# Misbehavior in Bitcoin: A Study of Double-Spending and Accountability

GHASSAN O. KARAME, NEC Laboratories Europe
ELLI ANDROULAKI, IBM Research Zurich
MARC ROESCHLIN, ARTHUR GERVAIS, and SRDJAN ČAPKUN, ETH Zurich

Bitcoin is a decentralized payment system that relies on Proof-of-Work (PoW) to resist double-spending through a distributed timestamping service. To ensure the operation and security of Bitcoin, it is essential that all transactions and their order of execution are available to all Bitcoin users.

Unavoidably, in such a setting, the security of transactions comes at odds with transaction privacy. Motivated by the fact that transaction confirmation in Bitcoin requires tens of minutes, we analyze the conditions for performing successful double-spending attacks against fast payments in Bitcoin, where the time between the exchange of currency and goods is short (in the order of a minute). We show that unless new detection techniques are integrated in the Bitcoin implementation, double-spending attacks on fast payments succeed with considerable probability and can be mounted at low cost. We propose a new and lightweight countermeasure that enables the detection of double-spending attacks in fast transactions.

In light of such misbehavior, accountability becomes crucial. We show that in the specific case of Bitcoin, accountability complements privacy. To illustrate this tension, we provide accountability and privacy definition for Bitcoin, and we investigate analytically and empirically the privacy and accountability provisions in Bitcoin.

Categories and Subject Descriptors: K.4.1 [**Computers and Society**]: Public Policy Issues—*Privacy*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Payment schemes, security*

General Terms: Design, Economics, Experimentation, Security

Additional Key Words and Phrases: Monetization, distributed computing, security, privacy, Bitcoin, double-spending

**ACM Reference Format:**
Ghassan O. Karame, Elli Androulaki, Marc Roeschlin, Arthur Gervais, and Srdjan Čapkun. 2015. Misbehavior in Bitcoin: A study of double-spending and accountability. ACM Trans. Info. Syst. Sec. 18, 1, Article 2 (May 2015), 32 pages.
DOI: http://dx.doi.org/10.1145/2732196

## 1. INTRODUCTION

First introduced in 2008, Bitcoin [Nakamoto 2009] is an emerging digital currency that is currently integrated across a number of businesses [BitcoinTrade 2013] and exchange markets (e.g., BitcoinCharts [2013]).

Bitcoin is a Proof-of-Work (PoW)-based currency that allows users to generate digital coins by performing computations. Users execute payments by digitally signing

Authors' addresses: G. O. Karame, Security Group, NEC Laboratories Europe, 69115 Heidelberg, Germany; email: ghassan.karame@neclab.eu; E. Androulaki, Storage Integrity Group, IBM Research Zurich, Switzerland; email: lli@zurich.ibm.com; M. Roeschlin, A. Gervais, and S. Čapkun, Department of Computer Science, ETH Zurich, 8092, Zurich, Switzerland; emails: romarc@student.ethz.ch, {arthur.gervais, capkuns}@inf.ethz.ch.

DOI: http://dx.doi.org/10.1145/2732196

their transactions and are prevented from double-spending their coins (i.e., signing over the same coin to two different users) through a distributed timestamping service [Nakamoto 2009]. This service operates on top of the Bitcoin Peer-to-Peer (P2P) network, which ensures that all transactions and their order of execution are visible to all Bitcoin users.

Today, Bitcoin is increasingly used in a number of “fast payment” scenarios, where the exchange time between the currency and goods is short. Examples include vending machine payments and fast-food payments (recently featured in media reports on Bitcoin [CNN 2011]), where the payment is followed by fast (in the order of a minute) delivery of goods. The Bitcoin PoW-based timestamping mechanism is essential for the detection of double-spending attacks (i.e., in which an adversary attempts to use some of her coins for two or more payments), but it requires tens of minutes to verify a transaction and is therefore inappropriate for fast payments. Since Bitcoin users are encouraged to hold many accounts, there is only limited value in verifying the payment after the user has obtained the goods (and left the store) or services (e.g., access to online content). The developers of Bitcoin implicitly acknowledge the problem of verifying fast payments and inform users that they do not need to wait for the payment to be verified as long as the transaction has been released in the network [BitcoinFAQ 2013]. This, however, as we show, does not prevent double-spending.

In this work, we start by analyzing double-spending attacks on fast Bitcoin payments and show that unless appropriate detection techniques are integrated in current Bitcoin clients, double-spending attacks on fast payments succeed with overwhelming probability and can be mounted against current Bitcoin clients at low cost. We further show that the detection measures recommended by Bitcoin developers are not always effective in detecting double-spending; we argue that even if those recommendations are followed, double-spending attacks on Bitcoin are still possible. Leveraging our findings, we propose and implement a modification to the current Bitcoin implementation that ensures the detection of double-spending attacks against fast payments. A variant based on our proposed technique is integrated in BitcoinXT [2014].

Given the increasing use of Bitcoin, misbehavior in Bitcoin is only expected to increase. Motivated by our double-spending investigation, we then proceed to analytically and empirically investigate the degree of privacy and accountability currently offered by Bitcoin. Our investigation aims at determining (1) the extent to which misbehaving users/profiles can be identified and (2) the privacy and accountability that Bitcoin offers to its users.

This article extends and improves our prior work in Karame et al. [2012] and Androulaki et al. [2013] with significant new material. More specifically, our contributions in this work can be summarized as follows:

—We measure and analyze the time required to confirm transactions in Bitcoin. Our analysis shows that transaction confirmation in Bitcoin can be modeled with a shifted geometric distribution with an average transaction confirmation time of 10 minutes and a standard deviation of approximately 20 minutes. We argue that this hinders the reliance on transaction confirmation when dealing with fast payment scenarios.

—We thoroughly analyze the conditions for performing successful double-spending attacks against fast payments in Bitcoin. We then present the first realization of double-spending attacks on fast payments in Bitcoin using a handful of hosts located around the globe.[1] Here, we extend our work in Karame et al. [2012] and show how an adversary can exploit block forks and version changes in Bitcoin to perform such double-spending attacks.

[1] In our experiments, we solely used Bitcoin wallets and accounts that we own; other Bitcoin users were not affected by our experiments.

—We explore the privacy and accountability provisions of Bitcoin. More specifically, we adapt existing privacy notions to the Bitcoin context and investigate analytically and experimentally the privacy and accountability provisions of Bitcoin. In this respect, we extend our analysis in Androulaki et al. [2013] and collect statistics acquired from the first 239,200 Bitcoin blocks using two of our proposed heuristics. We also extend our simulation results to categorize the privacy leakage in Bitcoin with respect to the user activity in Bitcoin (i.e., number of transactions performed by users). Finally, we show that in the case of Bitcoin, accountability can be seen as the complement of privacy.

The remainder of the article is organized as follows. In Section 2, we briefly describe Bitcoin. In Section 3, we analyze and evaluate the security of fast payments with existing Bitcoin clients. In Section 4, we evaluate the privacy and accountability provisions of the Bitcoin system. In Section 5, we overview related work. We conclude the article in Section 6.

## 2. BACKGROUND ON BITCOIN

Bitcoin is a decentralized P2P payment system [Nakamoto 2009] that was introduced in 2008. Electronic payments are performed by generating *transactions* that transfer Bitcoin coins (BTCs) among Bitcoin peers. These peers are referenced in each transaction by means of virtual pseudonyms—referred to as Bitcoin addresses. Each address is mapped through a transformation function to a unique public/private key pair. These keys are used to transfer the ownership of BTCs among addresses.

Peers transfer coins to each other by issuing a transaction. A transaction is formed by digitally signing a hash of the previous transaction where this coin was last spent along with the public key of the future owner and incorporating this signature in the coin [Nakamoto 2009]. Transactions take as input the references to an output of another transaction that spends the same coins, then outputs the list of addresses that can collect the transferred coins. Any peer can verify the authenticity of a BTC by checking the chain of signatures.

Transactions are included in Bitcoin *blocks* that are broadcasted in the entire network. To prevent double-spending of the same BTC, Bitcoin relies on the synchronous communication assumption along with a hash-based PoW concept. More specifically, to generate a block, Bitcoin peers, or *miners*, must find a nonce value that when hashed with additional fields (i.e., the Merkle hash of all valid and received transactions, the hash of the previous block, and a timestamp), the result is below a given target value. If such a nonce is found, miners then include it (as well as the additional fields) in a new block, thus allowing any entity to verify the PoW. Upon successfully generating a block, a miner is granted a number of BTCs (25 new BTCs since block 210,000). This provides an incentive for miners to continuously support Bitcoin. The resulting block is forwarded to all peers in the network, who can then check its correctness by verifying the hash computation. If the block is deemed to be “valid,”[2] then the peers append it to their previously accepted blocks. Since each block links to the previously generated block, the Bitcoin block *chain* grows upon the generation of a new block in the network. Note that when miners do not share the same view in the network (e.g., due to network partitioning), they might work on different block chains, thus resulting in “forks” in the block chain. Block forks are inherently resolved by the Bitcoin system; the longest block chain will eventually prevail. In rare occasions, the Bitcoin developers can force one chain to be adopted on the expense of others [Gervais et al. 2014b]. Transactions not appearing in blocks that are part of the main block chain (i.e., the longest) will

[2] That is, the block contains correctly formed transactions that have not been previously spent and has a correct PoW.

be readded to the pool of transactions in the system and reconfirmed in subsequent blocks.

The main intuition behind Bitcoin is that for peers to double-spend a given BTC, they would have to replace the transaction where the BTC was spent and the corresponding block in which it appeared; otherwise, their misbehavior would be detected immediately. This means that for malicious peers to double-spend a BTC without being detected, they would not only have to redo all of the work required to compute the block where that BTC was spentbut also would have to recompute all subsequent blocks in the chain. The older a Bitcoin transaction, and thus the deeper it is included in the block chain, the harder it becomes to modify/double-spend the transaction.

Further details on Bitcoin can be found in Nakamoto [2009], BitcoinRules [2012], and BitcoinProtocol [2013]. In what follows, we provide a summary (adapted from Nakamoto [2009]) of the steps that peers undergo in Bitcoin when a payment occurs.

- New transactions are broadcasted by peers in the network.
- When a new transaction is received by a peer, it checks whether the transaction is correctly formed and whether the BTCs have been previously spent in a block in the block chain. If the transaction is correct, it is stored locally in the *memory pool* of peers until it is included in a valid block. In the article, we refer to a transaction that appears in the memory pools of peers as a zero-confirmation transaction.
- Miners work on constructing a new block. If they find a PoW, they include all of the transactions that appear in their memory pool within the newly formed block. Miners then broadcast the block in the network. Transaction that are included in well-formed blocks are called *confirmed transactions*.[3]
- When peers receive a new block, they verify that the block hash is valid and that every transaction included within the block has not been previously spent. If the block verification is successful, miners continue working toward constructing a new block using the hash of the last accepted block in the "previous block" field.

## 3. SECURITY ANALYSIS OF FAST BITCOIN PAYMENTS

In what follows, we analyze and evaluate the effectiveness of double-spending attacks on fast Bitcoin payments (i.e., where the exchange between currencies and services happens simultaneously). Leveraging our findings, we propose and implement a modification to the current Bitcoin implementation to accelerate the detection of double-spending attacks against fast payments.

### 3.1. Model

Our system consists of a malicious client $\mathcal{A}$ and a vendor $\mathcal{V}$ connected through a Bitcoin network. We assume that $\mathcal{A}$ wishes to acquire a service from $\mathcal{V}$ without having to spend its BTCs. More specifically, $\mathcal{A}$ could try to double-spend the coin that she already transferred to $\mathcal{V}$. By double-spending, we refer to the case where $\mathcal{A}$ can redeem and use the same coins with which she paid $\mathcal{V}$ so as to acquire a different service elsewhere.

We assume that $\mathcal{A}$ can only control few peers in the network (that she can deploy since Bitcoin does not restrict membership) and does not have access to $\mathcal{V}$'s keys or machine. The remaining peers in the network are assumed to be honest and to correctly follow the Bitcoin protocol. In this article, we assume that $\mathcal{A}$ does not participate in the block generation process. This also suggests that when a transaction is confirmed in a block, this transaction cannot be modified by $\mathcal{A}$. To hide her profile, we assume that $\mathcal{A}$ generates a new Bitcoin address whenever it communicates with $\mathcal{V}$.

[3] In current Bitcoin clients, a transaction has to receive six confirmations before it is added to the user's wallet.

### 3.2. Transaction Confirmation Time

As described in Section 2, the most conventional way for a vendor $\mathcal{V}$ to accept a payment made by a customer $\mathcal{C}$ is to wait until the transaction issued from $\mathcal{C}$ to $\mathcal{V}$ is confirmed in at least one block before offering service to $\mathcal{C}$. In what follows, we analyze the block generation times in Bitcoin.

To generate a block, miners *work* on constructing a PoW. In particular, given the set of transactions that have been announced since the last block's generation, and the hash of the last block, Bitcoin miners need to find a nonce such that

$$\texttt{SHAd256}\{\mathsf{Bl}_l \;||\; \mathrm{MR}(\mathrm{TR}_1, \ldots, \; \mathrm{TR}_n) \;||\; \mathrm{No}\} \leq \mathit{target}, \qquad (1)$$

where $\texttt{SHAd256}$ is the $\texttt{SHA-256}$ algorithm applied twice, $\mathsf{Bl}_l$ denotes the last generated block, $\mathrm{MR}(\overline{x})$ denotes the root of the Merkle tree with elements $\overline{x}$, $\mathrm{TR}_1 \;||\; \ldots \;||\; \mathrm{TR}_n$ is a set of transactions that have been chosen by the miners to be included in the block,[4] No is the 32-bit nonce, and *target* is a 256-bit number. To generate the PoW, each miner chooses a particular subset of the candidate solutions' space and performs brute-force search. It is apparent that the bigger the *target*, the easier it is to find a nonce that satisfies the PoW.

In the following, we show (1) that the success of each miner in generating a block within sufficiently small time intervals can be simulated as a Bernoulli trial and (2) that the success probability of block generation within a sequence of small time intervals corresponds to successive Bernoulli trials with substitution (which is also known as shifted geometric distribution [ProofWiki 2013]). For the purpose of our analysis, we note the following:

(1) The probability of success in a single nonce trial is negligible. Taking into consideration that SHA-256 is a pseudorandom permutation function, each of the $2^{32}$ nonces has $\frac{\mathit{target}}{2^{256}-1}$ probability of satisfying the PoW.
(2) Miners compute their PoW independently; as such, the probability that one of them succeeds does not depend on the progress of PoW of the others.
(3) Miners frequently restart the generation of their PoW, and whenever a new transaction is added to the memory pool of a miner, the Merkle root (included in the block) changes.
(4) For the sake of our analysis, we approximate the time interval between the announcement of successive transactions as follows. We extract the various block generation times from the Bitcoin block explorer and assume that transactions are announced uniformly at random between two successive block generations. Our findings (Figure 1) show that the time interval between the announcement of most pairs of successive transactions is less than 15 seconds. Therefore, we assume in the sequel that the PoW for block generation is restarted approximately every dt $\approx$ 15 seconds.

Given the first two observations, the probability of a miner in succeeding in an *individual* block generation attempt can be modeled as an independent Bernoulli process with success probability $\varepsilon = \frac{\mathit{target}}{2^{256}-1}$. Based on the last observation, we claim that *consecutive* block generation attempts can be modeled as sequential Bernoulli trials with *replacement*. Our claim for replacement is justified by the fact that maximum possible

[4] These transactions are chosen from the transactions that have been announced (and not yet confirmed) since $\mathsf{Bl}_l$'s generation.

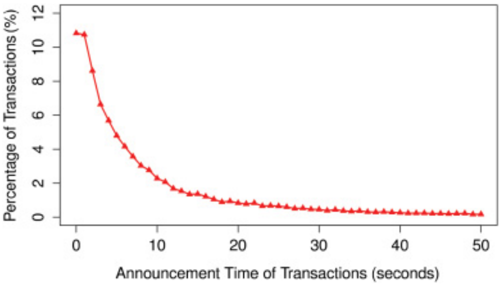


Fig. 1. Distribution of the announcement times of transactions. We assume that transactions are announced uniformly at random within two successive blocks.

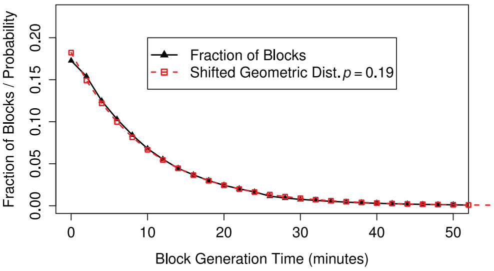


Fig. 2. Block generation times in Bitcoin. Assuming a (time) bin size of 2 minutes, the block generation function can be fitted to a shifted geometric distribution with $p = 0.19$.

PoW progress performed by a miner (expressed as a number of hash calculations) before its PoW resets is negligible in comparison to $2^{256} - 1$. This is the case since the PoW progress approximates $2^{35} \ll 2^{256} - 1$ given the computing power of most Bitcoin miners [Mining 2013; MiningHardware 2013].

Let $n_i$ refer to the number of attempts that a miner $\mathsf{m}_i$ performs within a time period $\delta$. Typically, $\delta$ is in the order of few minutes. The probability $p_i$ of $\mathsf{m}_i$ finding at least one correct PoW within these trials is given by $p_i = 1 - (1 - \varepsilon)^{n_i}$. Since $\varepsilon$ and $n_i$ are small, $p_i$ can be approximated to $p_i = 1 - (1 - \varepsilon)^{n_i} \approx n_i \varepsilon$.

Therefore, the set of trials of $\mathsf{m}_i$ within $\delta$ can be unified to constitute a single Bernoulli process with success probability $n_i \varepsilon$.

Assuming that there are $\ell$ miners, $\mathsf{m}_i$, $i = 1 \ldots \ell$ with success probability $p_i$, $i = 1 \ldots \ell$ respectively, the overall probability of success in block generation can be approximated to

$$\mathsf{pr} \approx 1 - \prod_{i=1}^{\ell}(1 - p_i), \text{ or } \mathsf{pr} = 1 - (1 - p)^{\ell} \approx \ell \cdot p.$$

This is true when $p\ell \ll 1$ and when the miners have equal computing power—that is, $p_i = p, i = 1 \ldots \ell$.

We divide time into equal intervals of size $\delta$; let $t_0 = 0$ denote the time when the last block was generated. Here, each miner can make up to $n_i$ trials for block generation within each interval. Let the random variable $X_k$ denote the event of success in the time interval between $t_k$ and $t_{k+1}$. That is,

$$X_k = \begin{cases} 1 & \text{if a block is created between } t_{k-1}, \text{ and } t_k, \\ 0 & \text{otherwise.} \end{cases}$$

It is evident that $\mathsf{Prob}(X_k = 1) = \mathsf{pr}$. Conceivably, after a success in block generation, miners stop mining for that particular block. We denote the number of attempts until a success is achieved by another random variable $\mathcal{Y}$:

$$\mathsf{Prob}(\mathcal{Y} = k) = \mathsf{Prob}(X_k = 1) \prod_{i=1}^{k-1} \mathsf{Prob}(X_i = 0) = \mathsf{pr}(1 - \mathsf{pr})^{k-1}.$$

Assuming a constant rate of trials per time window $\delta$, the number of failures until a success is observed in block generation is proportional to the time it takes for a block to be generated. Let $\mathcal{T}$ denote the time period until a block is generated:

$$\mathsf{Prob}(\mathcal{T} = k \cdot \delta) = \mathsf{Prob}(\mathcal{Y} = k) \ = \ \mathsf{pr}(1 - \mathsf{pr})^{k-1}.$$

Given this, we conclude that the distribution of block generation times can be modeled with a shifted geometric distribution with parameter $\mathsf{pr}$ [ProofWiki 2013].

In Figure 2, we confirm this analysis and show that (experimental) block generation times in Bitcoin can be fitted to a shifted geometric distribution with $p = 0.19$.[5] For the purpose of our experiments, we considered $\delta$ to be 2 minutes. To measure the generation time of existing Bitcoin blocks, we created a Python script that parses the block chain of Bitcoin (up to June 2013) and extracts the time intervals between the generation of consecutive blocks. Our findings show that although the average block generation time is approximately 10 minutes (10 minutes and 2.66 seconds), the standard deviation of the measurements is about 1241.3855 seconds, which corresponds to almost 20 minutes.

This also shows that the time required to confirm transactions impedes the operation of many businesses that are characterized by a fast-service time. As such, it is clear that vendors, such as vending machines and take-away stores [BitcoinMyths 2013], cannot rely on transaction confirmation when accepting Bitcoin payments. To address that, Bitcoin encourages vendors to accept fast Bitcoin payments with zero confirmations as soon as the vendor receives a transaction from the network transferring the correct amount of BTCs to one of its addresses [BitcoinFAQ 2013; BitcoinMyths 2013].

### 3.3. Necessary Conditions for Successful Double-Spending

To perform a successful double-spending attack, the attacker $\mathcal{A}$ needs to trick the vendor $\mathcal{V}$ into accepting a transaction $\mathrm{TR}_{\mathcal{V}}$ that $\mathcal{V}$ will not be able to redeem subsequently.

In this case, $\mathcal{A}$ creates another transaction $\mathrm{TR}_{\mathcal{A}}$ that has the same inputs as $\mathrm{TR}_{\mathcal{V}}$ (i.e., $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ use the same BTCs) but replaces the recipient address of $\mathrm{TR}_{\mathcal{V}}$—the address of $\mathcal{V}$—with a recipient address that is under the control of $\mathcal{A}$. If both transactions are sent at the same time, they are likely to have similar chances of getting confirmed in an upcoming block. This is the case since Bitcoin peers will not accept multiple transactions that share common inputs; they will only accept the version of the transaction reaching them first that they will consider for inclusion in their generated blocks and will ignore all subsequent transactions. Given this, a double-spending attack can succeed if $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{V}}$, and the majority of the peers in the network receive $\mathrm{TR}_{\mathcal{A}}$ so that $\mathrm{TR}_{\mathcal{A}}$ is more likely to be included in a subsequent block.

Let $t_i^{\mathcal{V}}$ and $t_i^{\mathcal{A}}$ denote the times at which node $i$ receives $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$, respectively. As such, $t_{\mathcal{V}}^{\mathcal{V}}$ and $t_{\mathcal{V}}^{\mathcal{A}}$ denote the respective times at which $\mathcal{V}$ receives $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$. Given this, we outline the necessary conditions for $\mathcal{A}$'s success in performing a double-spending attack.

*Requirement 1—$\mathrm{TR}_{\mathcal{V}}$ is added to the wallet of $\mathcal{V}$.* If $\mathrm{TR}_{\mathcal{V}}$ is not added to the memory pool of $\mathcal{V}$, then $\mathcal{V}$ cannot check that $\mathrm{TR}_{\mathcal{V}}$ was indeed broadcasted in the network. Note that for $\mathrm{TR}_{\mathcal{V}}$ to be included in $\mathcal{V}$'s wallet, then $t_{\mathcal{V}}^{\mathcal{V}} < t_{\mathcal{V}}^{\mathcal{A}}$; otherwise, $\mathcal{V}$ will first add $\mathrm{TR}_{\mathcal{A}}$ to its memory pool and will reject $\mathrm{TR}_{\mathcal{V}}$ as it arrives later.

*Requirement 2—$\mathrm{TR}_{\mathcal{A}}$ is confirmed in the block chain.* If $\mathrm{TR}_{\mathcal{V}}$ is confirmed first in the block chain, $\mathrm{TR}_{\mathcal{A}}$ can never appear in subsequent blocks. In other words, $\mathcal{V}$ will not

[5] We acquired $p = 0.19$ by fitting the average and variance of block generation times that we acquired experimentally from the block chain (up to June 2013) in a shifted geometric distribution [ProofWiki 2013].

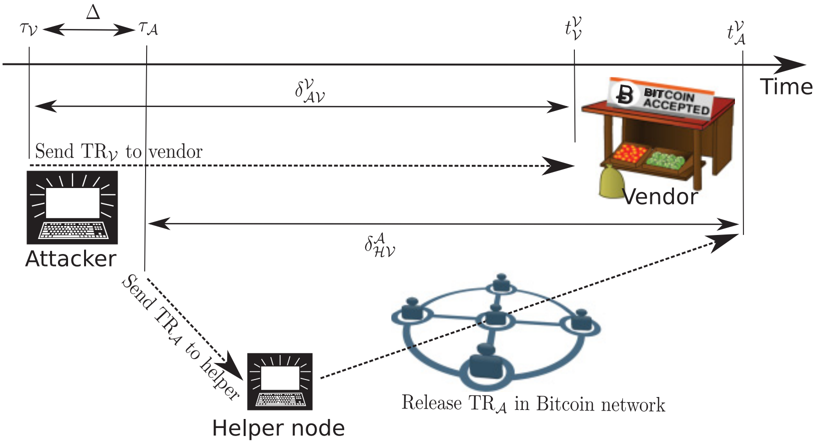

$\tau_{\mathcal{V}}$
$\Delta$
$\tau_{\mathcal{A}}$
$t_{\mathcal{V}}^{\mathcal{V}}$
$t_{\mathcal{A}}^{\mathcal{V}}$
Time
$\delta_{\mathcal{AV}}^{\mathcal{V}}$
Send TR$_{\mathcal{V}}$ to vendor
BITCOIN ACCEPTED
Vendor
Attacker
$\delta_{\mathcal{HV}}^{\mathcal{A}}$
Send TR$_{\mathcal{A}}$ to helper
Helper node
Release TR$_{\mathcal{A}}$ in Bitcoin network

Fig. 3. Sketch of a double-spending attack on fast payments in Bitcoin. Here, the attacker $\mathcal{A}$ dispatches two transactions that use the same BTCs in the Bitcoin network. The double-spending attack is successful if the BTCs that $\mathcal{A}$ used to pay for $\mathcal{V}$ cannot be redeemed (i.e., when the second transaction is included in the upcoming Bitcoin block).

have its BTCs back. Recall that the goal of $\mathcal{A}$ is to acquire a service offered by $\mathcal{V}$ without having to spend her BTCs.

*Requirement 3—$\mathcal{V}$'s service time is smaller than the time it takes $\mathcal{V}$ to detect misbehavior*. Since Bitcoin users are anonymous and users hold many accounts, there is only limited value in $\mathcal{V}$ detecting misbehavior after the user has obtained the service (and left the store). As such, for $\mathcal{V}$ to successfully detect any misbehavior by $\mathcal{A}$, the detection time must be smaller than the service time. In Section 4, we investigate the linkability of Bitcoin addresses in detail.

### 3.4. Performing Double-Spending Attacks in Bitcoin

In this section, we discuss how $\mathcal{A}$ can satisfy Requirements 1, 2, and 3. Table I summarizes the notations used in Section 3.

**Satisfying Requirement 1—TR$_{\mathcal{V}}$ is added to the wallet of $\mathcal{V}$.** In the sequel, we assume that $\mathcal{A}$ has access to a set of helper nodes, denoted by $\mathcal{H}$. $\mathcal{A}$ and $\mathcal{H}$ do not necessarily have to be on physically disjoint machines (e.g., $\mathcal{H}$ could run as a thread/process on the same machine as $\mathcal{A}$). We further assume that $\mathcal{H}$ never connects directly to $\mathcal{V}$ in the Bitcoin P2P network.

As shown in Figure 3, $\mathcal{A}$ sends TR$_{\mathcal{V}}$ to $\mathcal{V}$ at time $\tau_{\mathcal{V}}$ and TR$_{\mathcal{A}}$ to $\mathcal{H}$ at time $\tau_{\mathcal{A}}$ such that $\tau_{\mathcal{A}} = \tau_{\mathcal{V}} + \Delta$. $\mathcal{V}$ and $\mathcal{H}$ relay the transactions that they received from $\mathcal{A}$ in the network. Let $\delta_{\mathcal{HV}}^{\mathcal{A}}$ refer to the time it takes TR$_{\mathcal{A}}$ to propagate in the Bitcoin P2P network from $\mathcal{H}$ to $\mathcal{V}$ and $\delta_{\mathcal{AV}}^{\mathcal{V}}$ denote the time it takes TR$_{\mathcal{V}}$ to reach $\mathcal{V}$. In this case, $t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$ can be estimated as follows:

$$t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}} \approx \tau_{\mathcal{A}} + \delta_{\mathcal{HV}}^{\mathcal{A}} - \left(\tau_{\mathcal{V}} + \delta_{\mathcal{AV}}^{\mathcal{V}}\right) \approx \Delta + \delta_{\mathcal{VH}}^{\mathcal{A}} - \delta_{\mathcal{AV}}^{\mathcal{V}}. \qquad (2)$$

Note that since $\mathcal{H}$ is never a neighbor of $\mathcal{V}$, there is at least one hop on the path between $\mathcal{H}$ and $\mathcal{V}$. For simplicity, we assume that $\mathcal{A}$ connects directly to $\mathcal{V}$. We acknowledge that some vendors may not accept direct incoming connections [BitcoinWiki 2014] or may be located behind Network Address Translators (NATs). In Section 3.5, we show that our analysis can also apply in the case where $\mathcal{A}$ is not directly connected to $\mathcal{V}$.

Table I. Summary of Notations Used in Section 3

| Notation | Explanation |
|---|---|
| $\mathcal{A}$ | Attacker machine |
| $\mathcal{V}$ | Merchant machine |
| $\mathcal{H}$ | Helper nodes colluding with $\mathcal{A}$ |
| $\mathrm{TR}_{\mathrm{x}}$ | Transaction $x$ |
| $n_i$ | Number of attempts that a miner $\mathsf{m}_i$ performs to find a PoW |
| $p_i$ | Probability that $\mathsf{m}_i$ finds a PoW after $n_i$ trials |
| pr | Probability of success of all miners in finding a PoW |
| $\eta^k_{\mathcal{V}}$ | The number of Bitcoin peers that received (and mine for) $\mathrm{TR}_{\mathcal{V}}$ |
| $\eta^k_{\mathcal{A}}$ | The number of Bitcoin peers that received (and mine for) $\mathrm{TR}_{\mathcal{A}}$ |
| $\mathsf{p}_{\mathcal{V}}(k)$ | Probability that a block containing $\mathrm{TR}_{\mathcal{V}}$ is generated within the time interval $]t_k,\ t_{k+1}]$ |
| $\mathsf{p}_{\mathcal{A}}(k)$ | Probability that a block containing $\mathrm{TR}_{\mathcal{A}}$ is generated within the time interval $]t_k,\ t_{k+1}]$ |
| $\delta^{\mathcal{A}}_{\mathcal{H}\mathcal{V}}$ | Propagation delay of $\mathrm{TR}_{\mathcal{A}}$ to reach the merchant |
| $\delta^{\mathcal{V}}_{\mathcal{A}\mathcal{V}}$ | Propagation delay of $\mathrm{TR}_{\mathcal{V}}$ to reach the merchant |
| $t_{g_{\mathcal{A}}}$ | Time required by miners to generate a block containing $\mathrm{TR}_{\mathcal{A}}$ |
| $t_{g_{\mathcal{V}}}$ | Time required by miners to generate a block containing $\mathrm{TR}_{\mathcal{V}}$ |
| $\mathrm{TR}_{\mathcal{A}}$ | Transaction double-spending coins to $\mathcal{A}$'s addresses |
| $\mathrm{TR}_{\mathcal{V}}$ | Original transaction destined to merchant |
| $t^{\mathcal{V}}_i$ | Time at which node $i$ receives $\mathrm{TR}_{\mathcal{V}}$ |
| $t^{\mathcal{A}}_i$ | Time at which node $i$ receives $\mathrm{TR}_{\mathcal{V}}$ |
| $\Delta$ | Delay between the transmission of $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ |
| $\mathrm{P_S}$ | Probability that the double-spending attack succeeds |
| $\mathrm{P_D}$ | Probability that the merchant receives both $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ after waiting for 15 seconds |

Since $\mathcal{A}$ is an immediate neighbor of $\mathcal{V}$ and assuming no congestion at network paths, then $\delta^{\mathcal{A}}_{\mathcal{V}\mathcal{H}} > \delta^{\mathcal{V}}_{\mathcal{A}\mathcal{V}}$. In this case, $t^{\mathcal{V}}_{\mathcal{V}} < t^{\mathcal{A}}_{\mathcal{V}}$ for reasonably chosen $\Delta$ (e.g., $\Delta \geq 0$), thus satisfying Requirement 1.

**Satisfying Requirement 2—$\mathbf{TR}_{\mathcal{A}}$ is confirmed in the block chain.** Since $\mathcal{H}$ and $\mathcal{V}$ are highly likely to have different neighbors, the broadcasted transactions are likely to spread in the network until the point where either (1) all Bitcoin peers accept in their memory pools $\mathrm{TR}_{\mathcal{V}}$ or $\mathrm{TR}_{\mathcal{A}}$ or (2) $\mathrm{TR}_{\mathcal{V}}$ or $\mathrm{TR}_{\mathcal{A}}$ gets confirmed in a block.

In what follows, we estimate the probability that $\mathrm{TR}_{\mathcal{A}}$ is confirmed in a block first. In our analysis, we denote by $t_0$ the time at which both transactions $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ first coexist in the network,[6] and we assume that no block containing either one of them has been generated until that time. We argue that this is a realistic assumption given that $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ need to be typically broadcasted back to back given a small delay (in the order of few seconds); it is therefore unlikely that one of them is confirmed within the first few seconds in a new block. In the experiments in Section 3.5, we relax this assumption and evaluate the general case where either $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ can be confirmed immediately when they are broadcasted in the network.

We divide time into equal intervals of size $\delta$ such that the probability of successful block generation in each $\delta$ can be modeled as a Bernoulli trial with success probability $\eta \cdot p$, where $\eta$ is the number of peers that work toward block generation and $p$ is the success probability of a peer in generating a block within $\delta$.[7]

[6] This does not necessarily mean that $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ are broadcasted at the same time.

[7] The probability that at least a miner succeeds in block generation within $\delta$ is the complement of the probability that none of the peers who are working toward the block generation succeed and equals to $1 - (1 - p)^{\eta} \approx 1 - (1 - \eta \cdot p) = \eta \cdot p$, where $\eta \cdot p \ll 1$.

Let $t_k = k \cdot \delta + t_0$ and $\eta_{\mathcal{V}}^k$ and $\eta_{\mathcal{A}}^k$ denote the number of Bitcoin peers that received (and mine for) $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$, respectively, until time $t_k$. Each Bitcoin node will only add to its memory pool the transaction that it receives first among $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$. Since only the transactions that appear in the memory pool of peers are eligible to be confirmed in subsequent blocks, the probability that $\mathrm{TR}_{\mathcal{V}}$ is included in a block within time interval $]t_k,\ t_{k+1}]$ is $\mathrm{Pr}_{\mathcal{V}}^k = \eta_{\mathcal{V}}^k \cdot p$.[8] Similarly, for $\mathrm{TR}_{\mathcal{A}}$, the corresponding probability is $\mathrm{Pr}_{\mathcal{A}}^k = \eta_{\mathcal{A}}^k \cdot p$.[9] Thus, the probability $\mathsf{p}_{\mathcal{V}}(k)$ that a block containing $\mathrm{TR}_{\mathcal{V}}$ is generated within the time interval $]t_k,\ t_{k+1}]$ is

$$\mathsf{p}_{\mathcal{V}}(k) = \mathrm{Pr}_{\mathcal{V}}^k \cdot \prod_{i=0}^{k-1} \left(1 - \mathrm{Pr}_{\mathcal{V}}^i\right) = \eta_{\mathcal{V}}^k p \cdot \prod_{i=0}^{k-1} \left(1 - \eta_{\mathcal{V}}^i p\right).$$

Similarly, the probability that a block containing $\mathrm{TR}_{\mathcal{A}}$ is generated at the same time interval is given by

$$\mathsf{p}_{\mathcal{A}}(k) = \mathrm{Pr}_{\mathcal{A}}^k \cdot \prod_{i=0}^{k-1} \left(1 - \mathrm{Pr}_{\mathcal{A}}^i\right) = \eta_{\mathcal{A}}^k p \cdot \prod_{i=0}^{k-1} \left(1 - \eta_{\mathcal{A}}^i p\right).$$

If at time $t_s = s \cdot \delta + t_0$ every node in the network has received at least one of the transactions $\mathrm{TR}_{\mathcal{V}}$ or $\mathrm{TR}_{\mathcal{A}}$, the following holds:

$$\eta_{\mathcal{A}}^k \ \leq\ \eta_{\mathcal{A}}^{k+1} \text{ and } \eta_{\mathcal{V}}^k \leq \eta_{\mathcal{V}}^{k+1}, \text{ if } k < s$$

$$\eta_{\mathcal{A}}^k = \eta_{\mathcal{A}}^{k+1} \ =\ \eta_{\mathcal{A}}^s \text{ and } \eta_{\mathcal{V}}^k = \eta_{\mathcal{V}}^{k+1} = \eta_{\mathcal{V}}^s, \text{ otherwise.}$$

This suggests that $\forall i \geq s,\ \eta_{\mathcal{V}}^i + \eta_{\mathcal{A}}^i = \eta_{\mathcal{V}}^s + \eta_{\mathcal{A}}^s$. To compute the probability of success of the double-spending attack, we make the assumption that $\forall k$, $\eta_{\mathcal{V}}^k$, and $\eta_{\mathcal{A}}^k$ do not exchange their newly constructed blocks; in this way, the time $t_{g_{\mathcal{V}}}$ required by peers that are mining in favor of $\mathrm{TR}_{\mathcal{V}}$ to generate a new block is independent of that required by the peers that are mining in favor of $\mathrm{TR}_{\mathcal{A}}$, $t_{g_{\mathcal{A}}}$. Given this, the probability that Requirement 2 is satisfied, $\mathrm{P_S}^{(2)}$, is $\mathrm{P_S}^{(2)} = \mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) + \frac{1}{2}\mathrm{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}})$.

In other words, $\mathrm{P_S}^{(2)}$ is composed of two components: one corresponds to the event that the block containing $\mathrm{TR}_{\mathcal{A}}$ is first generated and the second to the event where the blocks containing $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ are generated at the same time (i.e., $t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}$). In the latter case, the probability that the block containing $\mathrm{TR}_{\mathcal{A}}$ is eventually adopted by the Bitcoin peers is 0.5. Here, $\mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}})$ and $\mathrm{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}})$ are computed as follows:

$$\mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) \ =\ \sum_{g_{\mathcal{A}}=0}^{\infty} \mathsf{p}_{\mathcal{A}}(g_{\mathcal{A}}) \cdot \mathsf{p}_{\mathcal{V}}(g_{\mathcal{V}} > g_{\mathcal{A}} | g_{\mathcal{A}}) \tag{3}$$

$$= \eta_{\mathcal{A}}^0 p\left(1 - \eta_{\mathcal{V}}^0 p\right) + \sum_{g_{\mathcal{A}}=0}^{\infty} \eta_{\mathcal{A}}^{g_{\mathcal{A}}} p \cdot \left(1 - \eta_{\mathcal{V}}^{g_{\mathcal{A}}} p\right) \cdot \prod_{j=0}^{g_{\mathcal{A}}-1} \left(1 - \eta_{\mathcal{V}}^j p\right)\left(1 - \eta_{\mathcal{A}}^j p\right). \tag{4}$$

$$\mathrm{Prob}(t_{g_{\mathcal{A}}} \ =\ t_{g_{\mathcal{V}}}) = \sum_{g_{\mathcal{A}}=1}^{\infty} p^2 \eta_{\mathcal{V}}^{g_{\mathcal{A}}} \eta_{\mathcal{A}}^{g_{\mathcal{A}}} \cdot \prod_{j=0}^{g_{\mathcal{A}}-1} \left(1 - \eta_{\mathcal{V}}^j p\right)\left(1 - \eta_{\mathcal{A}}^j p\right).$$

[8] This stems from the fact that the probability that at least one out of the $\eta_{\mathcal{V}}^k$ nodes succeeding in block generation in the time interval between $t_k$ and $t_{k+1}$ (each with success probability $p$ in this time interval) is given by $\mathrm{Pr}_{\mathcal{V}}^k = 1 - (1-p)^{\eta_{\mathcal{V}}^k}$). Since $\eta_{\mathcal{V}}^k \cdot p \ll 1$, the previous equation can be written as $\mathrm{Pr}_{\mathcal{V}}^k = \eta_{\mathcal{V}}^k \cdot p$.

[9] Note that in this case, both $\eta_{\mathcal{V}}^k \cdot p \ll 1$ and $\eta_{\mathcal{A}}^k \cdot p \ll 1$; here, we assume that the time interval between $t_k$ and $t_{k+1}$ is short enough to satisfy these constraints.

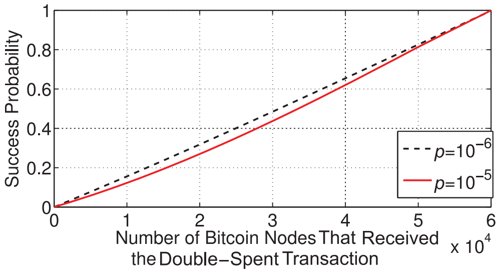


Fig. 4. $\mathrm{Ps}^{(2)}$ with respect to various values of $\eta^k_{\mathcal{A}}$ and $\eta^k_{\mathcal{V}}$. Here, $p = 10^{-6}$, $\delta = 10$ seconds, $t_0 = 0$, $t_s = \delta$, and the number of peers in the network is 60,000.

For the purpose of this analysis, we assume that $t_s = \delta$ and that $\delta = 10$ seconds. We can therefore rewrite $\mathrm{Ps}^{(2)}$ as follows:

$$
\begin{aligned}
\mathrm{Ps}^{(2)} &= \mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) + \frac{1}{2} Prob(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}).\\
\mathrm{Prob}(t_{g_{\mathcal{A}}} < t_{g_{\mathcal{V}}}) &= \eta^0_{\mathcal{A}} p(1 - \eta^0_{\mathcal{V}} p) + \eta^1_{\mathcal{A}} p(1 - \eta^0_{\mathcal{A}} p)(1 - \eta^0_{\mathcal{V}} p)(1 - \eta^1_{\mathcal{V}} p) \sum_{g_{\mathcal{A}}=2}^{\infty} \eta^1_{\mathcal{A}} p(1 - \eta^0_{\mathcal{A}} p)\\
&\quad \times (1 - \eta^1_{\mathcal{A}} p)^{(g_{\mathcal{A}}-2)} \cdot (1 - \eta^0_{\mathcal{V}} p)(1 - \eta^1_{\mathcal{V}} p)^{(g_{\mathcal{A}}-1)}\\
\mathrm{Prob}(t_{g_{\mathcal{A}}} = t_{g_{\mathcal{V}}}) &= \eta^0_{\mathcal{V}} \eta^0_{\mathcal{A}} p^2 + \sum_{g_{\mathcal{A}}=1}^{\infty} \eta^1_{\mathcal{V}} \eta^1_{\mathcal{A}} p^2 (1 - \eta^0_{\mathcal{V}} p)\\
&\quad \cdot (1 - \eta^0_{\mathcal{A}} p)\left[(1 - \eta^1_{\mathcal{V}} p) \cdot (1 - \eta^1_{\mathcal{A}} p)\right]^{(g_{\mathcal{A}}-1)}. \qquad (5)
\end{aligned}
$$

In Figure 4, we depict $\mathrm{Ps}^{(2)}$ for various values of $\eta^k_{\mathcal{V}}$, $\eta^k_{\mathcal{A}}$, and $p$ when $\delta = 10$ seconds, $t_s = \delta$, and the number of peers in the network is 60,000. Our analysis therefore shows that $\mathcal{A}$ can maximize $\mathrm{Ps}^{(2)}$ by increasing the number of peers that receive $\mathrm{TR}_{\mathcal{A}}$, $\eta^k_{\mathcal{A}}$, $\forall t_k$. $\mathcal{A}$ can achieve this (1) by sending $\mathrm{TR}_{\mathcal{A}}$ before $\mathrm{TR}_{\mathcal{V}}$ and therefore giving $\mathrm{TR}_{\mathcal{A}}$ a better advantage in spreading in the network and/or (2) by relying on multiple helpers to spread $\mathrm{TR}_{\mathcal{A}}$ faster in the network. In the former case, $\mathcal{A}$ can delay the transmission of $\mathrm{TR}_{\mathcal{V}}$ by a maximum of $\Delta = \delta^{\mathcal{A}}_{\mathcal{V}\mathcal{H}} - \delta^{\mathcal{V}}_{\mathcal{A}\mathcal{V}}$ (cf. Equation (2)) after sending $\mathrm{TR}_{\mathcal{A}}$ while ensuring that $\mathcal{V}$ first receives $\mathrm{TR}_{\mathcal{V}}$. In this way, Requirements 1 and 2 can be satisfied.

**Satisfying Requirement 3—$\mathcal{V}$'s service time is smaller than the time it takes $\mathcal{V}$ to detect misbehavior.** As advocated in BitcoinMyths [2013], one possible way for $\mathcal{V}$ to detect double-spending attempts is to adopt a "listening period," of a few seconds, before delivering its service to $\mathcal{A}$; during this period, $\mathcal{V}$ monitors all of the transactions that it receives and checks if any of them attempt to double-spend the coins that $\mathcal{V}$ previously received from $\mathcal{A}$. Note that $\mathcal{V}$ can also rely on additional nodes that it controls within the Bitcoin network—"observers"—that would directly relay to $\mathcal{V}$ all of the transactions that they receive.

These techniques are based on the intuition that since it takes every transaction a few seconds[10] to propagate to every node in the Bitcoin network, then it is highly likely that $\mathcal{V}$ or its observers would receive both $\mathrm{TR}_{\mathcal{V}}$ and $\mathrm{TR}_{\mathcal{A}}$ within the listening period (and before granting service to $\mathcal{A}$).

[10] Our experiments in Section 3.5 show that the average time for a peer to receive both $\mathrm{TR}_{\mathcal{A}}$ and $\mathrm{TR}_{\mathcal{V}}$ is approximately 3.354 seconds if both transactions were sent concurrently.

This detection technique can be circumvented by $\mathcal{A}$ as follows. $\mathcal{A}$ can attempt to delay the transmission of $\text{TR}_\mathcal{A}$ such that $t = (t_\mathcal{V}^\mathcal{A} - t_\mathcal{V}^\mathcal{V})$ exceeds the listening period (Requirement 3) while $\text{TR}_\mathcal{A}$ still has a significant chance of being spread in the network. On one hand, as $t$ increases, the probability that all immediate neighbors of $\mathcal{V}$ in the Bitcoin P2P network receive $\text{TR}_\mathcal{V}$ first also increases; when they receive $\text{TR}_\mathcal{A}$ later on, $\text{TR}_\mathcal{A}$ will not be added to the memory pool of $\mathcal{V}$'s neighbors and as such $\text{TR}_\mathcal{A}$ will not be forwarded to $\mathcal{V}$. On the other hand, $\mathcal{A}$ should make sure that $\text{TR}_\mathcal{A}$ was received by enough peers so that Requirement 2 can be satisfied. To that end, $\mathcal{A}$ can increase the number of helpers that it controls.

It is interesting to note that a Bitcoin node located at http://blockchain.info/ keeps track of all transactions exchanged in the system and attempts to identify double-spending transactions [BlockChain 2013]). However, this information is not propagated to peers in the network.

### 3.5. Experimental Evaluation

We now present the experimental results of double-spending experiments in the Bitcoin network. Our experiments aim at investigating the satisfiability of the aforementioned Requirements 1, 2, and 3.

*Experimental setup*. We adopt the setup described in Section 3.4 in which the attacker $\mathcal{A}$ is equipped with one or more helper nodes $\mathcal{H}$ that help her relay the double-spent transaction. In our experiments, we made use of 10 Bitcoin nodes located around the globe; this serves to better assess the different views seen from multiple points in the Bitcoin overlay network and to remove any bias that might originate from specific network topologies.

To perform the attack, we modified the C++ implementation of Bitcoin client version 0.5.2. Conforming with our analysis in Section 3.4, our new client does the following:

—The attacker connects to the vendor's machine. Here, we assume that $\mathcal{V}$ accepts direct connections if it has fewer than 125 connections.[11] If the connection is refused, $\mathcal{A}$ can wait until a neighbor of $\mathcal{V}$ disconnects before attempting to connect again.

—The attacker creates transactions $\text{TR}_\mathcal{V}$ and $\text{TR}_\mathcal{A}$ spending the same coins. She sends $\text{TR}_\mathcal{V}$ using the Bitcoin network to the neighboring vendor and $\text{TR}_\mathcal{A}$ to one or more helper nodes with an initial delay $\Delta$ of –1, 0, 1, and 2 seconds. Here, $\Delta$ refers to the time delay between the transmission of $\text{TR}_\mathcal{V}$ and $\text{TR}_\mathcal{A}$ by $\mathcal{A}$.

—Upon reception of $\text{TR}_\mathcal{A}$, each helper node broadcasts it in the Bitcoin network.

With this setup, we performed double-spending attempts when the vendors are located at four different network locations (two vendors were in North America, and the remaining two were in Asia Pacific). In our experiments, $\mathcal{A}$ was located in Europe. However, since $\mathcal{A}$ does not contribute in spreading any transaction herself, her location does not affect the outcome of the attack. In other words, the sole role of $\mathcal{A}$ is to send $\text{TR}_\mathcal{V}$ to $\mathcal{V}$ using a direct connection in the Bitcoin network.

We conducted our experiments with a varying number of connections of the vendor (8, 40, and 125 connections) and by varying the number of helper nodes (1 and 2). The helper nodes were connected to 125 other Bitcoin peers. Each data point in our measurements corresponds to 10 different measurements, totaling approximately 500 double-spending attempts. We also created a Python script that, for each

[11] Note that the maximum number of connections can be modified using the "-maxconnections" command [Connectivity 2013].

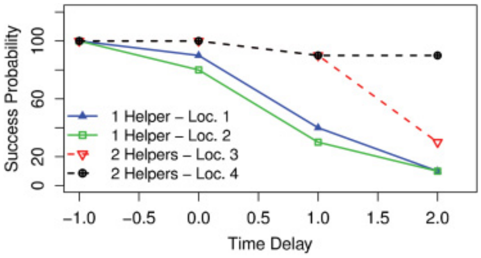


Fig. 5. $P_S$ versus $\Delta$ when $\mathcal{V}$ has 40 connections. Here, the vendors are located at four different network locations (Locations 1 and 2 are in North America, and Locations 3 and 4 are in Asia Pacific).

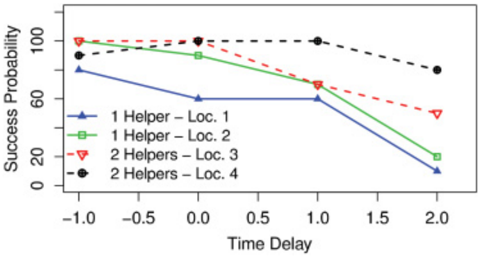


Fig. 6. $P_S$ versus $\Delta$ when $\mathcal{V}$ has 125 connections. Here, the vendors are located at four different network locations (Locations 1 and 2 are in North America, and Locations 3 and 4 are in Asia Pacific).

| **Location** | **# Helpers** | **$\Delta$(sec)** | **$P_S$(%)** |
|---|---|---|---|
| Asia Pacific 1, 125 connections | 2 | 0 | 100 |
| Asia Pacific 2, 125 connections | 2 | 0 | 100 |
| North America 1, 8 connections | 1 | 0 | 100 |
| North America 2, 40 connections | 1 | 0 | 90 |
| Asia Pacific 1, 8 connections | 2 | 1 | 100 |
| Asia Pacific 2, 125 connections | 2 | 1 | 100 |
| North America 1, 40 connections | 1 | -1 | 100 |

Fig. 7. Summary of results. Here, "Location" denotes the location of $\mathcal{V}$, and "connections" denote the number of $\mathcal{V}$'s connections.

conducted measurement, parses the generated logs along with the Bitcoin block explorer [BitcoinExplorer 2013] to check whether Requirement 2 is satisfied.

*Satisfying Requirements 1 and 2*. To assess the feasibility of double-spending in fast Bitcoin payments, we evaluate empirically the success probability with respect to the number of helper nodes, the number of connections of the vendor, and $\Delta$.

Our experimental results, depicted in Figures 5, 6, and 7 show that irrespective of a specific network topology, the probability that $\mathcal{A}$ succeeds in performing double-spending attacks is significant. Confirming our previous analysis, $P_S$ decreases as $\Delta$ increases. As explained in Section 3.4, this is because the higher is $\Delta$, the larger is the number of peers that receive $TR_{\mathcal{V}}$; in turn, the probability that $TR_{\mathcal{A}}$ is confirmed before $TR_{\mathcal{V}}$ decreases. As shown in Figures 5 and 6, this can be remedied if the number of helper nodes that spread $TR_{\mathcal{A}}$ increases. Our results show that even for a large $\Delta$ of 2 seconds, relying on two helper nodes still guarantees that double-spending succeeds with a considerable probability; when $\Delta = 1$ seconds, the attack is guaranteed to succeed ($P_S$ is close to 1) using two helpers. This is summarized in Figure 7. Generalizing these results, it is clear that $\mathcal{A}$ succeeds, with high probability, in spending the same coin to $n \geq 1$ different recipients as long as the number of helpers that assist $\mathcal{A}$ in spreading $TR_{\mathcal{A}}$ is greater or equal to $n$. As we show in the previous section, the larger is $n$, the higher is the probability that $\mathcal{A}$'s misbehavior is detected.

The number of $\mathcal{V}$'s connections considerably affects $P_S$ especially when $\mathcal{A}$ controls only one helper; in the case where $\mathcal{V}$ has a similar number of connections when compared to the number of connections of the helper, $P_S$ approaches 0.5. This corresponds to the case where $TR_{\mathcal{A}}$ and $TR_{\mathcal{V}}$ are spread equally in the network (Figure 6). On the other hand, as the connectivity of $\mathcal{V}$ decreases, $TR_{\mathcal{A}}$ spreads faster in the network (cf. Figure 5).

Table II. Example of Triplets ($\Delta$, $N_{\mathcal{H}}$, $C$) Where $P_S > 0$, $P_D = 0$ and $t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}} = \infty$

| | $P_S$(%) | $P_D$(%) | $t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$ (sec) | Observed (%) |
|---|---|---|---|---|
| South America, 8 Connections, 3 Helpers, $\Delta = 2.5$ | 7.7 | 0 | $\infty$ | 53 |
| South America, 8 Connections, 4 Helpers, $\Delta = 3.0$ | 13.33 | 0 | $\infty$ | 57 |
| Asia Pacific, 8 Connections, 3 Helpers, $\Delta = 2.75$ | 10 | 0 | $\infty$ | 57 |
| Asia Pacific, 8 Connections, 3 Helpers, $\Delta = 2.75$ | 5 | 0* | $\infty$ | 66 |
| North America, 20 Connections, 3 Helpers, $\Delta = 2.75$ | 5 | 0 | $\infty$ | 47 |
| Asia Pacific, 60 Connections, 1 Helper, $\Delta = 3.00$ | 10 | 0* | $\infty$ | 20 |

*In these cases, $\mathcal{V}$ never receives $TR_{\mathcal{A}}$ and as such cannot detect double-spending attacks, even if it adopts a very large listening period.

Table III. Example of Triplets ($\Delta$, $N_{\mathcal{H}}$, $C$) Where $P_S \geq P_D$

| | $P_S$(%) | $P_D$(%) | $t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}}$ (sec) | Observed (%) |
|---|---|---|---|---|
| Europe, 8 Connections, 3 Helpers, $\Delta = 2.00$ | 10 | 10 | 8.664 | 53 |
| Europe, 8 Connections, 3 Helpers, $\Delta = 2.25$ | 10 | 10* | 5.65 | 47 |
| South America, 8 Connections, 2 Helpers, $\Delta = 2.5$ | 20 | 6.66* | 3.749 | 62 |
| Asia Pacific, 8 Connections, 2 Helpers, $\Delta = 1.75$ | 55 | 20* | 5.5 | 91 |
| North America, 20 Connections, 5 Helpers, $\Delta = 3.00$ | 11 | 11 | 3.208 | 46 |
| North America, 20 Connections, 1 Helper, $\Delta = 1.25$ | 30 | 30* | 3.34 | 78 |
| North America, 20 Connections, 4 Helpers, $\Delta = 2.00$ | 82 | 63 | 2.85 | 78 |
| North America, 20 Connections, 2 Helpers, $\Delta = 2.00$ | 20 | 20* | 4.79 | 60 |
| North America, 20 Connections, 1 Helper, $\Delta = 1.50$ | 40 | 30* | 3.51 | 60 |
| Europe, 20 Connections, 3 Helpers, $\Delta = 1.0$ | 45 | 45* | 3.844 | 87 |
| Europe, 30 Connections, 1 Helper, $\Delta = 1.5$ | 15 | 10* | 3.412 | 42 |
| Asia Pacific, 40 Connections, 1 Helper, $\Delta = 2.9$ | 10 | 10* | 4.946 | 42 |
| Europe, 40 Connections, 1 Helper, $\Delta = 1.25$ | 10 | 10 | 1.841 | 36 |
| Europe, 40 Connections, 2 Helpers, $\Delta = 1.5$ | 20 | 20%* | 3.075 | 36 |
| South America, 40 Connections, 1 Helper, $\Delta = 2.0$ | 30 | 40 | 3.217 | 57 |
| Asia Pacific, 80 Connections, 1 Helper, $\Delta = 3.7$ | 10 | 20 | 5.04 | 18 |
| Europe, 80 Connections, 1 Helper, $\Delta = 2.75$ | 13.33 | 26.67 | 5.093 | 28 |
| Asia Pacific, 100 Connections, 1 Helper, $\Delta = 1.5$ | 80 | 80 | 2.807 | 88 |

*In these cases, $\mathcal{V}$ can detect double-spending attempts with some probability by adopting a listening period, but since $P_S \geq P_D$, then a number of $\mathcal{A}$'s double-spending attempts will not be detected, which gives her incentives to perform double-spending attempts.

*Satisfying Requirement 3*. In our experiments, we were looking for triplets ($\Delta$, $N_{\mathcal{H}}$, $C$), where $N_{\mathcal{H}}$ is the number of helper nodes and $C$ is the number of $\mathcal{V}$' connections, that minimize the probability $P_D$ that $\mathcal{V}$ receives $TR_{\mathcal{A}}$.

In Table II, we include a number of triplets ($\Delta$, $N_{\mathcal{H}}$, $C$) for which $P_S > 0$ and $P_D = 0$ (i.e., $t = t_{\mathcal{V}}^{\mathcal{A}} - t_{\mathcal{V}}^{\mathcal{V}} = \infty$). These are instances of the case where *all* of the neighbors of $\mathcal{V}$ receive $TR_{\mathcal{V}}$ first and do not forward $TR_{\mathcal{A}}$. We point out that in this case, although $P_S$ is modest, $\mathcal{A}$ has considerable incentives in performing double-spending attacks since the probability that $\mathcal{V}$ detects her misbehavior is zero. In Table III, we show other instances of ($\Delta$, $N_{\mathcal{H}}$, $C$) for which $P_S \geq P_D$. Here, although $P_D > 0$, $\mathcal{A}$ still has an advantage in performing double-spending attacks, as these attacks are more likely to succeed.

Our findings therefore show that even if $\mathcal{V}$ adopts a listening period of few tens of seconds, double-spending is still possible. Our experiments also show that the triplets ($\Delta$, $N_{\mathcal{H}}$, $C$) resulting in $P_S \geq P_D$ do not depend on the location of $\mathcal{H}$ nor $\mathcal{V}$ nor on the time of the measurements; as shown in Table III, the same triplets can be used repeatedly by $\mathcal{A}$ to perform attacks at various times and in different network topologies.

In addition, we evaluated this technique using up to five observers. Our findings in Tables II and III show that this method can help in detecting double-spending, as all double-spent transactions were received by at least one observer within a few seconds. However, given that $\mathcal{A}$ delays the transmission of $\text{TR}_{\mathcal{A}}$, our results show that only a subset of the observers receive $\text{TR}_{\mathcal{A}}$. As mentioned previously, this corresponds to the case where all neighbors of these observers have received $\text{TR}_{\mathcal{V}}$ first, and as such they will not forward $\text{TR}_{\mathcal{A}}$ back to the observers. Therefore, $\mathcal{V}$ needs to employ a considerable number of observers ($\approx$3) (that connect to a large number of Bitcoin peers) to ensure that at least one observer detects any double-spending attempt; this, however, comes at the expense of additional costs for $\mathcal{V}$ to maintain the observers in the network.

Clearly, in the general case where $\mathcal{A}$ attempts to $n$-times spend the same coins, the larger is $n$, the bigger is the probability that this misbehavior is detected by fewer observers in the network. Our results show that double-spending attacks can be successfully mounted even when $\Delta > 1$ second. This also suggests that $\mathcal{A}$ does not have to be directly connected to $\mathcal{V}$ and may release $\text{TR}_{\mathcal{V}}$ in the Bitcoin network $\Delta$ seconds before $\mathcal{H}$ releases $\text{TR}_{\mathcal{A}}$. In other words, when $\Delta$ is not small (e.g., $\Delta > 1$), this also means that (1) the merchant's transaction will exhibit a comparable spread in the network irrespective of whether $\mathcal{A}$ is directly connected to $\mathcal{V}$ or not,[12] and (2) the probability that the merchant receives $\text{TR}_{\mathcal{V}}$ before receiving $\text{TR}_{\mathcal{A}}$ is high. As shown in our experiments, $\mathcal{A}$ can maximize its probability of success by relying on multiple helpers when $\Delta$ is large.

### 3.6. Abusing Forks in Bitcoin

Our analysis in Sections 3.4 and 3.5 shows that double-spending fast transactions is feasible during the *normal* operation of the Bitcoin system. In what follows, we discuss double-spending attacks in the special case where Bitcoin is subject to block chain forks [Decker and Wattenhofer 2013].

*Block forks*. During the normal Bitcoin operation, miners work on extending the longest block chain in the network. If miners do not share the same view in the network (e.g., due to network partitioning), they might work on different block chains, thus resulting in "forks" in the block chain. As an example, in March 2013, due to a difference in how Bitcoin versions 0.7 and 0.8 handled the block chain database, a serious block chain fork occurred (cf. Figure 8). The fork started at block-height 225,430, and at block-height 225,451 the 0.8 fork exceeded the 0.7 fork by 13 blocks. The Bitcoin developers, however, decided to support the smaller chain supported by the Bitcoin version 0.7. During block forks, the adversary bears little risk in performing double-spending attacks. Indeed, under such settings, the adversary can try to include $\text{TR}_{\mathcal{V}}$ in one chain and $\text{TR}_{\mathcal{A}}$ in another [Finney 2013].

*Double-spending using forks*. In what follows, we present an exemplary double-spending attack, which we tested in Bitcoin, that takes advantage of block forks. Our attack leverages an exploit in Bitcoin that arises from the simultaneous adoption of client versions 0.8.1 and 0.8.2 (or beyond) in the network. Starting from version 0.8.2, Bitcoin clients no longer accept transactions that do not follow a given signature encoding. As we show, this incompatibility with prior client versions can potentially lead to a double-spending attack in a fast payment setting in Bitcoin. The attack can only work when $\mathcal{V}$ operates on any client version prior to 0.8.2.

[12] Indeed, since Bitcoin users will immediately broadcast transactions where they appear as senders or recipients in the network, the merchant's transaction will be almost directly broadcasted in the network after it is received.

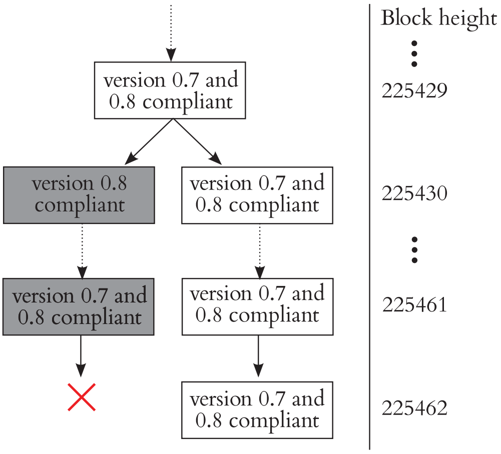

Block height
225429
225430
225461
225462
version 0.7 and 0.8 compliant
version 0.8 compliant
version 0.7 and 0.8 compliant
version 0.7 and 0.8 compliant
version 0.7 and 0.8 compliant
version 0.7 and 0.8 compliant

Fig. 8. Bitcoin chain fork in March 2013 due to concurrent adoption of client versions 0.7 and 0.8.

Up to version 0.8.1, a transaction signature could contain zero-padded bytes and the signature check would still be valid. However, starting from version 0.8.2, transactions with padding will no longer be accepted to the memory pool of nodes nor will they be relayed to other nodes.[13] This gives a considerable advantage for $\mathcal{A}$ to mount a double-spending attack as follows:

(1) $\mathcal{A}$ sends a transaction $TR_{\mathcal{V}}$ with a zero-padded signature to $\mathcal{V}$.
(2) $TR_{\mathcal{V}}$ will be relayed to the miners. Miners that use any Bitcoin version newer than 0.8.1 will not accept the transaction in their memory pool and thus will not include it into a block. Miners with an older Bitcoin version will accept it.
(3) $\mathcal{A}$ waits for a small time $t$ (e.g., 1 to 5 minutes), until she acquired service from the merchant.
(4) Then, provided that $TR_{\mathcal{V}}$ was still not included in a Bitcoin block, $\mathcal{A}$ sends another transaction $TR_{\mathcal{A}}$ that double-spends the inputs of $TR_{\mathcal{V}}$ to the benefit of a new Bitcoin address that is controlled by $\mathcal{A}$. $TR_{\mathcal{A}}$ is not padded with additional zeros.
(5) If most peers in the network use newer client versions than version 0.8.1, they will accept $TR_{\mathcal{A}}$ (and will reject $TR_{\mathcal{V}}$). The higher the fraction of peers that use version 0.8.2 (or beyond), the larger the likelihood that $TR_{\mathcal{A}}$ is included in a block and that the attack succeeds.

We implemented this double-spending attack in a private "test" Bitcoin network comprised of two Bitcoin miners, a merchant, and the adversary's machine. In our setup, the merchant was running client version 0.8.1, and the miners were running version 0.8.2. Our results show that such a double-spending attack succeeds with 100% probability in the investigated setting.

We therefore hope that our findings increase the awareness within the Bitcoin community on the delicacy of version releases. Indeed, although block forks might "naturally" occur from time to time in the network, such forks are unlikely to last for more than few blocks, as the network views tend to naturally converge on the longest block chain within few blocks. We argue that new version releases, on the other hand, can cause more serious damages, as they might result in long-lasting block forks that can only be stopped by manual intervention. Version releases should therefore be

[13] This applies to all Bitcoin versions starting from version 0.8.2 until the time of this writing (i.e., version 0.8.5).

carefully designed for backward compatibility; otherwise, the Bitcoin system might witness severe misbehavior.

### 3.7. Countermeasure: Forwarding Double-Spending Attempts in the Network

To efficiently detect double-spending on fast Bitcoin payments, we propose that Bitcoin peers forward transactions that attempt to double-spend the same coins in the Bitcoin network. Namely, our technique unfolds as follows. Whenever a peer receives a new transaction, it checks whether the transaction uses coins that have not been spent in any other transaction that resides in the block chain and in their memory pool. If so, then peers follow the current protocol of Bitcoin; peers add the transaction to their memory pool and forward it in the network. If, on the other hand, peers detect that there is another transaction in their memory pool that spends the same coins with different recipients, then peers forward the transaction to their neighbors (without adding the transaction to their memory pools).

The main intuition behind this technique is that although $\mathcal{A}$ might be able to prevent $\mathcal{V}$ and a subset of $\mathcal{V}$'s observers from receiving $\text{TR}_{\mathcal{A}}$, a considerable number of Bitcoin peers receive both $\text{TR}_{\mathcal{A}}$ and $\text{TR}_{\mathcal{V}}$. If the majority of these peers are honest,[14] both transactions would eventually reach $\mathcal{V}$ within a few seconds. The double-spending of $\mathcal{A}$ can be therefore detected before $\mathcal{A}$ actually receives the service from $\mathcal{V}$. We emphasize that our proposed technique does not change the spread of each transaction within the memory pools of Bitcoin peers (as such, it does not affect the success probability of the attack). Instead, this technique ensures that both transactions are received within a few seconds by $\mathcal{V}$ and that any possible double-spending attempt is detected almost immediately. This intuition is based on our previous measurements: our experiments in Section 3.4 show that the average time for a transaction to be received by the vendor is approximately 3.354 seconds after the transaction has been released in the Bitcoin network.

We implemented this technique and integrated it with the official Bitcoin client. Our modified Bitcoin client also keeps track of the number of established connections to warn the user when this number drops below a threshold value (80 in our case) and uses our proposed detection technique to (visually) alert the user when a double-spending attempt was detected in any of its transactions. We have evaluated the performance of our modified client by integrating it in the Bitcoin network for a period of 7 consecutive days. During the evaluation period, our modified client was able to forward all double-spent attempts (that we manually injected in the network) with a detection rate of 100%.

We acknowledge that this detection technique can result in the increase of the number of transactions circulating in the Bitcoin network and could be (ab)used to affect the performance of the network (e.g., to conduct Denial of Service attacks [Bitcoin-DoS 2013]). We argue, however, that peers can only forward the first double-spending transaction attempt in the network and drop all subsequent double-spending of the same coin. This variant ensures that all peers in the network can identify (and verify the misbehaving address) and refuse to receive any subsequent payment/transaction from this address. This variant detection technique is integrated in BitcoinXT [2014]; a number of Bitcoin nodes already use this technique to report double-spending attempts [DoubleSpending 2014].

## 4. EVALUATING USER PRIVACY AND ACCOUNTABILITY IN BITCOIN

Our findings in Section 3 suggest that misbehavior in Bitcoin is inevitable and is only expected to increase as the utility of the system increases. Currently, Bitcoin nodes

[14] This is the underlying assumption that ensures the correct operation of Bitcoin.

locally ban the IP address of the misbehaving user for 24 hours. Clearly, such an approach is not sufficient to deter misbehavior, as malicious peers can, for example, modify/spoof their IPs or even try to connect to and attack other peers, who still have not blacklisted their IP address.

We argue that if Bitcoin is to sustain another decade of service, then it must incorporate accountability measures to ensure that a misbehaving user is indeed "punished." In this respect, one possible solution would be to enforce Bitcoin *address* blacklisting. Here, the idea would be that Bitcoin addresses that have been found to misbehave (e.g., double-spend) are added to a public blacklist. Ideally, the BTCs of the blacklisted addresses will not be accepted by Bitcoin peers and will therefore lose their value. Besides the concerns/issues related to the management and maintenance of such lists, this approach is not sufficient, when used alone, to deter misbehavior, as misbehaving users can be equipped with many addresses, each containing low balances.

Therefore, one natural question that emerges is whether it is possible to *link* different Bitcoin addresses of the same (misbehaving) user (address linkability). If such linking were possible, misbehaving users could receive some degree of punishment for their misbehavior by not being able to spend (a large fraction of) their funds. Clearly, this comes at odds with user privacy, which is strongly coupled with the notion of address unlinkability. More specifically, although privacy in Bitcoin reduces to activity unlinkability, accountability is strongly coupled with the traceability of a user's transactions. In this section, we analyze the tension between privacy and accountability in Bitcoin. Our analysis aims to answer the following question: *to which extent can one infer information about users in Bitcoin?*

### 4.1. Methodology

We frame the preceding question by defining the information leakage using novel privacy and accountability definitions of Bitcoin. More specifically, we observe the public log of Bitcoin, denoted by $\mathsf{pubLog}$, within a period of time $\Delta$. During this period, $\mathrm{n_U}$ users, $\mathrm{U} = \{u_1, u_2, \ldots, u_{\mathrm{n_U}}\}$, participate in $\mathsf{pubLog}$ through a set of $\mathrm{n_A}$ addresses: $\mathsf{A} = \{a_1, a_2, \ldots, a_{\mathrm{n_A}}\}$. We assume that within $\Delta$, $\mathrm{n_T}$ transactions have taken place as follows: $\mathrm{T} = \{\tau_1(\mathsf{S}_1 \rightarrow \mathsf{R}_1), \ldots, \tau_{\mathrm{n_T}}(\mathsf{S}_{\mathrm{n_T}} \rightarrow \mathsf{R}_{\mathrm{n_T}})\}$, where $\tau_i(\mathsf{S}_i \rightarrow \mathsf{R}_i)$ denotes a transaction with (unique) ID number $i$ and $\mathsf{S}_i$ and $\mathsf{R}_i$ denote the sets of senders' addresses and recipients' addresses, respectively. Given $\mathsf{pubLog}$, we quantify and evaluate the privacy and accountability provisions of Bitcoin. In the sequel, we assume a security parameter $\kappa$; the security parameter can be seen as a measure of the running time of our adversary.

### 4.2. Quantifying Privacy and Accountability in Bitcoin

*Activity linkability* refers to the ability of an adversary $\mathcal{A}$ to link two different addresses (address linkability) or transactions (transaction linkability) that pertain to the same user of the system, and in this sense, activity unlinkability is strongly associated to accountability. In other words, the more a third party, such as law enforcement, is able to reconstruct the set of addresses or transactions of an individual, the easier it is to make Bitcoin users accountable for any misbehavior. More specifically, fee-based punishments for double-spending acts could be more effective, such as by blacklisting or invalidating the BTCs of the addresses that are linked to the double-spender address.

However, activity linkability seems to contradict the privacy requirements of a payment system with public transaction logs as Bitcoin, where it is crucial to maintain the confidentiality of each individual's balance and transactions. Therefore, we see *activity unlinkability* as the privacy-preserving complement of linkability.

We note that since two Bitcoin transactions are not more linkable than the addresses that participate in those transactions, we focus our analysis on unlinkability of addresses. In particular, we define address unlinkability through the following AddUnl game, and we quantify it by assessing the advantage of an adversary $\mathcal{A}$ in winning this game over an adversary who responds to all game challenges with random guesses, $\mathcal{A}^R$. We assume that $\mathcal{A}$ has access to $\mathsf{pubLog}$ and that both $\mathcal{A}$ and $\mathcal{A}^R$ have gathered (the same) a priori knowledge $\mathcal{K}_{\mathcal{A}}$ with respect to correlations of a subset of addresses. $\mathcal{K}_{\mathcal{A}}$ can include any information related to address ownership, such as the identity of the owner of the address, the transactional habits of the latter, and whether two specific addresses are owned by the same individual. For simplicity, we assume in the following that $\mathcal{K}_{\mathcal{A}}$ consists of a list of probabilities of correlating every pair of addresses in $\mathsf{pubLog}$; clearly, the correlation probability between addresses for which the adversary has no prior knowledge about equals the default probability that the two addresses are owned by the same individual (depending on the assumed game). The adversary can gather this a priori knowledge, such as by interacting with users in the system [Meiklejohn et al. 2013].

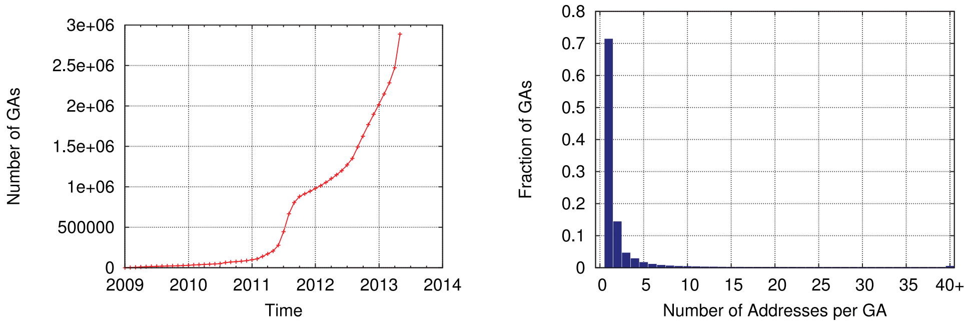
We construct the following address unlinkability game in Bitcoin, AddUnl, which consists of an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ who knows the correct assignment of addresses to Bitcoin entities. The adversary $\mathcal{A}$ chooses an address $a_0$ chosen among the addresses that appear in $\mathsf{pubLog}$, but for which the adversary has no prior knowledge (expressed in $\mathcal{K}_{\mathcal{A}}$), and sends it to the challenger $\mathcal{C}$. The challenger $\mathcal{C}$ chooses a bit $b$ uniformly at random. If $b = 1$, then $\mathcal{C}$ chooses another address $a_1$ randomly from $\mathsf{pubLog}$ such that $a_0, a_1$ belong to the same user; otherwise, $\mathcal{C}$ randomly chooses $a_1$ such that the two addresses are owned by different users. The challenger sends $\langle a_0, a_1 \rangle$ to $\mathcal{A}$, who responds with her estimate $b'$ on whether the two addresses belong to the same user. $\mathcal{A}$ wins the game if she answers correctly (i.e., $b = b'$). We say that Bitcoin satisfies address unlinkability if for all probabilistic polynomial time (p.p.t.) adversaries $\mathcal{A}$, and $\forall \langle a_0 \rangle$, $\mathcal{A}$ has only at most a negligible advantage over $\mathcal{A}^{\mathcal{R}}$ in winning—that is, if

$$\text{Prob}[b' \leftarrow \mathcal{A}(\mathsf{pubLog}, \mathcal{K}_{\mathcal{A}}, a_0, a_1) : b = b'] - \text{Prob}[b' \leftarrow \mathcal{A}^{\mathcal{R}}(\mathcal{K}_{\mathcal{A}}, a_0, a_1) : b = b'] \leq \varepsilon,$$

where $\varepsilon$ is negligible with respect to the security parameter $\kappa$.

*Quantifying address (un)linkability.* In what follows, we quantify the unlinkability offered by Bitcoin by measuring the *degree* to which Bitcoin addresses can be *linked* to the same user. To do so, we express the estimate of $\mathcal{A}$ through an $\mathrm{n_A} \times \mathrm{n_A}$ matrix, $\mathrm{E_{link}}$, where $\mathrm{E_{link}}[i, j] = \{p_{i,j}\}_{i,j \in [1, \mathsf{n_A}]}$. In other words, for every address $a_i$, $\mathcal{A}$ assesses the probability $p_{i,j}$ with which $a_i$ is owned by the same user as every other address $a_j$ in $\mathsf{pubLog}$. Note that $\mathrm{E_{link}}$ incorporates $\mathcal{K}_{\mathcal{A}}$ and any additional information that $\mathcal{A}$ could extract from $\mathsf{pubLog}$ (by means of clustering, statistical analysis, etc.). Similar to Pfitzmann and Hansen [2008], we quantify the success of $\mathcal{A}$ in the AddUnl game as follows. Let $\mathrm{GT_{link}}$ denote the genuine address association matrix—that is, $\mathrm{GT_{link}}[i, j] = 1$ if $a_i$ and $a_j$ are of the same user and $\mathrm{GT_{link}}[i, j] = 0$ otherwise for all $i, j \in [1, \mathrm{n_A}]$. For each address $a_i$, we compute the error in $\mathcal{A}$'s estimate—that is, the distance of $\mathrm{E_{link}}[i, *]$ from the genuine association of $a_i$ with the rest of the addresses in $\mathsf{pubLog}$, $||\mathrm{E_{link}}[i, *] - \mathrm{GT_{link}}[i, *]||$, where $|| \cdot ||$ denotes the L1 norm of the corresponding row matrix. Thus, the success of $\mathcal{A}$ in AddUnl, $\mathrm{Succ}_{\mathcal{A}}$, can then be assessed through $\mathcal{A}$'s maximum error: $\max_{\forall a_i \notin \mathcal{K}_{\mathcal{A}}}(||\mathrm{E_{link}}[i, *] - \mathrm{GT_{link}}[i, *]||)$.

Similarly, we represent the estimate of $\mathcal{A}^{\mathcal{R}}$ in the AddUnl game for all possible pairs of addresses by the $\mathrm{n_A} \times \mathrm{n_A}$ matrix $\mathrm{E}^{\mathcal{R}}_{\mathrm{link}}$, which is constructed as follows. $\mathrm{E}^{\mathcal{R}}[i, j] = \pi_{i,j}$ if $\langle a_i, a_j \rangle \in \mathcal{K}_{\mathcal{A}}$, and $\mathrm{E}^{\mathcal{R}}_{\mathrm{link}}[i, j] = \rho + (1 - \rho)\frac{1}{2}$ otherwise. Here, $\pi_{i,j}$ represents the probability that addresses $a_i$ $a_j$ correspond to the same user according to $\mathcal{K}_{\mathcal{A}}$, and $\rho$ is the fraction of addresses that cannot be associated to other addresses (i.e., when their owners have only one address). For pairs of addresses that are not included in $\mathcal{K}_{\mathcal{A}}$, this probability equals to $\frac{1}{2}(1 + \rho)$—that is, to the probability that at least one of the two



(a) Evolution of GA dynamics

(b) Number of addresses per GA

Fig. 9. Number of addresses until June 2013 and the number of addresses per GA.

happens: (1) $a_0$ is the only address of its owner or (2) $\mathcal{A}^{\mathcal{R}}$ did not succeed in guessing $b$ correctly.

Given this, we measure the degree of address linkability in Bitcoin by evaluating the additional success that $\mathcal{A}$ can achieve from publog when compared to $\mathcal{A}^{\mathcal{R}}$. We call this advantage $\mathrm{Link}^{\mathrm{abs}}_{\mathcal{A}} = \mathrm{Succ}_{\mathcal{A}} - \mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}$ and its normalized version $\mathrm{Link}_{\mathcal{A}} = \frac{\mathrm{Succ}_{\mathcal{A}} - \mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}}{\mathrm{Succ}\mathcal{A}^{\mathcal{R}}}$.

Address unlinkability can then be measured by the normalized complement of $\mathrm{Link}^{\mathrm{abs}}_{\mathcal{A}}$, $\mathrm{UnLink}_{\mathcal{A}} = 1 - \frac{\mathrm{Succ}_{\mathcal{A}} - \mathrm{Succ}_{\mathcal{A}^{\mathcal{R}}}}{\mathrm{Succ}\mathcal{A}^{\mathcal{R}}}$.

### 4.3. Exploiting Existing Bitcoin Client Implementations

Current Bitcoin client implementations enable $\mathcal{A}$ to link a fraction of Bitcoin addresses that belong to the same user.

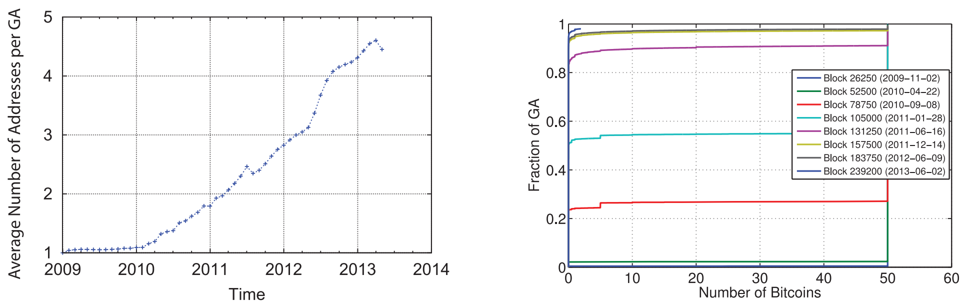
*Heuristic I—multi-input transactions*. As mentioned earlier, multi-input transactions occur when $u$ wishes to perform a payment and the payment amount exceeds the value of each of the available BTCs in $u$'s wallet. In fact, existing Bitcoin clients choose a set of BTCs from $u$'s wallet (such that their aggregate value matches the payment) and perform the payment through multi-input transactions. It is therefore straightforward to conclude that if these BTCs are owned by different addresses, then the input addresses belong to the same user [Ron and Shamir 2013; Androulaki et al. 2013].

*Heuristic II—"shadow" addresses*. As mentioned earlier, the standard Bitcoin client generates a new address, the "shadow" address [Bitcoin 2013], on which each sender can collect back the "change" [Androulaki et al. 2013].

This mechanism suggests a distinguisher for shadow addresses. Namely, in the case when a Bitcoin transaction has $n$ output addresses, $\{a_{R_1}, \cdot, a_{R_n}\}$, such that only one address is a new address (i.e., an address that has never appeared in pubLog before), and all other addresses correspond to an old address (an address that has appeared previously in pubLog), we can safely assume that the newly appearing address constitutes a shadow address for $a_i$. Note that the official Bitcoin client started to support transactions with multiple recipients since December 16, 2010.

*Evaluating Heuristics I and II*. In what follows, we evaluate the implications of these heuristics on user privacy and accountability in Bitcoin. For that purpose, we modified the blockchain parser in Znort987 [2013] to parse the first 239,200 blocks (June 2013).

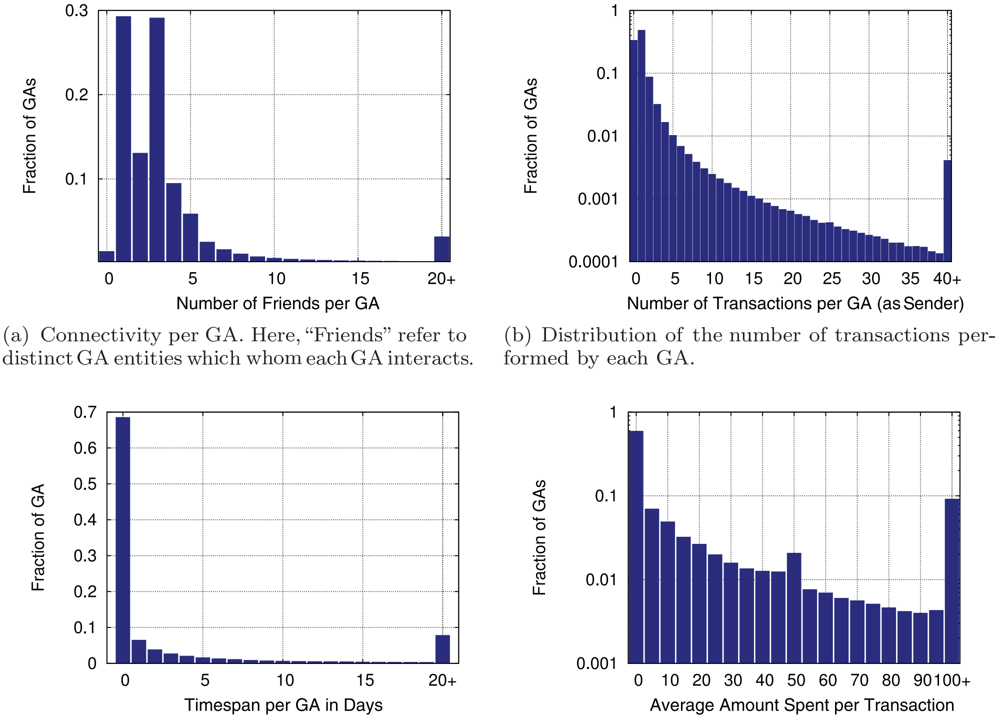
Our modified C++ parser extracts all of the addresses in each block and categorizes them in clusters of general addresses (GAs) given the two aforementioned heuristics. The parser then outputs a list of addresses organized in different GAs. Our results are depicted in Figures 9, 10, and 11.



(a) Evolution of the number of addresses per GA

(b) Evolution of the balance of GA

Fig. 10. Evolution of GA dynamics with time.



(a) Connectivity per GA. Here, "Friends" refer to distinct GA entities which whom each GA interacts.

(b) Distribution of the number of transactions performed by each GA.

(c) Lifetime of GAs.

(d) Average amount per transaction within GAs.

Fig. 11. Characterization of GAs obtained using Heuristics I and II.

As shown in Figure 9(a), our results show that the number of GAs considerably increased over time since the genesis of Bitcoin. Our parser distinguishes almost 3,000,000 GAs, each comprising on average 4.5 addresses. Figure 9(b) depicts the distribution of addresses per GA. More than 70% of GAs comprise of a single address; we believe that these are addresses of "inactive" users in the system who only "mine" and collect BTCs but rarely perform transactions in the system. Nevertheless, our heuristics enable the linking of addresses of up to 30% of the GAs. Note that our parser identified a number of very large GAs (i.e., comprising more than 50 addresses). We believe that these are addresses of big players in the Bitcoin ecosystem, such as GAs of electronic wallets, marketplaces, or mining pools. For example, we were able to distinguish 4,238,361 addresses belonging to one GA corresponding to the mining pool deepbit [Deepbit 2011].

In Figure 10(a), we show the evolution of the average number of addresses per GA with respect to time. Here, it is interesting to note that the number of addresses per GA only started increasing 1 year after the genesis block. This supports our observation that at the start, most Bitcoin users were inactive and simply were involved in the mining process. Recall that our heuristics can only link addresses of active entities that participate in Bitcoin transactions. In Figure 10(b), we depict the accumulated balance per GA over time. Until the end of 2009, we can see that nearly no GA had fewer than 50 BTCs. This is consistent with the fact that within the first 6 months of Bitcoin's existence, few transactions have been conducted and block generation was awarded with 50 BTCs. On the other hand, our measurements in June 2013 show that 98% of the GAs have fewer than 2 BTCs. This is reminiscent to the fact that Bitcoin users have become increasingly active and are spreading their coins across several of their addresses, collecting transaction fees, and so forth.

In Figure 11, we capture the information leakage due to our heuristics from the public Bitcoin log. Figure 11(a) depicts the relationship among various GAs; more specifically, we show the number of different GA entities that send transactions to other GAs. We refer to GAs that interact with each other as *friends*. Similarly, Figure 11(b) shows the number of transactions captured within each GA; our results show that almost 35% of the GAs did not participate in any transaction and as such were only mining and collecting BTCs. On the other hand, most GAs that were captured by our heuristics participated in fewer than five transactions. This is the case since our best-effort heuristics could only track the transactions performed by GAs within the period of few weeks; as shown in Figure 11(c), these heuristics were ineffective in tracking the activity of GAs for longer periods. Finally, in Figure 11(d), we depict the average transaction amount (in BTCs) per GA.

Clearly, our results in Figures 9, 10, and 11 hint that an adversary might be able to win the AddUnl game with high probability. Indeed, simply by observing the public Bitcoin log, the adversary can link some of the addresses and transactions of entities. However, our heuristics can only cluster a small number of addresses and link addresses that perform transactions close in time. In Section 4.4, we show that the advantage of the adversary in linking Bitcoin addresses can be significantly boosted when combining the use of Heuristics I and II with standard clustering algorithms.

As an application of our analysis, we identified two Bitcoin addresses belonging to Torservers.net (using information available from blockchain.info). Given the knowledge of these two addresses, we were able to identify a total of 47 addresses, belonging to the operator of Torservers, with a total balance of 498.20 BTC.

Note that it is not easy for the adversary to evade both Heuristics I and II. In other words, to evade Heuristic I, the adversary should not combine its coins/addresses in a single transaction; considering that it is unlikely that all payments made by the adversary will exactly match the coins in her possession, a large number of shadow addresses pertaining to the adversary will be created to collect the change. These addresses can be subsequently captured by Heuristic II [Androulaki et al. 2013].

One way to evade Heuristics I and II would be for the adversary to rely on mixers, such as Bitcoin banks, Web wallets, or CoinJoin [CoinJoin 2013]. These mixers shuffle the coins of all of their customers, thus preventing an external entity to link between the inputs and outputs of a transaction. Clearly, however, this strategy does not protect against an honest but curious mixer and only provides a degree of anonymity for the adversary against an external entity (e.g., which is not participating in the CoinJoin protocol). Furthermore, it is not clear whether such mixing techniques (e.g., Bonneau et al. [2014], CoinJoin [2013], and Ruffing et al. [2014]) can completely hide the profiles of Bitcoin users. As we show in the following, the amounts in each payment and the transaction times constitute a strong distinguisher for an adversary to cluster the transactions/addresses of Bitcoin users.

### 4.4. Clustering Bitcoin Addresses

Besides exploiting current Bitcoin implementations, clustering techniques can also be used to link different addresses. In what follows, we proceed to measure the effectiveness of linking different addresses in Bitcoin using clustering algorithms such as K-means clustering (KMC) and hierarchical agglomerative clustering (HAC) algorithms. We also measure the degree of address unlinkability.

To evaluate the success of $\mathcal{A}$ in the AddUnl game, we simulate a realistic case of using Bitcoin in the Department of Computer Science at ETH Zurich. Here, we assume that the shops located around the university also accept BTCs as a currency. Given the lack of details and statistics about the current use of Bitcoin, this was one of the few "workable" uses of Bitcoin that we could try to accurately model and through which we could evaluate the advantage of $\mathcal{A}$ in the system.

*Experimental setup*. To evaluate the privacy implications of using Bitcoin in a university environment, we constructed a Bitcoin simulator and devised the setup shown in Figure 12. Our Bitcoin simulator takes an XML configurations file as input and outputs: (1) a log that details the events that were simulated, the "ground truth," as well as (2) the resulting simulated public Bitcoin log, pubLog. The XML configurations file contains all of the necessary parameters to run the simulator. These include the number of users, the number of miners, the simulation time, the difficulty in block generation, and usage configurations for creating user profiles and Bitcoin sellers/buyers.

Our simulator is round based; in each simulation round (defined as a "weekly timestepping" interval), events are added to a priority queue with a probability dictated by the configuration file. These events correspond to one of the following operations:

—*Issue a new transaction:* Users might issue new Bitcoin transactions whose time, value, beneficiary, and purpose stem from the XML configurations file. The process of transaction issuance in our simulator fully mimics its counterpart in the genuine Bitcoin system.

—*Generate a new Bitcoin address:* Here, "privacy-aware" users might decide to manually generate a number of new addresses to further obfuscate their usage of Bitcoin [Ron and Shamir 2013]. This captures the behavior of an adversary who tries to prevent linking of its addresses by constantly creating new addresses.

Our Bitcoin simulator abstracts away network delays, congestion, jitter, and so forth. We also assume that all transactions in the system are well formed, and we do not model transaction fees that are incurred in the network. Moreover, our simulator relies on a variant coin selection algorithm that approximates the algorithm used in Bitcoin for coin selection; our greedy algorithm chooses the coins that will result in the smallest number of inputs for any given transaction. Throughout our experiments, we assume that new blocks are generated every 20 minutes on average; by doing so, we ensure that the number of transactions confirmed within blocks generated by our simulator is comparable to that of Bitcoin.

As shown in Figure 12, the outputs of our simulator are used to evaluate $\mathcal{A}$'s success. In fact, once the simulations terminate, a Perl-based parser uses the simulated Bitcoin block as input and preclassifies the simulated addresses into GAs according to Heuristics I and II. The resulting "prefiltered output" is then fed into our clustering algorithms—the HAC and the KMC algorithms (both implemented in C). The output of these algorithms is then compared using another Perl-based script with the ground truth to evaluate the success of $\mathcal{A}$.

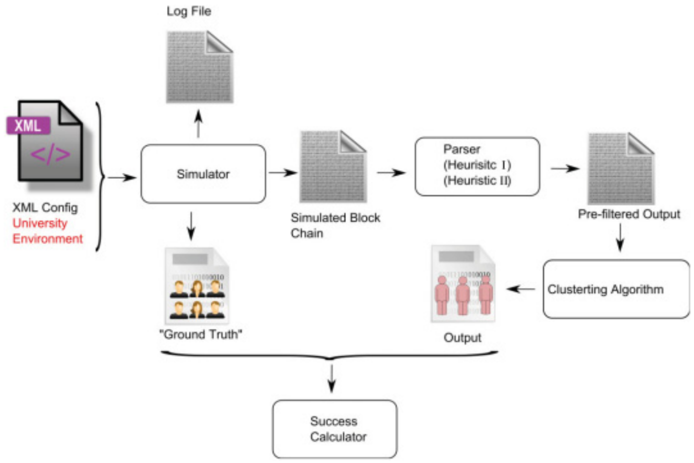


Fig. 12. Experimental setup used throughout our simulations. The outputs of our Bitcoin simulator are prefiltered according to Heuristics I and II and then fed as input to our clustering algorithm. The clustering result is then compared with the “ground truth” that is emulated by our simulator.

We tuned our simulator to match a real-world scenario that reflects the actual behavior of the staff and student members in a computer science department of a university in the fall 2012 semester. In our setting, we consider a variable number of users, 5.2% of which are “Professors,” 42.0% are “Staff,” and the remaining 52.8% are “Students.” We consider a total of six events, each having several options: lunching/dining (12 options), buying groceries (2 options), buying from vending machines (4 options), online shopping (5 options), purchasing books (2 options), and performing barters with other users, totaling 25 different Bitcoin vendors present in our system. For each user, we assign a probability that the user undergoes each of the possible options of each event. These probabilities are assigned according to the “category” of the user—that is, if the user is a “Professor,” then it is more likely that he or she would eat lunches at more expensive restaurants compared to the case where the user falls in the “Student” category. For each event, we specify in the XML configuration the following parameters: the frequency of the event and the price range per option of the event. Note that each option is assigned a rating that would reflect its popularity. The probability of performing an option is interpolated from the frequency of occurrence of the event per week and from the rating of the option. To ensure a large variety of profiles in our user base, we specify a minimum and maximum value for the frequency, rating, and price fields in the XML configuration. These bounds depend on the category of the user, the event, and option in question. At the start of our experiments, users originally have few ($<$10) Bitcoin addresses; as they issue new transactions, new (shadow) addresses are created in their wallets. In the XML file, we also model the behavior of “privacy-aware” users. We assume that these users create new Bitcoin addresses in their wallets and send some of their BTCs from their old to their new addresses.

*Clustering addresses*. We cluster addresses in our setting using the KMC, and the HAC algorithms. The HAC algorithm assumes that initially each GA represents a separate cluster ($\{z_i = i\}_{i=1}^{\mathrm{n_{GA}}}$) and computes similarity values for each pair of clusters. Clusters with higher similarity value are combined into a single cluster, and cluster-to-cluster similarity values are recomputed. The process continues until the number of created clusters equals the number of users $\mathrm{n_U}$. KMC is then initialized using the output of HAC and assumes that each user is represented by the center of each cluster. The algorithm iterates assignments of GAs to clusters and aims at minimizing the overall distance of GAs to the center of the cluster to which they have been assigned. The centers of the clusters and the GA-to-cluster distances are recomputed in each round.

Let $\mathsf{U}$ be the set of users populating Bitcoin and $(\mathsf{GA}_1, \ldots, \mathsf{GA}_{\mathrm{n_{GA}}})$ denote the GAs that $\mathcal{A}$ has obtained by applying the two aforementioned heuristics on $\mathsf{pubLog}$, respectively. Given this, the goal of $\mathcal{A}$ is to output a group of clusters of addresses $\mathrm{E_{prof}} = \{\mathsf{g}_1, \ldots, \mathsf{g}_{n_{\mathrm{U}}}\}$ such that $\mathrm{E_{prof}}$ best approximates $\mathsf{U}$. Since each GA is owned by exactly one user, the estimate on the assignment of each $\mathsf{GA}_i$ can be modeled by a variable $z_i$ such that $z_i = k$ if and only if $\mathsf{GA}_i$ belongs to $\mathsf{g}_k$. In our implementation, we represent each transaction that appears within a GA using (1) the time at which the transaction took place, (2) the indexes of the different GAs that appear within the transaction (as senders or recipients), and (3) the values of the BTCs spent by the transaction. Let $\tau_x$ denote the set of transactions of $\mathsf{GA}_x$. The degree of similarity between $\mathsf{GA}_i$ and $\mathsf{GA}_j$, denoted by $\mathsf{Sim}^{\mathsf{hac}}(\mathsf{GA}_i, \mathsf{GA}_j)$, is then represented by the cosine similarity of lists $\tau_i$ and $\tau_j$—that is, $\mathsf{Sim}^{\mathsf{hac}}(\mathsf{GA}_i, \mathsf{GA}_j) = \frac{\sum_{\forall \tau \in \tau_i \cap \tau_j} (f_{(\tau,i)} \cdot f_{(\tau,j)})}{\|\tau_i\| \cdot \|\tau_j\|}$, where $f_{(\tau,i)}$, $f_{(\tau,j)}$ are the occurrences of item $\tau$ in lists $\tau_i$ and $\tau_j$, respectively, and $\|X\|$ denotes the L2 norm of vector $X$. Given this, the resulting distance metric in KMC is $\mathsf{Dist}^{\mathsf{kmc}}(\mathsf{GA}_i, \mathsf{g}_k) = \frac{2}{1+\mathsf{Sim}^{\mathsf{hac}}(\mathsf{GA}_i, \mathsf{g}_k)} - 1$.

Our implementation also accounts for constraints that are posed in realistic deployments. Namely, since users cannot be physically located in two different places at the same time, they cannot participate in two different (physical) exchanges of goods at the same time. To account for this case, we apply different weighting for similarity of GAs who participate in transactions concurrently.

We quantify the success of $\mathcal{A}$ in clustering addresses in Bitcoin by measuring the similarity of $\mathcal{A}$'s estimate $\mathrm{E_{prof}}$ from the genuine grouping of profiles $\mathrm{GT_{prof}}$, $\mathrm{Sim}(\mathrm{E_{prof}}, \mathrm{GT_{prof}})$, where the similarity function Sim ranges in [0, 1]. Similar to quantifying address unlinkability, we assess the advantage of $\mathcal{A}$ in approximating $\mathrm{GT_{prof}}$ over $\mathcal{A}^{\mathcal{R}}$ by $\mathrm{Prof}_{\mathcal{A}} = \mathrm{Sim}(\mathrm{E_{prof}}, \mathrm{GT_{prof}}) - \mathrm{Sim}(\mathrm{E}^{\mathcal{R}}_{\mathrm{prof}}, \mathrm{GT_{prof}})$.

We evaluate $\mathrm{Sim}(\mathrm{E_{prof}}, \mathrm{GT_{prof}})$ and $\mathrm{Prof}_{\mathcal{A}}$ by relying on two commonly used entropy-based distance metrics, namely the *normalized mutual information* (NMI) and the *adjusted mutual information* (AMI). NMI assesses the similarity of two groupings of the same items (in our case, $\mathrm{E_{prof}}$ and $\mathrm{GT_{prof}}$) and takes higher values (1) the more identical the groupings [Vinh et al. 2009, 2010]. On the other hand, given the two groupings $\mathrm{E_{prof}}$ and $\mathrm{GT_{prof}}$, AMI approaches 0 when $\mathrm{E_{prof}}$ is close to random assignment of addresses/transactions to groups (i.e., $\mathrm{E}^{\mathcal{R}}_{\mathrm{prof}}$) and is 1 when $\mathrm{E_{prof}}$ matches $\mathrm{GT_{prof}}$ [Vinh et al. 2009, 2010]. Assuming address-based profiles, NMI and AMI are computed as follows:

$$\mathrm{NMI} = \frac{\mathcal{I}(\mathrm{E_{prof}}, \mathrm{GT_{prof}})}{\max(\mathrm{H}(\mathrm{E_{prof}}), \mathrm{H}(\mathrm{GT_{prof}}))}, \quad \mathrm{AMI} = \frac{\mathcal{I}(\mathrm{E_{prof}}, \mathrm{GT_{prof}}) - \mathcal{E}}{\max(\mathrm{H}(\mathrm{E_{prof}}), \mathrm{H}(\mathrm{GT_{prof}})) - \mathcal{E}},$$

where

$$\mathcal{I}(\mathrm{E_{prof}}, \mathrm{GT_{prof}}) = \sum_{i=1}^{\mathrm{n_U}} \sum_{j=1}^{\mathrm{n_U}} \frac{n_{(i,j)}}{\mathrm{n_A}} \log\left(\frac{n_{(i,j)} \cdot \mathrm{n_A}}{n_{(i,*)} n_{(*,j)}}\right),$$

$$\mathrm{H}(\mathrm{E_{prof}}) = -\sum_{i=1}^{\mathrm{n_U}} \frac{n_{(i,*)}}{\mathrm{n_A}} \log\left(\frac{n_{(i,*)}}{\mathrm{n_A}}\right), \quad \mathrm{H}(\mathrm{GT_{prof}}) = -\sum_{j=1}^{\mathrm{n_U}} \frac{n_{(*,j)}}{\mathrm{n_A}} \log\left(\frac{n_{(*,j)}}{\mathrm{n_A}}\right),$$

$$\mathcal{E} = \sum_{i=1}^{\mathrm{n_U}} \sum_{j=1}^{\mathrm{n_U}} \sum_{n \in \mathcal{M}} \frac{n}{\mathrm{n_A}} \log\left(\frac{\mathrm{n_A} n}{n_{(i,*)} n_{(*,j)}}\right) \frac{n_{(i,*)}! n_{(*,j)}! (\mathrm{n_A} - n_{(i,*)})! (\mathrm{n_A} - n_{(*,j)})!}{\mathrm{n_A}! (n_{(i,*)} - n)! (n_{(*,j)} - n)! (\mathrm{n_A} - n_{(i,*)} - n_{(*,j)} - n)!}.$$

Here, $\mathrm{n_A}$ is the number of Bitcoin addresses, $n_{(i,j)}$ is the number of $u_i$'s addresses, which are assigned to group $\mathsf{g}_j$, and $n_{(i,*)}$ and $n_{(*,j)}$ are the number of addresses of $u_i$

Table IV. Clustering Results in the “Partial Knowledge” and “No Knowledge” Scenarios

| Partial Knowledge | | 100 (50%) | 200 (0%) | 200 (50%) | 200 (100%) | 400 (50%) |
|---|---|---|---|---|---|---|
| $\mathrm{Link}_{\mathcal{A}}$ | | 0.91 ±0.01 | 0.90 ±0.01 | 0.91 ±0.01 | 0.92 ±0.01 | 0.93 ±0.01 |
| $\mathrm{Prof}^{a}_{\mathcal{A}}$ | NMI | 0.76 ± 0.01 | 0.87 ± 0.01 | 0.79 ± 0.01 | 0.70 ± 0.01 | 0.80 ± 0.01 |
| | AMI | 0.75 ± 0.01 | 0.86 ± 0.01 | 0.77 ± 0.01 | 0.68 ± 0.01 | 0.77 ± 0.01 |
| $\mathrm{Prof}^{\tau}_{\mathcal{A}}$ | NMI | 0.68 ± 0.01 | 0.73 ± 0.02 | 0.70 ± 0.01 | 0.65 ± 0.01 | 0.72 ± 0.01 |
| | AMI | 0.67 ± 0.01 | 0.72 ± 0.01 | 0.69 ± 0.01 | 0.63 ± 0.01 | 0.70 ± 0.01 |
| No Knowledge | | 100 (50%) | 200 (0%) | 200 (50%) | 200 (100%) | 400 (50%) |
| $\mathrm{Link}_{\mathcal{A}}$ | | 0.90 ±0.01 | 0.90 ±0.01 | 0.91 ±0.01 | 0.92 ±0.01 | 0.93 ±0.01 |
| $\mathrm{Prof}^{a}_{\mathcal{A}}$ | NMI | 0.79 ± 0.01 | 0.89 ± 0.01 | 0.79 ± 0.01 | 0.71 ± 0.02 | 0.80 ± 0.01 |
| | AMI | 0.78 ± 0.02 | 0.88 ± 0.01 | 0.78 ± 0.02 | 0.69 ± 0.02 | 0.78 ± 0.01 |
| $\mathrm{Prof}^{\tau}_{\mathcal{A}}$ | NMI | 0.69 ± 0.01 | 0.73 ± 0.03 | 0.69 ± 0.03 | 0.65 ± 0.01 | 0.72 ± 0.01 |
| | AMI | 0.68 ± 0.01 | 0.72 ± 0.01 | 0.68 ± 0.03 | 0.63 ± 0.01 | 0.70 ± 0.01 |

*Note*: A column entitled $X$ ($Y$%) denotes an experiment featuring $X$ users among which $Y$% are privacy aware. Each data point in our plots is averaged over five rounds of experiments; we also present the corresponding 95% confidence intervals (shown after the “±” sign).

and $\mathsf{g}_j$, respectively. $\mathcal{E}$ reflects the *expected* mutual information between $\mathrm{GT}_{\mathrm{prof}}$ and a random grouping of addresses ($\mathrm{E}^{\mathcal{R}}_{\mathrm{prof}}$). In addition, $\mathcal{M} = [\max(n_{(i,*)} + n_{(*,j)} - \mathrm{n_A}, 0), \min(n_{(i,*)}, n_{*,j})]$. Similar calculations can be derived to compute NMI and AMI for transaction-based profiles.

*Experimental results*. Throughout our experiments, we emulated two different scenarios for each simulation round. In the first scenario, denoted by “Partial Knowledge,” we assume that $\mathcal{A}$ is aware of the location/service of all Bitcoin vendors and as such can distinguish whether a transaction was performed in exchange of a physical good. In this case, we include the vendors’ addresses in the prior knowledge of $\mathcal{A}$ when computing $\mathrm{Link}_{\mathcal{A}}$; we also assume that $\mathcal{A}$ can tune the clustering algorithm to take into account that the same user performing this transaction cannot appear in other transactions that take place at the same time. This case emulates the realistic setting where $\mathcal{A}$ can extract a subset of the addresses owned by geographically co-located Bitcoin users/vendors from the overall public Bitcoin log; for example, $\mathcal{A}$ can extract from the Bitcoin log all addresses that interact with a known address of a vendor located within the university environment. In the second scenario, denoted by “No Knowledge,” we consider the case where $\mathcal{A}$ does not know the location or service of the vendors, and as such does not have any prior knowledge, but assumes that up to 10% of the transactions are performed in exchange of goods delivered over the Internet.

Given this setup, we evaluate the metrics $\mathrm{Link}_{\mathcal{A}}$, $\mathrm{Prof}^{a}_{\mathcal{A}}$ (for address-based profiles), and $\mathrm{Prof}^{\tau}_{\mathcal{A}}$ (for transaction-based profiles) with respect to (1) the fraction of “privacy-aware” users and (2) the number of users $\mathrm{n_U}$. By privacy-aware users, we refer to users that manually generate new Bitcoin addresses (following a configuration in the XML file) to enhance their privacy in the system. Table IV depicts our findings. Our results show that both the “Partial Knowledge” and the “No Knowledge” configurations exhibited comparable results.

In the first round of experiments, we evaluate the success of $\mathcal{A}$ with respect to the fraction of privacy-aware users. More specifically, we run our clustering and accountability (privacy) evaluation algorithm in a setting featuring 200 users, among which 0%, 50%, and 100% of the users are privacy aware. The table shows $\mathrm{Link}_{\mathcal{A}}$, $\mathrm{Prof}^{a}_{\mathcal{A}}$, and $\mathrm{Prof}^{\tau}_{\mathcal{A}}$ with respect to the fractions of privacy-aware users. Here, we use a normalized version on $\mathrm{Link}_{\mathcal{A}}$.

The advantage of $\mathcal{A}$ is only negligibly affected by the fraction of the privacy-aware users in the system. More specifically, we can see that our adversary outperforms $\mathcal{A}^{\mathcal{R}}$

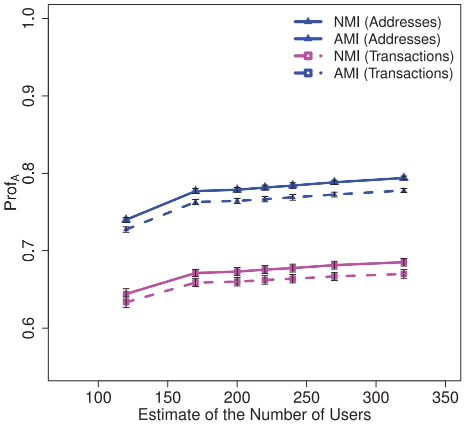


Fig. 13. The case where $\mathcal{A}$ cannot accurately estimate $n_U$. We assume the "Partial Knowledge" case where $n_U = 200$.

by almost 90%. On the contrary, $\text{Prof}_{\mathcal{A}}^{a}$ and $\text{Prof}_{\mathcal{A}}^{\tau}$ show a better dependency on the fraction of privacy-aware users. When none of users in the system are privacy aware, the performance of our clustering algorithms is high. In particular, in both configurations, $\text{Prof}_{\mathcal{A}^a}$ (NMI and AMI for addresses) range within 0.87 to 0.89, whereas $\text{Prof}_{\mathcal{A}}^{\tau}$ (NMI and AMI for transactions) are 0.73. However, as the fraction of the privacy-aware users increases, the performance of $\mathcal{A}$ drops and results in $\text{Prof}_{\mathcal{A}}^{\tau}$ and $\text{Prof}_{\mathcal{A}}^{a}$ of 0.70 and 0.63, respectively. This mischief can be explained by the fact that privacy-aware users add noise to the Bitcoin log. However, the fact that AMI values remain consistently far from 0 and close to 1 indicates that $\mathcal{A}$ performs much better than $\mathcal{A}^R$ and that the estimate chosen by $\mathcal{A}$ is close to the genuine assignment of users to clusters. We therefore conclude that the privacy of users in Bitcoin can still be compromised even if users manually create new addresses to prevent the linking of their addresses [Ron and Shamir 2013]. Furthermore, our results show that $\mathcal{A}$'s advantage over $\mathcal{A}^{\mathcal{R}}$ is not significantly affected by the number of participant users in the case of address unlinkability. $\text{Prof}_{\mathcal{A}}^{a}$ and $\text{Prof}_{\mathcal{A}}^{\tau}$ increase from 0.76 and 0.68 to 0.80 and 0.72, respectively, as the number of users increases from 100 to 400. This is mostly because when the number of users increases, the assignment of addresses (or transactions) into groups of users ($\mathcal{A}^{\mathcal{R}}$) performs worst. In Figure 13, we evaluate the case where $\mathcal{A}$ does not have an accurate estimate of the number of users in the university setting. Our findings show that even if $\mathcal{A}$'s estimate of the number of users is not accurate, the privacy of a considerable fraction of users is still compromised.

In our simulation environment, the number of transactions performed by users largely depends on their budget and "hobbies," which results in a considerable variety of frequency of transactions per user. In Figure 14, we assess the advantage of $\mathcal{A}$ with respect to different classes of users and different privacy-awareness levels. Here, we throttle the average number of transaction per user to 30%, 50%, 90%, and 100% of the total possible transactions to which all users would commit (as defined in their profile within the simulator); by doing so, we simulate cases where the network features various levels of transactional traffic. In each investigated setting, we classify the profiles of Bitcoin users into three groups based on the number of transactions in which they were involved (i.e., groups were divided according to their 3-quantiles) and measure the overall fraction of user profiles (measured by means of the similarity of transactions appearing in a user's wallet and the corresponding cluster) that are captured by $\mathcal{A}$ in each case. Our results in Figure 14 show that in the case of $n_U = 200$ users with

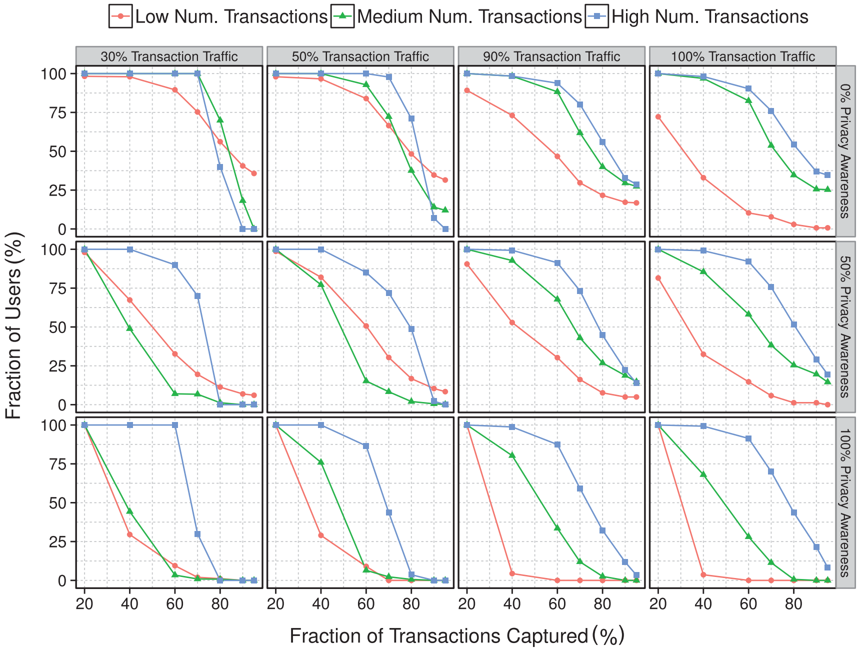


Fig. 14. Fraction of transactions captured by our clustering algorithms in the "No Knowledge" case and 200 users in the system.

0% privacy awareness and 100% transactional traffic, almost 42% of the users have their profiles captured with 80% accuracy. In the case featuring 100% privacy-aware users, this fraction drops to 35% of the users whose profile was correctly clustered with an accuracy of at least 80%. In summary, our results suggest the following:

—The profile leakage is larger when users participate in a large number of transactions and decreases as the number of transactions performed by the user decreases. This is mainly because users who participate in more transactions can be more easily profiled compared to those users who only participate in few transactions.
—The overall number of transactions exchanged in Bitcoin has little impact on the profile leakage of users in the system. Our results show that even when the network features 70% fewer transactions, the fraction of captured transactions per user does not decrease significantly—irrespective of the activity level of each user.

It is straightforward to see that accountability provisions in Bitcoin become strongeras the privacy provisions of Bitcoin become weaker. Notably, the less untraceable the user activity is within Bitcoin log, and therefore the more accountable the system, the more individual privacy, such as activity unlinkability, is compromised. This tension is also depicted in our metrics; the higher is $\text{Link}_{\mathcal{A}}$, the smaller is $\text{UnLink}_{\mathcal{A}}$.

Our results suggest that the privacy provisions of Bitcoin are not strong, which opens the door to the integration of accountability measures in the system.

As mentioned earlier, the manual creation of new addresses can only partly conceal the profiles of users who participate in a small amount of transactions. Such a countermeasure, however, does not increase the privacy of users who are active in the network and participate in a large number of transactions. Moreover, protocols that mix the inputs and outputs of transactions cannot fully prevent clustering analysis, as they do not hide the amounts and times of payments. Transaction amounts and times can be hidden, such as by using the protocols in Ben-Sasson et al. [2014] and Androulaki and Karame [2014], and by randomizing the time of sending transactions in the network. However, such protocols incur considerable computational overhead and require modifications to the Bitcoin protocol.

## 5. RELATED WORK

In what follows, we overview related work in the area. Elias [2011] investigates the legal aspects of privacy in Bitcoin. Reid and Harrigan [2011] explore user anonymity limits in Bitcoin. Babaioff et al. [2011] address the lack of incentives for Bitcoin peers to include recently announced transactions in a block. Ron and Shamir [2013] analyze the behavior of users (i.e., how they acquire and how they spend their BTCs) and investigate how users move BTCs between their various accounts to better protect their privacy. To our knowledge, their analysis was only based on Heuristic I (cf. Section 4.3) and did not take into account classifying addresses using our second heuristic. Meiklejohn et al. [2013] try to identify big players in the Bitcoin system by leveraging our Heuristics I and II presented in Section 4.3; notably, the authors perform transactions with big vendors, such as Mt. Gox, and use our heuristics to identify a bigger cluster of addresses particular to such vendors. Our analysis in Section 4.3 explores the information leakage due to the public Bitcoin block chain and reveals the first comprehensive clustering results of all Bitcoin addresses using our two heuristics. Decker and Wattenhofer [2013] investigate transaction and block propagation time in Bitcoin. Koshy et al. [2014] investigate the possibility of linking addresses of the same user together by utilizing the Bitcoin peers network address information (IPs). The authors identified (in terms of IP) certain addresses whose behavior deviated from the average address behavior in the Bitcoin network but did not perform any generic analysis covering all Bitcoin addresses. Gervais et al. [2014a] show that the reliance on Bloom filters within existing SPV clients leaks considerable information about the addresses of Bitcoin users.

To enhance user privacy in Bitcoin, mixing transactions emerges as an effective technique to hide the linkability between inputs and outputs of Bitcoin transactions. Existing solutions (e.g., Bonneau et al. [2014] and CoinJoin [2013]) rely on a mixing server to harden the tracing of coin expenditure in the network; here, the mixing server still needs to be trusted to ensure anonymity, since it learns the mapping of coins to addresses. Ruffing et al. [2014] propose a mixing protocol that does not require any centralized mixing server. Miers et al. [2013] introduce ZeroCoin, a cryptographic extension to Bitcoin that augments the protocol to prevent the tracing of coin expenditure. In this work, we have shown that standard clustering algorithms can be used to acquire considerable information about the user profiles in Bitcoin. These algorithms mainly leverage user spending patterns, such as transaction amounts and transaction times, to profile users. Clearly, ZeroCoin and existing mixing protocols can impede, to some extent, the linkability of transactions belonging to the same address. In these schemes, the advantage of our adversary in linking addresses is likely to be reduced; nevertheless, it is not clear whether clustering analysis can be completely deterred in ZeroCoin and in mixing protocols, as the transaction times, transaction amounts, and address balances can still be derived from the block chain. Garman et al. [2014] briefly describe a ZKPoK-based technique that enables the construction of transactions between anonymous coins in ZeroCoin. ZeroCoin was later extended in Androulaki and Karame [2014] and Ben-Sasson et al. [2014] to hide the transaction values and address balances in the system.

Syed and Syed [2011] propose a user-friendly technique for managing Bitcoin wallets. Gervais et al. [2014b] analyze the limits of decentralization in Bitcoin and show that the vital operations and decisions that Bitcoin is currently undertaking are not decentralized. Finney [2013] describes a double-spending attack in Bitcoin where the attacker includes in her generated blocks transactions that transfer some coins between her own addresses; these blocks are only released in the network after the attacker double-spends the same coins using fast payments and acquires a given service. Barber et al. [2012] analyze possible ways to enhance the resilience of Bitcoin against a number of security threats; they do not, however, analyze the security of fast Bitcoin payments.

Anonymity and unlinkability have been explored in different contexts by the research community. More specifically, Pfitzmann and Hansen [2008] define unlinkability and privacy for pseudonymous systems in high level. Diaz et al. [2002] and Steinbrecher and Koepsell [2003] model and measure anonymity and unlinkability in communication systems. Malin [2008] provides a model the unlinkability of distributed data, whereas Franz et al. [2007] investigate how important the context is for the unlinkability provisions of a system and provide metrics for that.

Micropayments [Rivest 2004; Androulaki et al. 2008; Karame et al. 2011] is an efficient payment scheme aiming primarily at enabling low-cost transactions. Here, the payer provides signed endorsements of monetary transfers on the vendor's name. Digital signatures in these systems constitute the main double-spending resistance mechanism. ECash [Chaum et al. 1990; Brands 1995; Camenisch et al. 2005] and anonymous credit cards were the first attempts to define privacy-preserving transactions. Privacy in ECash consists of user anonymity and transaction unlinkability; by relying on a set of cryptographic primitives, ECash ensures that payments pertaining to the same user cannot be linked to each other or to the payer, provided that the latter does not misbehave.

## 6. CONCLUSION

Bitcoin has already witnessed a wider adoption and attention than any other digital currency proposed to date. One of the main reasons for such a broad adoption of Bitcoin has been a promise of a low-cost, anonymous, and decentralized currency that is inherently independent of governments and any centralized authority.

In this article, we analyzed and evaluated the double-spending resilience of Bitcoin in fast payments. We showed that not only do these attacks succeed with overwhelming probability, but they also—contrary to common beliefs- do not incur any significant overhead on the attacker. We also proposed a lightweight countermeasure that enables the detection of double-spending attacks in fast transactions.

Motivated by our analysis, we then analytically and empirically investigated the privacy and accountability provisions in Bitcoin. Our findings show that the public transaction log of Bitcoin leaks considerable information about user profiles in Bitcoin. This information can be used to link different Bitcoin addresses pertaining to Bitcoin users to implement accountability measures within the system (e.g., "blacklist" linked addresses from the network) within the Bitcoin system. We therefore hope that our findings motivate further research in this area.

## ACKNOWLEDGMENTS

The authors would like to thank Matthias Herrmann and Hubert Ritzdorf for collecting various measurements in the Bitcoin network, and Tobias Scherer for the help in implementing the HAC and KMC algorithms. The authors also thank the anonymous reviewers for their valuable feedback and comments.

## REFERENCES

Elli Androulaki and Ghassan Karame. 2014. Hiding transaction amounts and balances in Bitcoin. In *Proceedings of the International Conference on Trust and Trustworthy Computing (TRUST)*.

Elli Androulaki, Ghassan Karame, and Srdjan Capkun. 2013. Evaluating user privacy in Bitcoin. In *Proceedings of Financial Crypto 2013*. http://eprint.iacr.org/2012/596.pdf.

Elli Androulaki, Mariana Raykova, Shreyas Srivatsan, Angelos Stavrou, and Steven M. Bellovin. 2008. PAR: Payment for anonymous routing. In *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies (PETS'08)*. 219–236.

Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. 2011. On Bitcoin and red balloons. *ACM SIGecom Exhanges* 10, 3, 5–9.

Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. 2012. Bitter to better—how to make Bitcoin a better currency. In *Financial Cryptography and Data Security*. Lecture Notes in Computer Science, Vol. 7397. Springer, 399–414.

Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Practical decentralized anonymous e-cash from Bitcoin. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*. IEEE, Los Alamitos, CA.

Bitcoin. 2013. Introduction. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Introduction.

BitcoinCharts. 2013. Bitcoin Charts. Retrieved April 2, 2015, from http://bitcoincharts.com.

BitcoinDoS. 2013. Weaknesses. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Weaknesses.

BitcoinExplorer. 2013. Bitcoin Block Explorer. Retrieved April 2, 2015, from http://blockexplorer.com.

BitcoinFAQ. 2013. FAQ. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/FAQ.

BitcoinMyths. 2013. Myths. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Myths#Point_of_sale_with_bitcoins_isn.27t_possible_because_of_the_10_minute_wait_for_confirmation.

BitcoinProtocol. 2013. Protocol Documentation. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Protocol_specification.

BitcoinRules. 2012. Protocol Rules. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Protocol_rules.

BitcoinTrade. 2013. Trade. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Trade.

BitcoinWiki. 2014. Double-Spending. Retrieved April 2, 2015, from https://en.bitcoin.it/Double-spending.

BitcoinXT. 2014. Bitcoin XT. Retrieved April 2, 2015, from https://github.com/bitcoinxt/bitcoinxt.

BlockChain. 2013. 200 Double Spends. Retrieved April 2, 2015, from https://blockchain.info/double-spends.

Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. 2014. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *Proceedings of Financial Crypto 2014*.

Stefan Brands. 1995. Electronic cash on the Internet. In *Proceedings of the Symposium on Network and Distributed System Security*. 64–84.

Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2005. Compact e-cash. In *Advances in Cryptology—EUROCRYPT 2005*. Lecture Notes in Computer Science, Vol. 3494. Springer, 302–321.

David Chaum, Amos Fiat, and Moni Naor. 1990. Untraceable electronic cash. In *Proceedings of Advances in Cryptology (CRYPTO'88)*. 319–327.

CNN. 2011. CNN: Bitcoin's Uncertain Future as Currency. Retrieved April 2, 2015, from https://www.youtube.com/watch?v=75VaRGdzMM0.

CoinJoin. 2013. CoinJoin: Bitcoin Privacy for the Real World. Retrieved April 2, 2015, from https://bitcointalk.org/index.php?topic=279249.0.

Connectivity. 2013. Satoshi Client Node Connectivity. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Satoshi_Client_Node_Connectivity.

Christian Decker and Roger Wattenhofer. 2013. Information propagation in the Bitcoin network. In *Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing*.

Deepbit. 2011. Deepbit. Retrieved April 2, 2015, from https://deepbit.net.

Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. 2002. Towards measuring anonymity. In *Proceedings of the Privacy Enhancing Technologies Workshop (PET'02)*.

DoubleSpending. 2014. Observed Double-Spends. Retrieved April 2, 2015, from http://respends.thinlink.com.

Elias. 2011. Bitcoin: Tempering the Digital Ring of Gyges or Implausible Pecuniary Privacy. Retrieved April 2, 2015, from http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1937769.

Finney. 2013. Weaknesses. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Weaknesses#The_.22Finney.22_attack.

Matthias Franz, Bernd Meyer, and Andreas Pashalidis. 2007. Attacking unlinkability: The importance of context. In *Privacy Enhancing Technologies*. Lecture Notes in Computer Science, Vol. 4776. Springer, 1–16.

Christina Garman, Matthew Green, Ian Meiers, and Aviel Rubin. 2014. Rational zero: Economic security for zerocoin with everlasting anonymity. In *Proceedings of the Financial Cryptography and Data Security Conference*.

Arthur Gervais, Srdjan Capkun, Ghassan O. Karame, and Damian Gruber. 2014a. On the privacy provisions of Bloom filters in lightweight Bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC'14)*. 326–335. DOI:http://dx.doi.org/10.1145/2664243.2664267

Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. 2014b. Is Bitcoin a decentralized currency? *IEEE Security and Privacy* 12, 3, 54–60.

Ghassan Karame, Aurelien Francillon, and Srdjan Čapkun. 2011. Pay as you browse: Microcomputations as micropayments in Web-based services. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. 307–316.

Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending fast payments in Bitcoin. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'12)*. ACM, New York, NY, 906–917. DOI:http://dx.doi.org/10.1145/2382196.2382292

Philip Koshy, Diana Koshy, and Patrick McDaniel. 2014. An analysis of anonymity in Bitcoin using p2p network traffic. In *Proceedings of Financial Crypto 2014*.

Bradley Malin. 2008. K-unlinkability: A privacy protection model for distributed data. *Data and Knowledge Engineering* 64, 1, 294–311.

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 Internet Measurement Conference (IMC'13)*. ACM, New York, NY, 127–140. DOI:http://dx.doi.org/10.1145/2504730.2504747

Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy*.

Mining. 2013. Comparison of Mining Pools. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Comparison_of_mining_pools.

MiningHardware. 2013. Mining Hardware Comparison. Retrieved April 2, 2015, from https://en.bitcoin.it/wiki/Mining_hardware_comparison.

Andreas Pfitzmann and Marit Hansen. 2008. Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management: A consolidated proposal for terminology. *Fachterminologie Datenschutz und Datensicherheit* 2008, 111–144. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.18.pdf.

ProofWiki. 2013. Definition:Geometric Distribution/Shifted. Retrieved April 2, 2015, from https://proofwiki.org/wiki/Definition:Geometric_Distribution/Shifted.

Fergal Reid and Martin Harrigan. 2011. An analysis of anonymity in the Bitcoin system. arXiv:1107.4524.

Ronald Rivest. 2004. Peppercoin micropayments. In *Financial Cryptography*. Lecture Notes in Computer Science, Vol. 3110. Springer, 2–8.

Dorit Ron and Adi Shamir. 2013. Quantitative analysis of the full Bitcoin transaction graph. In *Proceedings of Financial Crypto 2013*.

Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. 2014. CoinShuffle: Practical decentralized coin mixing for Bitcoin. In *Computer Security—ESORICS 2014*. Lecture Notes in Computer Science, Vol. 8713. Springer, 345–364.

Satoshi Nakamoto. 2009. Bitcoin: A Peer-to-Peer Electronic Cash System. Retrieved April 2, 2015, from https://bitcoin.org/bitcoin.pdf.

Sandra Steinbrecher and Stefan Koepsell. 2003. Modelling unlinkability. In *Privacy Enhancing Technologies.* Lecture Notes in Computer Science, Vol. 2760. Springer, 32–47.

Syed and Syed 2011. Bitcoin Gateway, A Peer-to-Peer Bitcoin Vault and Payment Network. Retrieved April 2, 2015, from http://arimaa.com/bitcoin/.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2009. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research* 11, 2837–2854.

Znort987. 2013. Znort987/Blockparser. (2013). Retrieved April 2, 2015, from https://github.com/znort987/blockparser.

Received December 2013; revised December 2014; accepted February 2015