Privacy-Enhancing Overlays in Bitcoin

Sarah Meiklejohn[1] and Claudio Orlandi[2]

[1] University College London
s.meiklejohn@ucl.ac.uk
[2] Aarhus University
orlandi@cs.au.dk

Abstract. In this paper, we explore the role of privacy-enhancing overlays in Bitcoin. To examine the effectiveness of different solutions, we first propose a formal definitional framework for virtual currencies and put forth a new notion of anonymity, *taint resistance*, that they can satisfy. We then approach the problem from a theoretical angle, by proposing various solutions to achieve provable taint resistance, and from a practical angle, by examining the taint resistance of the Coinjoin protocol.

1 Introduction

Virtual currencies have existed in various forms — e.g., customer loyalty programs — for decades, yet only in recent years have they exploded in popularity. The initial virtual currency driving this success, Bitcoin, was introduced in January 2009; today, it boasts an exchange rate of over 250 EUR per bitcoin and is in the process of integrating into the traditional banking system via payment gateways such as Bitpay — which as of March 2014 had signed up 26,000 merchants to accept the currency — and partnerships between Bitcoin exchanges such as Bitcoin.de and banks such as Fidor Bank AG. This tremendous growth has impelled regulators and law enforcement officials around the world to take a stand on virtual currencies, with the European Central Bank calling Bitcoin "the most successful — and probably most controversial — virtual currency scheme to date" [7] and the Federal Bureau of Investigation cautioning that, within the Bitcoin network, "law enforcement faces difficulties detecting suspicious activity, identifying users and obtaining transaction records" [8].

This tension between the growing popularity of virtual currencies and their perceived anonymity provides a unique problem for both users of these currencies and for regulators seeking to understand the true risks that they pose. The initial perception of Bitcoin was arguably that it provided anonymity, as evidenced by its adoption in the underground marketplace Silk Road (where bitcoins could be exchanged for goods such as drugs, firearms, and assassins) and by criminals running ransomware such as CryptoLocker or Ponzi schemes [18]. A recent line of research [15,16,2,12,19], however, showed that it was often possible to trace the movement of bitcoins throughout the network, so as a result the average Bitcoin user was not achieving much anonymity at all.

Perhaps in reaction to these results, a variety of new privacy-enhancing techniques have been proposed for virtual currencies. These techniques can be split into roughly two types: the first type introduces a new virtual currency — such as Zerocash [13,4] or DarkCoin — that seeks to improve on the anonymity properties of Bitcoin, and the second type proposes *overlays* that can be used without modifying the existing Bitcoin protocol. It is this latter approach that we focus on in this paper.

The main obstacle towards achieving anonymity in Bitcoin is its inherent transparency: while peers can identify themselves using a variety of pseudonyms, every transaction that has ever taken place — and thus the entire spending history of any given bitcoin — is globally visible. One method for improving anonymity in Bitcoin is to *mix* bitcoins together as follows: Alice holds 1 bitcoin and wishes to send it to Charlie, and Bob holds 1 bitcoin and wishes to send it to Dora. If Alice sends her bitcoin to Dora and Bob sends his to Charlie, then they have now essentially swapped the spending histories of these bitcoins; if Alice is a thief, Bob a legitimate user, and Charlie the exchange where Alice wants to cash out her bitcoin, then this swap has effectively "cleaned" the stolen bitcoin. To address the difficulty of finding users to mix with, *mix services* such as Bitcoin Fog and BitLaundry accept bitcoins and — in exchange for a fee — promise to send untainted bitcoins (i.e., bitcoins independent of the ones it received) to any address provided by the user.

Until a year ago, mix services were fairly unattractive, as a fair amount of trust was required to assume that a user would in fact receive his promised bitcoins. In August 2013, however, Gregory Maxwell proposed Coinjoin [10], which promised trustless mixing, and essentially provided a way for users like Alice and Bob in the above example to mix their bitcoins in a single transaction, without relying on either a central service or even any trust in each other. Since then, numerous other trustless mix services have been introduced, such as SharedCoin (`sharedcoin.com`) and CoinWitness [11], and SharedCoin has been integrated into `blockchain.info`'s popular wallet service (as of November 2013).

Our contributions. In this paper, we examine the landscape of privacy-enhancing overlays for Bitcoin in an attempt to determine the extent to which they succeed in achieving anonymity, and the extent to which their anonymity can be strengthened. By looking at the problem from these dual perspectives, we can obtain a picture of both what is feasible in theory and what happens in practice.

To understand the extent to which existing overlays achieve anonymity, it is first necessary to have a common framework in which these techniques can be analyzed. In Section 2, we propose such a framework and provide — to the best of our knowledge — the first formal definition of anonymity, *taint resistance*, that can be satisfied by Bitcoin and related virtual currencies. (The main previous definition of anonymity for virtual currencies, unlinkability, cannot be met here.) In Section 3, we then analyze how different mechanisms can provide taint resistance. While our framework provides a way to analyze the security of protocols, it is also important to understand the performance of these protocols in practice. In Section 4, we thus perform an experimental analysis of the Coinjoin protocol.

We do so from the perspective of both a passive adversary, who uses only the publicly available data from the transaction ledger, and — by engaging in our own coinjoin transactions — an active adversary, who uses its own participation to attempt to de-anonymize the activity of other participants. We find that naïve versions of these adversaries are not particularly successful in identifying which input addresses taint a given output address, and moreover that their success is heavily tied to the quirks of the Coinjoin protocol being used.

2 Definitions and Notation

We begin with formal specifications of decentralized electronic cash (or e-cash) and Coinjoin. We then introduce a notion of anonymity for virtual currencies that we call *taint resistance*.

2.1 Distributed electronic cash

To the best of our knowledge, the only existing formal definitions of anonymity in electronic cash are either in a centralized setting or the definition put forth for Zerocoin [13]. The former definitions are out of the scope of this work, as all our currencies of interest are fundamentally decentralized. The Zerocoin definitions do explicitly consider decentralization, but are not applicable in our setting because their definition of anonymity still closely matches the standard cryptographic notion of unlinkability (where, briefly, a user should not be able to distinguish between two coins), which is impossible to achieve for Bitcoin and the many altcoins that it has inspired.

With these considerations in mind, we set out to create a formal definition that encompasses existing virtual currencies. We define a *decentralized e-cash* protocol as a set of PT algorithms $(\mathsf{KeyGen}, \mathsf{Mint}, \mathsf{Spend}, \mathsf{Verify}, \mathsf{KeyCheck}, \mathsf{HistCheck}, \mathsf{ValCheck})$ that behave as follows: via $(pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$ one can generate a keypair; via $(\mathsf{hist}, \{\mathsf{coin}_i, sk_i\}_i) \xleftarrow{\$} \mathsf{Mint}(\mathsf{hist}, aux)$ one can generate coins and create a record of this generation; via $\mathsf{tx} \xleftarrow{\$} \mathsf{Spend}(\mathsf{hist}, \{\mathsf{coin}_i, sk_i\}_{i\in[m]}, \{pk_j, v_j\}_{j\in[n]})$ one can send v_j coins to recipients pk_j; via $\mathsf{accept/reject} \leftarrow \mathsf{Verify}(\mathsf{hist}, \mathsf{tx})$ one can (deterministically) decide if a transaction is valid or not; via $\mathsf{KeyCheck}(sk, \mathsf{coin})$ one can (deterministically) determine if a coin is compatible with a given secret key; via $\mathsf{HistCheck}(\mathsf{hist}, \mathsf{coin})$ one can (deterministically) determine if a coin is compatible with a given history; and via $\mathsf{ValCheck}(v, \{\mathsf{coin}_i\}_i)$ one can (deterministically) determine if a set of coins is compatible with a given value.

To define correctness of the protocol, we say that a set $\{(\mathsf{coin}_i, sk_i)\}_i$ is *valid* with respect to a given history hist and value v if (1) $\mathsf{KeyCheck}(sk_i, \mathsf{coin}_i) = \mathsf{accept}$ for all i, (2) $\mathsf{HistCheck}(\mathsf{hist}, \mathsf{coin}_i) = \mathsf{accept}$ for all i, and (3) $\mathsf{ValCheck}(v, \{\mathsf{coin}_i\}_i) = \mathsf{accept}$. We say that the protocol achieves correctness if $\mathsf{Verify}(\mathsf{hist}', \mathsf{Spend}(\mathsf{hist}, \{(\mathsf{coin}_i, sk_i)\}_i, \{(pk_j, v_j)\}_j)) = \mathsf{accept}$ for all sets $\{(\mathsf{coin}_i, sk_i)\}_i$ that are valid with respect to hist and $\sum_j v_j$, and for all hist' such that $\mathsf{hist} \subseteq \mathsf{hist}'$.

2.2 Coinjoin

The Coinjoin protocol is designed to make a simplistic taint tracking more difficult. If we consider how bitcoins get spent (or the inputs to the Spend algorithm specified above), we recall that the sender needs to create a transaction using the secret key corresponding to each of the input public keys. One might thus think that a party forming a transaction needs to know each of these keys, so that the input keys in a transaction are all owned by the same entity. Indeed, this observation has formed the basis of a number of heuristics used to cluster together different Bitcoin addresses [15,16,2,12,19]. In fact, the signatures required in a transaction can be formed in a distributed manner, so that — while highly indicative of single ownership until a year ago — multiple entities can come together to form an ordinary-looking transaction without having to share secret signing keys at all. Coinjoin exploits this ability to distribute the generation of a transaction, and in its simplest form operates as seen in Algorithm 2.1.

Algorithm 2.1: `coinjoin`: Output a transaction tx

Input: Sets $\left\{\{\mathsf{coins}_{k,i}\}_{i\in[m_k]}, \{pk_{k,j}, v_{k,j}\}_{j\in[n_k]}\right\}_{k\in[N]}$ belonging to each of N parties.

1. Each party k sends the set $\{pk_{k,j}, v_{k,j}\}_{j\in[n_k]}$ to all the other parties.
2. Define $S_o = \cup_{k\in[N],j\in[n_k]}\{pk_{k,j}, v_{k,j}\}$; after the previous step, each party can now compute S_o. Each party k further computes $\sigma_{k,i} \xleftarrow{\$} \mathsf{Sign}(sk_{k,i}, \mathsf{hist}, S_o)$ for all i, $1 \le i \le m_k$.
3. Each party k sends the set $\{pk_{k,i}, \sigma_{k,i}\}_{i\in[m_k]}$ to all the other parties.
4. Define $S_i = \cup_{k\in[N],i\in[m_k]}\{pk_{k,i}, \sigma_{k,i}\}$; after the previous step, each party can now compute S_i. The final transaction is $\mathsf{tx} = (S_i, S_o)$, which each party can broadcast to the Bitcoin network.

Briefly, each party broadcasts their desired output keys to the other parties, individually signs the completed list, and then broadcasts the signatures to the other parties; each party can then broadcast this list of signatures and recipients — presumably in randomized order to prevent trivial de-anonymization — as the collective transaction. The way in which the order and the transaction fees are determined can be used to identify the behavior of different Coinjoin services, as we will see in Section 4.

To attempt to understand the growing popularity of coinjoin transactions, we looked at how many transactions matched a rough pattern of what we might expect coinjoins[3] to look like. We defined this pattern as any transaction having more than five inputs and more than five outputs, and looked at how many transactions matched this pattern (which is admittedly only roughly correlated with coinjoins, but as coinjoins are indistinguishable from other transactions we

[3] In the rest of the paper, we follow the convention of called the protocol Coinjoin and the resulting transactions coinjoins.

cannot do better than an approximation). Prior to August 2013, we saw an average of zero matching transactions per block, but afterwards we saw a steady increase: first to an average of two matching transactions per block in December 2013, then to a peak of 14 in March 2014, and finally settling to an average of 10 in July 2014, where it has remained ever since.

2.3 Taint resistance

As mentioned above, the existing notions of unlinkability for electronic cash require that a valid coin belonging to one user is indistinguishable from a valid coin belonging to another. In Bitcoin, it is impossible to satisfy this definition: a bitcoin essentially *is* its spending history, and it is thus trivial to distinguish two valid bitcoins. Any notion of anonymity that is useful for Bitcoin must therefore focus less on the coins themselves and more on ownership. Looking back at the Coinjoin protocol in Section 2.2, we can see that it is attempting to obscure not the individual origins of the bitcoins involved — as, again, this can be trivially discerned using the public ledger — but rather the ownership of the bitcoins at each point in their spending histories.

To focus more on ownership, we thus present a notion of anonymity called *taint resistance*, which attempts to capture how well an adversary can discern the ownership of a bitcoin based on its previous spending history. Our definition has the advantage that we can not only provide proofs of security (i.e., prove that a protocol achieves optimal taint resistance), but that it also provides a concrete measurement of the degree to which a proposed solution such as Coinjoin is effective in improving anonymity.

In order to define taint resistance, we must first define what it means for bitcoins to be tainted at all. The naïve version of the definition is simple: if a public key has received some bitcoins as the result of a transaction, then all the inputs to that transaction have tainted those bitcoins (or, in a public-key-based definition, have tainted that output public key). As this is exactly the type of taint analysis that protocols like Coinjoin are designed to thwart, however, we consider a more specific definition.

Definition 2.1 (Taint set). *For a coinjoin* tx *produced as specified in Algorithm 2.1 and a public key* $pk \in \mathsf{outputs}(\mathsf{tx})$*, the* taint set *for* pk *is the set* $T = \{pk_i\}_{i \in [m]}$ *such that* $v_{i,j} \neq 0$*.*

The definition of the taint set only makes sense when applied to "real" coinjoins, since it explicitly mentions the values $v_{i,j}$ transferred from U_i to U_j. In general, it is impossible to define a taint set unless we make some assumption on how the transaction was generated; to see why, recall the example in the introduction, where Alice and Bob pay 1 coin each to Charlie and Dora. If one looks only at the transaction, there are potentially exponentially many explanations for the transaction; i.e., for each value $v < 1$, Alice and Bob could have been paying $(v, 1 - v)$ respectively to Charlie and $(1 - v, v)$ to Dora (or vice versa).

To consider the success of an adversary, we use the Matthews correlation coefficient (MCC), which is a way to measure the quality of a binary classifier. This fits well with an adversary attempting to de-anonymize transactions, as we can view it as assigning a 'yes' value to an input if it thinks it taints a given output, and a 'no' value if it does not. The MCC ranges from -1 to $+1$: -1 indicates a complete mismatch between the underlying truth and the classification, 0 indicates that the classifier does no better than random, and $+1$ indicates a perfect match. (We define $0/0 = 0$.)

For our purposes, we can define all the relevant terms in the original MCC formula using the set A output by an adversary, the underlying taint set T, and the full set of input keys S. Plugging these in, we get the following definition:

Definition 2.2 (Accuracy). *For a transaction* tx*, a public key* $pk \in \mathsf{outputs}(\mathsf{tx})$ *and its corresponding taint set* T*, auxiliary information* $aux \in \{0,1\}^*$*, and an adversary* $\mathcal{A}$*, we say that the adversary achieves* ϵ-accuracy *with respect to* aux*, where*

$$\epsilon = \frac{|A \cap T| \cdot |S \setminus (A \cup T)| - |A \setminus T| \cdot |T \setminus A|}{\sqrt{|A| \cdot |T| \cdot |S \setminus T| \cdot |S \setminus A|}}$$

for $S \leftarrow \mathsf{inputs}(\mathsf{tx})$ *and* $A \xleftarrow{\$} \mathcal{A}(\mathsf{tx}, aux)$*.*

Definition 2.3 (Taint resistance). *For a transaction* tx*, a public key* $pk \in \mathsf{outputs}(\mathsf{tx})$*, auxiliary information* $aux \in \{0,1\}^*$*, and a set of public keys* $S \subseteq \mathsf{inputs}(\mathsf{tx})$*, we say that* tx *achieves* ϵ-taint resistance *with respect to* aux *if no (potentially unbounded) adversary* $\mathcal{A}$ *can achieve more than* $(1-\epsilon)$*-accuracy.*

Unlike accuracy, taint resistance is quantified over all possible adversaries (it is a property of the transaction, not the adversary), and taint resistance ranges from 0 to 1 since there is always a trivial adversary that outputs S or the empty set and thus achieves accuracy 0. In the sequel, we first identity constructive solutions for achieving taint resistance, and then attempt to analyze the effect that the auxiliary information *aux* can have on a transaction's taint resistance.

3 Achieving Taint Resistance

In this section, we describe and analyze different mechanisms for achieving taint resistance. All of the solutions use well-known methods from the cryptographic literature; i.e., we do not claim any novelty in the cryptographic tools, but instead seek to explore how they can be used to achieve taint resistance.

It is instructive to start our discussion by describing how an adversary can identify the taint set of a given public key in a coinjoin; this will also be useful in our experimental analysis in Section 4. We describe a few scenarios below:

- **The transaction itself.** A coinjoin has two input values x and y and two output values $x - F_x$ and $y - F_y$ (where the transaction fee is $F_x + F_y$). If $x \neq y$ and F_x and F_y are small, the adversary can use this information to increase his belief in which input address transferred value to a given output address.

– **Participating in coinjoins.** An unlucky user performs a coinjoin with the adversary as his sole mixing partner. Regardless of how the coinjoin is performed, the adversary — who controls one input address and one output address — can trivially compute the taint set of the other output address (assuming one input and one output address per participant).
– **Influencing the creation of coinjoins.** An adversary can learn information about the taint set of an output address in a coinjoin by maliciously influencing the way in which a coinjoin is created. This includes not only the protocol for generating the coinjoin, but also the process of choosing the other participants (where, for example, the adversary might try to force an honest user to perform coinjoins with adversarially controlled addresses).

3.1 Using a trusted server

We describe now an ideal setting for performing coinjoins; i.e., one that leads to optimal taint resistance. We then proceed to replace some of the (unrealistic) assumptions using cryptographic tools.

To start, we assume a central trusted server, which partitions users into different coinjoins, permutes their addresses at random, and transfers an amount equal to the *minimum* of all input values (and returns the differences to change addresses).

Grouping users: A user U^j who wish to perform a coinjoin sends his addresses (pk_i^j, pk_o^j, pk_c^j) (for *input address*, *output address* and *change address* respectively) to a central server S, where pk_o^j and pk_j^c are freshly generated addresses. The server S randomly assigns the user to a bucket ℓ of size n.

Performing transactions: Let $U^1, \ldots, U^n$ be the users assigned to a certain bucket ℓ. Let v_i^j be the values associated to pk_i^j and $v^* = \min(v_i^j)$. The server S prepares a transaction with input addresses $(pk_i^1, \ldots, pk_i^n)$ and outputs addresses $(pk_o^{\pi(1)}, pk_o^{\pi(n)}, pk_c^1, \ldots, pk_c^n)$, for a random permutation π. Let F be the transaction fee: then for each j, the address pk_c^j receives $v_c^j = v_i^j - v^* - F/n$ coins, and all pk_o^j receive the same value v^*.

Signing transactions: The server S sends the transactions tx to all users U^j in the bucket ℓ. If U^j's output address is present in the transaction and the amount received by the change address of U^j is correct, U^j signs the transaction and sends it back to the server.

Claim 1 *Any* tx *generated using the above protocol achieves an expected* 1*-taint resistance for the first* n *output addresses, against every adversary who does not control* $(U^1, \ldots, U^n, S)$.

The proof of the claim is trivial: the first n output addresses are fresh unused addresses in a randomly permuted order, are generated independently, and all receive the same amount of bitcoins. Since the adversary does not control any other user participating in the transaction nor the server, the adversary has no auxiliary information about the permutation π.

3.2 Reducing trust in the central server

The solution presented in the previous section offers taint resistance against only simplistic adversaries. We propose here a solution that also works against a partially corrupted server. The solution uses anonymous channels in a crucial way and is a simple application of the results of [9] in this setting.

Grouping users: User U^j sends his input and change address (pk_i^j, pk_c^j) to the server S, which replies with the index ℓ of the bucket to which the user has been assigned. Now, *using a (different) anonymous channel*,[4] user U^j sends the pair (ℓ, pk_o^j) to the server.
Performing Transactions: As before.
Signing Transaction: As before.

Intuitively, since the users communicate the input and output key to the server using two distinct anonymous channels, the server cannot link those addresses together.

Claim 2 *Any* tx *generated using the above protocol achieves an expected* $(1-\epsilon)$*-taint resistance for the first* n *output addresses, against every adversary who does not control* $(U^1, \ldots, U^n)$ *and corrupts* S *in a passive way.*

Since S is passively corrupted, we need to add S's view to the auxiliary information *aux* given to the adversary. The main idea is that, since each user sends its output public key using a fresh anonymous channel, even the central server does not learn the mapping between the input keys and output keys.

More formally, as shown by [9], our use of the anonymous channel is a secure implementation of an ideal functionality that performs a secure shuffle. Therefore, the view of the server can be efficiently simulated, which implies that any adversary that can achieve non-negligible accuracy in this protocol can be used to do so in the previous protocol as well.

3.3 Removing the central server

Finally, we sketch a solution that does not require any central server. Here we think of a high number of users N who want to perform coinjoins and have access to some broadcast channel. It is at this point (conceptually) trivial to let all parties run a secure computation protocol to replace the central server; in practice, however, this is quite cumbersome, as it requires high communication and computational resources.

Instead, we seek a solution that allows the users to partition themselves in smaller sets of (expected) size n and then perform simple "mix-nets" between them (a very similar solution has also been described in [17]). Here it is crucial that users are assigned to groups at random (even in the presence of other actively corrupted users) to guarantee that an adversary who controls few (say $n-1$) parties cannot force an honest user to perform a coinjoin with those addresses. Let $H : \{0,1\}^* \to [n]$ be a cryptographic hash function.

[4] This can implemented by creating a new identity on Tor.

Grouping users: All users perform a *simultaneous exchange* of their input and change addresses (pk_i^j, pk_c^j). (This can be implemented by having all parties commit to their addresses, and then send the opening only after all commitments have been received). Each user j is assigned to group $H(pk_i^j)$.

Performing transactions: Let $U^1, \ldots, U^n$ be the users assigned to a certain bucket ℓ. Now these users can perform a simple mix-nets as follows: User U^1 encrypts his output address pk_o^j under all the public keys of the other parties—i.e., forms $C_2^1 = E_{pk_2}(E_{pk_3}(\cdots(E_{pk_n}(pk_o^j))))$—and sends it to U^2. Now U^2 samples a random permutation π of $\{1, 2\}$, decrypts the received ciphertexts and computes $C_3^{\pi(1)} = D_{sk_2}(C_2^1)$ as well as $C_3^{\pi(2)} = E_{pk_3}(E_{pk_4}(\cdots(E_{pk_n}(pk_o^j))))$ and so on. That is, at every step user U^j removes one layer of encryption from all the ciphertexts he receives, encrypts his own output public key under the public keys of the next users, shuffles the ciphertexts and sends them on to the next user. Finally, U^n decrypts and obtains the full list of the output addresses, and prepares a transaction tx in the same way as the server did in the previous solutions.

Signing Transaction: The user U^n sends the transactions tx to all users U^j in the bucket ℓ. If pk_o^j is present in the transaction and the amount received by pk_c^j is correct, U^j signs the transaction and sends it back to U^n.

Claim 3 *Any* tx *generated using the above protocol achieves an expected* $(1 - n\tau^{n-1})$*-taint resistance for the first* n *output addresses, against every adversary who controls a fraction* τ *of parties.*

There are two key ideas here. First, there is no guarantee that a user will perform the correct decryption and shuffle. Still, it suffices to use a passively secure mix-nets protocol, as users will sign the transaction only if their address is present in the final transaction. Second, unless the adversary controls exactly $n-1$ users in a given bucket, then the resulting mix between the two hones user will leave the adversary without any information to do any better than guessing. Since the adversary cannot control how the users are assigned into buckets (due to the use of commitments and the hash function), assuming that the adversary controls less than $\tau \cdot N$ parties then the probability that he controls exactly the other $n - 1$ users in the bucket assigned to the target public key is less than $n \cdot \tau^{n-1}$.

4 Experimental Analysis

In the previous section, we saw methods for achieving taint resistance. All of these approaches, however, achieved security only with respect to the auxiliary information that an adversary could obtain. In this section, we explore this auxiliary information and consider how much it could affect taint resistance.

We consider different forms of auxiliary information from two perspectives. First, we consider a minimal *passive* adversary that can glean information about coinjoins based only on what it sees in the block chain. To emulate such an

adversary, we need only download the Bitcoin block chain, which we did as recently as 2 September 2014 (at which point there were 318,768 blocks, 45.84 million transactions, and 45.79 million distinct public keys).

Next, we consider an *active* adversary that participates in coinjoins. To emulate this adversary, we used `blockchain.info`'s SharedCoin service to participate in our own coinjoins. We used this service 54 times between 30 June 2014 and 11 August 2014; in each transaction, we sent 0.02 BTC from a single address we owned to a new one (freshly generated). As `blockchain.info` requires at least two repetitions of the Coinjoin protocol (in their own words, "a higher number of repetitions makes the transaction more difficult to trace and improves privacy"), this resulted in two transactions: in the first 0.0205 BTC was sent to a freshly generated intermediate address and the remainder was sent to a freshly generated change address, and in the second 0.02 BTC was sent to the address we specified. We therefore ended up with 108 distinct coinjoins, each of which took between 8 and 74 seconds to complete (on average, 30), and had between 4 and 40 input addresses (on average, 14.5) and between 4 and 42 output addresses (on average, 25.8).

4.1 Auxiliary information based on value

One bitcoin is divisible down to the eighth decimal place, meaning that it is often possible to end up with "jagged" bitcoin values. If each user in a coinjoin sends different jagged values, then — as discussed in Section 3 — these values might help an adversary discover the permutation between input and output addresses. This is an acknowledged limitation of the Coinjoin protocol, and the only solution for achieving full taint resistance is to ensure that all participants send the same amount of bitcoins. From a usability perspective, however, this is not particularly desirable, so it is useful to understand the degree to which differing amounts really degrade taint resistance.

We first consider the different behaviors in which coinjoin users might engage. For example, a user might *aggregate* bitcoins by combining the balances of m separate addresses into one address, and a user might *split* bitcoins by moving the balance of one address into n separate addresses. (This latter case often arises in the case of making change, where $n = 2$.) We attempt to correlate the values in a coinjoin by identifying subsets of the input values whose sum adds up to an output value (the m-to-1 scenario) and which output values are part of a subset whose sum adds up to an input value (the 1-to-n scenario). As these are the only two behaviors that we engaged in in our own coinjoins (and because the subset sum problem is **NP**-complete), we do not consider the more general m-to-n setting.

Finally, because transactions have fees, we perform a *noisy* subset sum by allowing the sum of the subsets to potentially exceed the target value by at most the transaction fee. We present our algorithm in Algorithm 4.1.

As a sanity check, we ran this algorithm using the ground truth data collected from our coinjoins; i.e., for each of our known output addresses. (As we knew that we engaged only in 1-to-2 or 1-to-1 transactions, we also "cheated" by

Algorithm 4.1: `find_taint`: Output the taint set for a public key

Input: a transaction tx and a public key $pk \in \mathsf{outputs}(\mathsf{tx})$.

Compute the fee F for tx
Compute $S_{m\text{-to-}1} \leftarrow \texttt{noisy\_subset\_sum}(\mathsf{inputs}(\mathsf{tx}), \mathsf{val}(pk), F)$
$S_{1\text{-to-}n} \leftarrow \emptyset$
forall the $pk' \in \mathsf{inputs}(\mathsf{tx})$ **do**
    Compute $S \leftarrow \texttt{noisy\_subset\_sum}(\mathsf{outputs}(\mathsf{tx}), \mathsf{val}(pk'), F)$
    if $pk \in S$ **then**
        $S_{1\text{-to-}n} \leftarrow S_{1\text{-to-}n} \cup \{pk'\}$
return $S_{m\text{-to-}1} \cup S_{1\text{-to-}n}$

considering only subsets of size at most 2.) We then compared this to the known taint set to get the accuracy (see Definition 2.2) of an adversary running this algorithm, and plotted the results in Figure 1.

As we can see, the algorithm was fairly inconsistent in its accuracy. This is due in part to a quirk of `blockchain.info`'s service: in the first step, 0.0205 BTC was sent to an intermediate address, but in the second step, only 0.02 BTC was sent to the final address; the remainder was kept by `blockchain.info` as a service charge. As a result, our algorithm did not positively associate the input and output addresses, so any non-empty set it output was a false positive (as indicated by the largely non-positive MCC for the m-to-1 taint). For the first type of transaction, in which the initial address split its value between the change and intermediate addresses, we see that (unsurprisingly) the 1-to-n and combined taint did fairly well overall, and — especially for transactions with fewer input keys and thus fewer options — often found the taint set perfectly (as indicated by the points with an MCC of 1).

Active adversaries. To emulate the benefit that an active adversary has — that in a coinjoin in which it participated, it can rule out its own addresses — we started with our own coinjoins, eliminated our own addresses, and ran Algorithm 4.1 on the remainder. As we could no longer use ground truth data to compute the accuracy, we instead plotted the size of the set S. The results are in Figure 2.

On average, we can see that the set S was of a fairly small size; in the many cases that it was 0, either (1) a more general m-to-n Coinjoin was used, (2) the input subset was of size larger than 4, or (3) most likely, our basic algorithm did not take into account some quirk of the `blockchain.info` service (as was the case with our own transactions). In the cases where S was larger, we ultimately do not know whether it truly captured the taint set or whether it was simply capturing "noise" from other inputs that happened to sum to the target value. To nevertheless attempt to capture some of this noise, we considered for the case of m-to-1 taint not the possible taint set but rather how many subsets of input addresses summed to the correct output value. Here, a lower value indicates that the algorithm is more sure about which input addresses taint the given output (as there are fewer options), while a larger number of subsets indicates noise from the inputs (for example, if many of the inputs have the same value), and

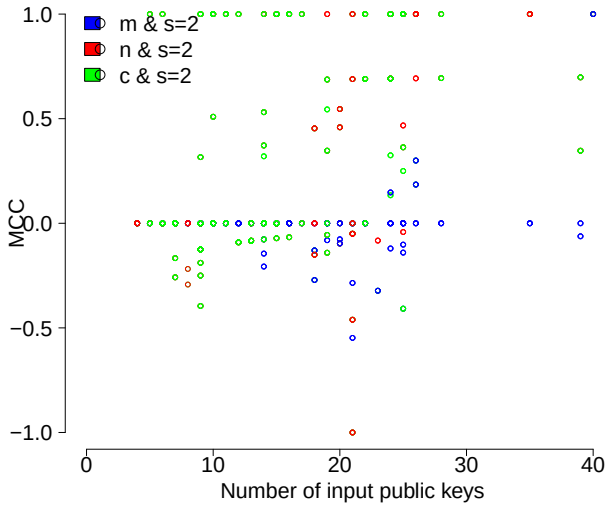

Fig. 1: For our known output addresses, the MCC for Algorithm 4.1 when run on coinjoins with differing numbers of input addresses. The points in blue represent the MCC when considering just m-to-1 taint ($S_{m\text{-to-}1}$), the points in red 1-to-n taint ($S_{1\text{-to-}n}$), and the points in green the combined taint set S. All use a maximum subset size of 2 when running `noisy_subset_sum`.

consequently less certainty. Of our 5156 data points, only 62 had exactly one matching subset (again, using only the m-to-1 taint), and 4812 had no matching subsets at all. Our simple active adversary was thus not particularly successful at identifying the taint set, although it did at least manage to avoid false positives.

Passive adversaries. Finally, we consider a passive adversary; i.e., one that does not participate in any coinjoins, but instead tries to infer from an examination of the block chain which transactions are coinjoins and which coinjoin input addresses taint which output addresses. To do this, we considered the same pattern used in Section 2.2: a potential coinjoin has more than five inputs, more than five outputs, and took place after 30 August 2013. This is of course a very rough heuristic, and before using this data we eliminated any obvious mismatches; this included behavior such as a user combining funds to place bets on different odds in dice and other gambling games. From this set, we then randomly sampled 100 transactions (to match the size of our set of real coinjoins). We again ran Algorithm 4.1 and recorded the size of the set S; the results are in Figure 3.

Again, we ultimately cannot know how successfully the analysis identified the taint set. We can, however, see a clear increase in the size of the set S for these passively identified coinjoins over the size for the definite coinjoins. Running the same analysis to determine the number of possible subsets in the m-to-1 scenario, we found that of our 3476 data points, 278 had exactly one matching subset and 2513 had no matching subsets. The subsets produced by the algorithm were thus

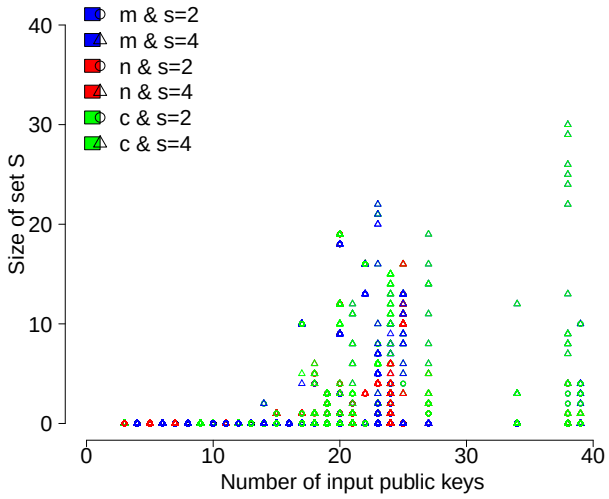

Fig. 2: For the unknown addresses in known coinjoins, the size of the set output by Algorithm 4.1. The points in blue consider just m-to-1 taint ($S_{m\text{-to-}1}$), the points in red 1-to-n taint ($S_{1\text{-to-}n}$), and the points in green the combined taint set S. The value of s is the maximum subset size when running `noisy_subset_sum`.

far noisier than the ones produced for the definite coinjoins, indicating that an active adversary has a distinct advantage over a passive one.

5 Related Work

We consider related work that both proposes ways to enhance privacy in virtual currencies and attempts to analyze the existing anonymity in Bitcoin.

In terms of the latter, perhaps the work most similar to our own is Coinjoin Sudoku [3], in which Atlas analyzes the same mixing service (namely, the SharedCoin one provided by `blockchain.info`) and finds that it "offers only limited privacy to users due to weaknesses in its design." As we did, Atlas performed his own coinjoins and examined possible relationships between input and output addresses based on the values. He claims to be able to taint the majority of inputs and outputs, but the details of the analysis are not given and as of this writing no source code or tool is available, despite a promised release date of 23 June 2014. A recent line of research has examined anonymity in the overall Bitcoin network [15,16,2,12,19], and a recent paper by Möser et al. found that centralized mix services do make tracing transactions significantly more difficult [14].

In terms of privacy enhancements, alternative virtual currencies have been proposed such as Zerocash [13,4] and Darkcoin [1]. There are also a number of proposed overlays for Bitcoin, such as Pinocchio Coin [6], Mixcoin [5], CoinWitness [11], and CoinShuffle [17]. None of these papers formally prove the security of their proposed constructions, but our decentralized construction is almost

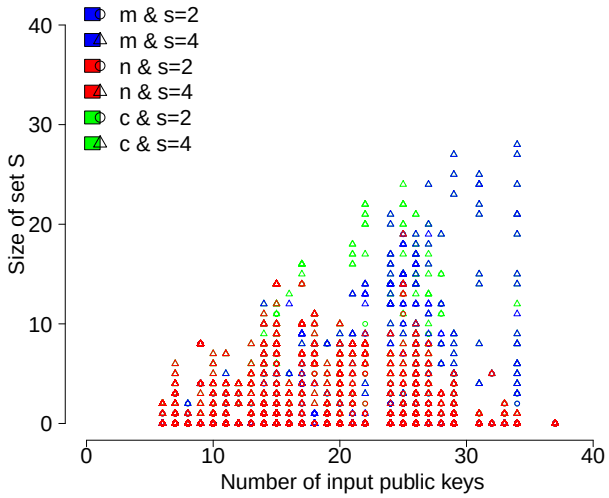

Fig. 3: For potential coinjoins, the size of the set output by Algorithm 4.1. The points in blue consider just m-to-1 taint ($S_{m\text{-to-}1}$), the points in red 1-to-n taint ($S_{1\text{-to-}n}$), and the points in green the combined taint set S. The value of s is the maximum subset size when running `noisy_subset_sum`.

identical to the one in CoinShuffle and our centralized construction is related to the one in Mixcoin.

6 Conclusions and Open Problems

In this paper, we presented a definitional framework for the anonymity that virtual currencies such as Bitcoin provide. We then provided constructive solutions for achieving this new notion of anonymity, and analyzed the extent to which it was already being achieved by existing Bitcoin overlays. For both of our results, several interesting open problems and extensions remain. Our constructions require additional cryptographic techniques, and it is important to understand the overhead that these techniques require. Similarly, our analysis of SharedCoin was relatively simplistic and did not take into account certain quirks of the service being used. Extending the analysis to consider these quirks — or understanding the extent to which doing so would affect scalability — would provide a more complete picture of the successes and limitations of the Coinjoin protocol.

Acknowledgments

The second author was supported by the Danish National Research Foundation (under the grant 61061130540, CTIC research center) and by the Danish Strategic Research Council (CFEM research center) within which this work was performed.

References

1. Darkcoin. https://www.darkcoin.io.
2. E. Androulaki, G. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating User Privacy in Bitcoin. In *Proceedings of Financial Cryptography 2013*, 2013.
3. K. Atlas. Coinjoin Sudoku. http://www.coinjoinsudoku.com.
4. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2014.
5. J. Bonneau, J. Clark, J. A. Kroll, A. Miller, and A. Narayanan. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *Proceedings of Financial Cryptography 2014*, 2014.
6. G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno. Pinocchio Coin: Building Zerocoin from a Succinct Pairing-based Proof System. In *Proceedings of PETShop 2013*, 2013.
7. European Central Bank. Virtual Currency Schemes. ECB Report, Oct. 2012. www.ecb.europa.eu/pub/pdf/other/virtualcurrencyschemes201210en.pdf.
8. Federal Bureau of Investigation. (U) Bitcoin Virtual Currency Unique Features Present Distinct Challenges for Deterring Illicit Activity. Intelligence Assessment, Cyber Intelligence and Criminal Intelligence Section, Apr. 2012. cryptome.org/2012/05/fbi-bitcoin.pdf.
9. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Cryptography from anonymity. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 239–248, 2006.
10. G. Maxwell. CoinJoin: Bitcoin privacy for the real world. Post on bitcointalk.org. https://bitcointalk.org/index.php?topic=279249.
11. G. Maxwell. Really Really ultimate blockchain compression: CoinWitness. Post on bitcointalk.org. https://bitcointalk.org/index.php?topic=277389.
12. S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In *Proceedings of IMC 2013*, 2013.
13. I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2013.
14. M. Möser. An Inquiry into Money Laundering Tools in the Bitcoin Ecosystem. In *Proceedings of the IEEE 2013 eCrime Researchers Summit*, 2013.
15. F. Reid and M. Harrigan. An Analysis of Anonymity in the Bitcoin System. In *Security and Privacy in Social Networks*, pages 197–223. Springer New York, 2013.
16. D. Ron and A. Shamir. Quantitative Analysis of the Full Bitcoin Transaction Graph. In *Proceedings of Financial Cryptography 2013*, 2013.
17. T. Ruffing, P. Moreno-Sanchez, and A. Kate. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In *Proceedings of ESORICS 2014*, 2014.
18. Securities and Exchange Commission. SEC Charges Texas Man With Running Bitcoin-Denominated Ponzi Scheme, July 2013. www.sec.gov/News/PressRelease/Detail/PressRelease/1370539730583.
19. M. Spagnuolo, F. Maggi, and S. Zanero. BitIodine: Extracting Intelligence from the Bitcoin Network. In *Proceedings of Financial Cryptography 2014*, 2014.