the ground truth. Unfortunately, Heuristic I does not provide any ground truth on RingCTs, hence, we cannot estimate the accuracy

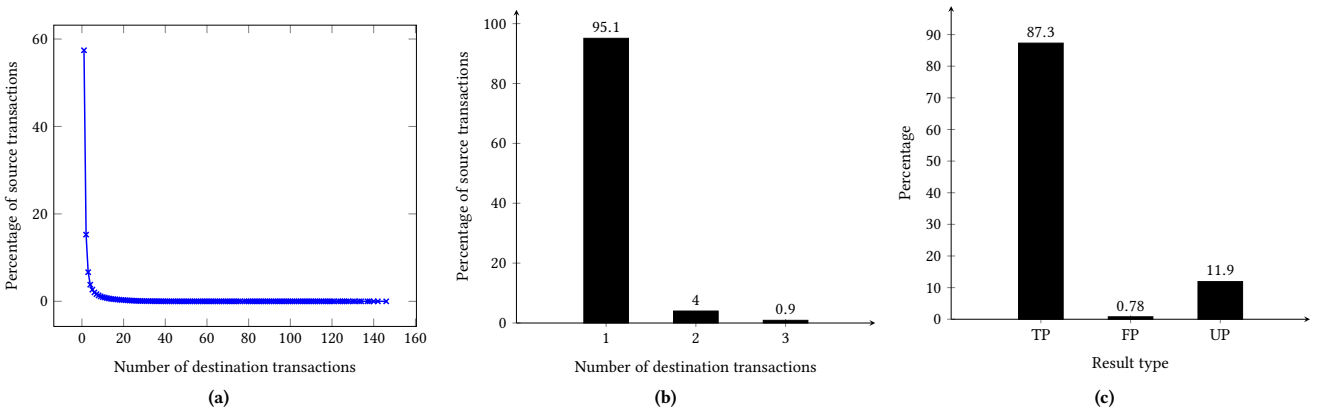

Figure 10: Plot (a): Result of employing Heuristic II on non RingCTs. The x-axis presents all the observed number of destination transactions for a given source transaction. In y-axis, we show the number of source transactions that admit a given number of destination transactions. The number is given as a fraction of 409,601 non RingCT source transactions. Plot (b): Result of employing Heuristic II on RingCTs. The x-axis presents all the observed number of destination transactions for a given source transaction. In y-axis, we show the number of source transactions that admit a given number of destination transactions. The number is given as a fraction of 636 RingCT source transactions. Plot(c): Overall observed percentage of TP, FP and UP.

of Heuristic II on RingCTs. However, if it performs well on non RingCTs, then we may extrapolate this result over RingCTs and expect similar accuracy.

In order to compare the two heuristics, we use the following terms:

- **True positive (TP):** An input creates a *true positive* if: a) Heuristic II identifies a unique key as the one being spent in the input and, b) the key is the same as the one identified by Heuristic I. In other words, the two heuristics agree on the real key being spent in the input.
- **False positive (FP):** An input creates a *false positive* if all the keys identified as being spent by Heuristic II were actually found to be spent in a different input by Heuristic I. In other words, none of the probable keys identified by Heuristic II was actually the real key being spent in the input. Hence, the two heuristics disagree on the real key being spent.
- **Unknown positive (UP):** An input creates an *unknown positive* if at least one of the keys identified by Heuristic II could not be identified as being spent in any input (of any transaction) by Heuristic I. The ambiguity is due to the fact that Heuristic I does not give ground truth for all inputs.

Now that the terms TP, FP and UP are established, we are ready to present the results on the accuracy of Heuristic II. The overall accuracy of Heuristic II computed over all non RingCT inputs for which it returns a result is given in Figure 10c. The result shows that Heuristic II has an overall true positive rate of 87%, while the false positive rate is as low as 0.78%, while is inconclusive for around 12% of inputs. The high true positive rate on non-ring CTs clearly demonstrates that it should do equally well even on RingCTs. However, due to the lack of ground truth it is impossible to verify this.

A breakdown of TP, FP and UP as a function of number of mix-ins is given in Figure 11. The plot shows that as the number of mix-in increases, the percentage of TP decreases, while the percentage of UP increases. Moreover, irrespective of the number of mix-ins used, the number of FP remains very close to 0. All of this is essentially due to the fact that the output of Heuristic I deteriorates as the number of mix-ins increases and hence it becomes hard to verify the result of Heuristic II due to a lack of ground truth.

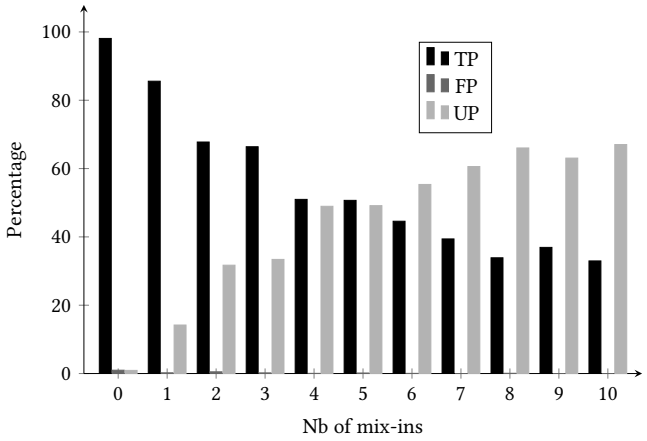

Figure 11: Observed percentage of TP, FP and UP as a function of number of mix-ins. The result is for non RingCTs.

5.3 Heuristic III: Temporal Analysis

Our third heuristic leverages the fact that a TXO does not remain unspent for an infinite time. In general, its probability of being

spent should increase with time. Indeed, a TXO that has been on the blockchain for 100,000 blocks is much more likely to have already been spent than an output that has been on the blockchain for only 100 blocks. In light of this, we define Heuristic III in the following manner: *Given a set of input keys used to create a ring signature, the real key being spent is the one with the highest block height*, where it appeared as a TXO.

5.3.1 Results. We tested Heuristic III and compared its results with the ground truth data obtained from Heuristic I ($\eta = 5$). Globally, we observed that Heuristic III has a true positive rate of 98.1%. This clearly shows that Heuristic III is very accurate and very often the most recent TXO in an input is the real one being spent.

In theory, Heuristic III can be prevented by mixing with only those TXOs that are yet to be redeemed. However, this is intractable due to the underlying ring signature. In order to circumvent this problem, Monero developers have decided since April 5^{th} 2015 to sample mix-ins from a triangular distribution.[4] A triangular distribution essentially gives higher probability to newer TXOs than to ones that are old and hence can potentially mitigate the attack. Note that prior to April 5^{th} 2015, mix-ins were sampled from a uniform distribution, *i.e.*, each TXO had the same probability of being a mix-in for any input at any given time. We evaluate how well triangular distribution can be useful in mitigating our attack based on Heuristic III. The results are shown in Table 5. While, using triangular distribution does help in reducing the number of true positives, the gain over uniform distribution is however marginal, *i.e.*, only 3.5%.

Table 5: Breakdown of traceable inputs obtained using Heuristic I. In the first row, we show the total number of traceable inputs that employ uniform distribution and triangular distribution. The second and third row show the true and false positive rate observed using Heuristic III.

	Uniform dist. (until April 4, 2015)	**Triangular dist.** (since April 5, 2015)
#Traceable inputs	9885810	6174801
True positive	99.5%	96%
False positive	0.5%	4%

Our results of Table 5 clearly show that sampling mix-ins from a triangular distribution does not mitigate well the attack based on Heuristic III. In fact, the choice of mix-ins must take into account the real spending behavior of users. To this end, we use the results of Heuristic I to extract information on when a TXO (in terms of block height) is created and when it gets spent. We then compute the difference of the two block heights. The resulting data is shown in Figure 12, where we plot the frequency of TXOs that share the same difference. We observe that users' spending habits can be grouped into four distinct categories characterized by the difference in the block heights:

(1) $[0, 10]$ blocks: Only 0.17% of TXOs fall into this category.
(2) $]10, 100]$ blocks: 9.16% of TXO fall into this category
(3) $]100, 1000]$ blocks: 28.4%
(4) $]1000, \infty]$ blocks: 62.27%

We propose that instead of choosing mix-ins from a triangular distribution, mix-ins should in fact be chosen while considering the probability density function of Figure 12. For instance, the probability of choosing a TXO that is less than 10 blocks old should be smaller than the probability of choosing a TXO that is at least 100 blocks old. In fact, if mix-ins are not chosen according to the spending behavior of Monero users, then it becomes possible to break traceability.

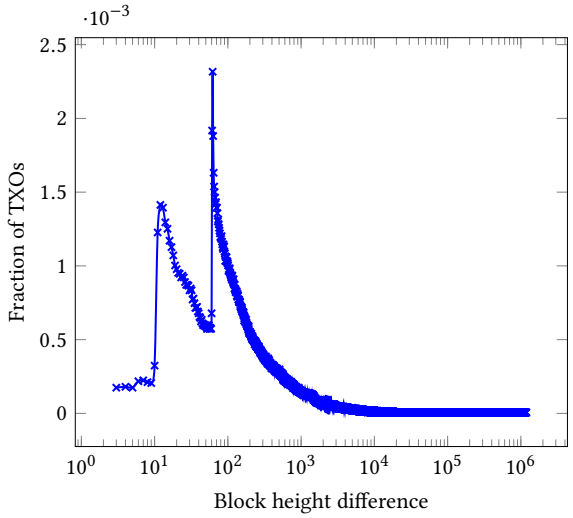

Figure 12: Spending habit of Monero users. In x-axis, we have the difference between the block height where the TXO was created and the block height where the TXO was spent. It measures the duration for which the TXO remained unspent. In y-axis, we plot the fraction of TXOs that share the same difference. Only 1 out of every 100 data points are shown.

6 RELATED WORK

Our work is motivated by two prior unpublished works on the privacy analysis of Monero: MRL-001 [18] and MRL-004 [9]. Both of these have been authored by Monero researchers and developers in the form of Monero Research Lab (MRL) report. Below, we categorically position our paper *wrt* these two prior works.

1. MRL-001 [18]: This work studies our attack using Heuristic I and claims that it may lead to a *chain reaction* — equivalent to our term *cascade effect*. While the authors argue that it is plausible to mount the attack on the blockchain, they however do not provide any data analysis to demonstrate its impact in practice. Our work complements [18] by presenting a comprehensive empirical analysis that shows 1) The risks of using no mix-in in practice, 2) How often the risks may arise? 3) How far the chain reaction can propagate? 4) How the impact has evolved over time?

Moreover, the authors conclude that:

[4] https://github.com/monero-project/monero/commit/f2e8348be0c91c903e68ef582cee687c52411722, Accessed on April 14^{th} 2017.

> *Any CryptoNote coin that allows for only 1 mix-in is vulnerable to a slow chain reaction in which the owner of very few private keys can violate the untraceability of much larger number of other users.*

The report further recommends enforcing a minimum mix-in of two per input (whenever feasible). The recommendation was later incorporated in the Monero protocol. Our work however shows that even almost a year after implementing the recommendation, the traceability risk still persists.

2. MRL-004 [9]: This is a more recent report that mainly studies the need of denominations, Heuristic II and Heuristic III.

The work identifies the problem with TXOs corresponding to dust values for which it may be impossible to achieve a desired anonymity-set size. In order to facilitate mixing and to guarantee that sufficient liquidity is maintained, the authors propose the idea of denominations. However, as we show, over 99.8% of output values are not denomination compliant. Moreover, 93% of output values appear only once. While the recommendation has indeed been incorporated in Monero, our work clearly shows that denominations cannot be fully enforced in practice due to the underlying system constraints.

Heuristic II has also been studied as a hypothetical situation and a mitigation strategy has been suggested. More concretely, the report suggests that the protocol must prevent output merging by incorporating the idea of "pay-in-stream". The idea is to create as less outputs as possible per transaction by splitting the amount and making the payment over several transactions. Our results on Heuristic II suggest that the output merging is still prevalent among non RingCTs, a bit less so for RingCTs.

As for Heuristic III, the report mentions that temporal analysis is a potential threat and argue that in order to prevent the attack:

> *... the developers of Monero must estimate the probability distribution governing the age of transaction outputs.*

The report however does not provide any result how on to estimate the age of TXOs. It is argued that this would require performing a blockchain analysis. In light of this, our work studies the spending habit users and empirically provides the desired probability distribution function. We hope that our results can be used to improve the mix-in strategy.

Note that since Monero developers chose not to perform the required blockchain analysis, they decided on employing a triangular distribution to sample the mix-ins from. Our work shows that a triangular distribution does not mitigate well attacks based on temporal analysis.

7 CONCLUSION

To summarize, this work presents several attacks on the traceability guarantees of Monero. Our attacks are easy to mount and only requires a passive blockchain analysis. Moreover, our attacks are effective as over 87% of inputs are rendered traceable. We also found some traceability results on RingCTs and finally propose a better method (than the one currently employed) to choose mix-ins that mitigates temporal analysis.

Our results hereby reaffirm the weaknesses of anonymity-set size as a privacy metric. As a future work, we aim to study traceability under active attacks on Monero, where the adversary can take part in the protocol as a malicious user.

REFERENCES

[1] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. 459–474.

[2] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. 2012. High-speed high-security signatures. *J. Cryptographic Engineering* 2, 2 (2012), 77–89.

[3] David Chaum. 1982. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*. Plenum Press, New York, USA, 199–203.

[4] Dash 2017. Dcash. (2017). https://www.dash.org/ Accessed on 2017-04-07.

[5] Elliptic 2016. (2016). https://www.elliptic.co/.

[6] Michael Fleder, Michael S. Kester, and Sudeep Pillai. 2015. Bitcoin Transaction Graph Analysis. *CoRR* abs/1502.01657, Article arXiv:1502.01657 (2015), 8 pages.

[7] Eiichiro Fujisaki and Koutarou Suzuki. 2007. Traceable Ring Signature. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*. 181–200. DOI: https://doi.org/10.1007/978-3-540-71677-8_13

[8] Tom Elvis Jedusor. 2016. Mimblewimble. (2016). https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt Accessed on 2017-04-07.

[9] Adam Mackenzie, Surae Noether, and Monero Core Team Burkhard. 2015. *Improving Obfuscation in the CryptoNote Protocol*. Research Bulletin MRL-0004. Monero Research Lab.

[10] Greg Maxwell. 2015. Confidential Transactions. (2015). https://people.xiph.org/~greg/confidential_values.txt Accessed on 2017-04-07.

[11] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2013. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, October 23-25, 2013*. ACM, Barcelona, Spain, 127–140.

[12] Monero 2016. Monero Developer Guide: Daemon RPC Documentation. (2016). https://getmonero.org/knowledge-base/developer-guides/daemon-rpc, Accessed on 2017-02-07.

[13] Shen Noether, Adam Mackenzie, and the Monero Research Lab. 2016. Ring Confidential Transactions. *Ledger* 1, 0 (2016), 1–18.

[14] Fergal Reid and Martin Harrigan. 2011. An Analysis of Anonymity in the Bitcoin System. In *PASSAT/SocialCom 2011, Privacy, Security, Risk and Trust (PASSAT), 2011 IEEE Third International Conference on and 2011 IEEE Third International Conference on Social Computing (SocialCom), 9-11 Oct., 2011*. IEEE, Boston, MA, USA, 1318–1326.

[15] Ronald L. Rivest, Adi Shamir, and Yael Tauman. 2001. How to Leak a Secret. In *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*. 552–565.

[16] Dorit Ron and Adi Shamir. 2013. Quantitative Analysis of the Full Bitcoin Transaction Graph. In *Financial Cryptography and Data Security - 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 6–24.

[17] Andrei Serjantov and George Danezis. 2002. Towards an Information Theoretic Metric for Anonymity. In *Privacy Enhancing Technologies, Second International Workshop, PET 2002 (Lecture Notes in Computer Science)*, Vol. 2482. Springer, San Francisco, CA, USA, 41–53.

[18] Sarang Noether Surae Noether and Adam Mackenzie. 2014. *A Note on Chain Reactions in Traceability in CryptoNote 2.0*. Research Bulletin MRL-0001. Monero Research Lab.

[19] Nicolas van Saberhagen. 2013. *CryptoNote v2.0*. Technical Report. CryptoNote.

[20] Zerocoin Electric Coin Company. 2017. Zcash. (2017). https://z.cash/, Accessed on 2017-04-07.