An Exploration of Bitcoin Anonymity

CS203: Network and Distributed System Security

Christopher A. Wood
Department of Computer Science
University of California Irvine
Email: woodc1@uci.edu

Chris H. Vu
Department of Computer Science
University of California Irvine
Email: ChrisHVu@gmail.com

***Abstract*—TODO**

I. Introduction

Electronic commerce would benefit greatly from the existence of a complete secure, private, and anonymous form of digital currency that did not rely on trusted third parties or external financial institutions to manage transactions. Indeed, there has been significant research invested in this problem in recent years resulting in various forms of cryptography-based digital payment systems, or cryptocurrencies for short, such as DigiCash [**?**], ecash [**?**], HashCash [**?**], Namecoin [**?**], Peercoin [**?**], Litecoin [**?**], Ripple [**?**], and perhaps the most popular variant, Bitcoin [1]. Each of these schemes offer different tradeoffs of security, privacy, and anonymity, and as such have varying popularity among users. However, it is the distribtued, decentralized nature of Bitcoin that has led to its leading popularity to the general public and research communities.

More specifically, Bitcoin is distinguished from other solutions by the fact that it does not rely on trusted third parties. Specifically, the global and publicly accessible ledger which stores records of financial transactions is maintained by a widely distributed, peer-to-peer network of (untrusted) users. Even though each transaction is linked to the public key of a particular user via a digital signature, rather than their real identity, user privacy and anonymity are still at risk because public keys must be owned by specific users. This is true even if a user has multiple public keys and uses them with caution to deter attackers looking for such links. Consequently, the privacy of Bitcoin is an open problem and illustrates the difficulty in achieving a distributed form of cryptocurrency, i.e., one that does not rely on trusted third parties, and one that provides sufficiently useful characteristics such as privacy and anonymity.

Currently, techniques to address such privacy issues with Bitcoin are rather limited and include the use of Chaumian's entirely independent e-cash system [3], which relies on trusted third parties, or Zerocoin [4], which achieves privacy and anonymity at the protocol-level by working in conjunction with Bitcoin, among others. The former is not ideal for several reasons, the most significant of which is that it directly conflicts with the decentralized nature of Bitcoin. That is, reliance on trusted third parties is generally unfavorable if at all feasible. The latter technique is very young, having only been published in the past year, and is just now starting to gain considerable attention. Of course, there exists other academic efforts to further the cause for Bitcoin user privacy, including studies by Ron and Shamir [6] and Androulaki et al. [5], and we can expect to see similar work publishing in the coming years.

In this work we survey Bitcoin and related forms of cryptocurrency with respect to the security, privacy, and anonymity guarantees provided by each. We assess proposed solutions that have and have not been implemented in practice, and offer critical insight into the open problems and difficulties in achieving complete security, privacy, and anonymity with minimal resource consumption (e.g., bandwidth, computational cycles, etc.). We hope that this survey will motivate continued research on this critical problem that has the potential to change financial instutitions and forms of currency for future generations.

TODO: outline the sections here

II. Bitcoin Preliminaries

In this section we introduce some common notation used in the literature involving security, privacy, and anonymity, and then present an overview of the Bitcoin system and underlying protocol for making payments (transactions).

A. Bitcoin Basics

In what follows we distill a description of the Bitcoin system and underlying protocol from [1]; interested readers may acquire more specific details therein if required. To begin, Bitcoin is a distributed, decentralized form of cryptocurrency. Accordingly, this enables all (digitally signed) transactions between two parties to be conducted in a peer-to-peer fashion without the inclusion of a trusted third party, such as a bank or other financial institution. This form of decentralized exchange comes at a price, however, as there must be some way to prevent users from *double spending*, or using the same funds to simultaneously pay multiple parties. Bitcoin achieves this property by relying on its users to construct a history for every transaction that takes place in the system. If a majority of the users accept the validity of a particular transaction, or a set of transactions, the global history of the system is affirmatively updated and "confirmed" via a cryptographic hash digest that all users agree upon. This hash digest, referred to as a hash-based proof-of-work, is what constitutes the validity of the system. By the properties of the underlying hash function, the history of the system cannot be changed without breaking

the function (i.e., finding collisions) or re-doing the proof-of-work, which is computationally infeasible for small groups of nodes. Therefore, so long as a majority of the Bitcoin users are honest, the system history is deemed correct and thus all signed transactions are verified, preventing double spending by potentially malicious users participating in direct, peer-to-peer transactions.

Unfortunately, while the above scheme is semantically correct and provides strong guarantees that all financial transactions are valid, there are inherent limitations in the amount of user privacy and anonymity that can be achieved in Bitcoin. In order to adequately define these limitations, we first describe how Bitcoin transactions are generated and how the system history is maintained. For simplicity, consider the scenario in which user A wants to send N BTCs (Bitcoins) to user B. Rather than identify users by name, Bitcoin uses *addresses* that are tied to specific users to use in such transactions. Denote addr_A and addr_B as the addresses of users A and user B used in this transaction. It is often convenient to think of Bitcoin addresses as public keys pk_A and sf_B, and as such there are corresponding private keys, which we denote as sk_A, and sk_B, respectively.

Structurally, a transaction T is a tuple comprised of the *source* transactions which supplied the funds necessary to make this transaction, denoted as source, the (public) address of the recipient, addr_B, the amount of BTCs to send, N, and a digital signature of these three properties, $\mathsf{Sign}_{\mathsf{sk_A}}(\mathsf{source}, \mathsf{pk}_B, N)$. In other words, we have

$$T = (\mathsf{source}, \mathsf{pk}_B, N, \sigma),$$

where $\sigma = \mathsf{Sign}_{\mathsf{sk_A}}(\mathsf{source}, \mathsf{pk}_B, N)$. Note that this signature is embedded in T so that any other Bitcoin user may verify the validity of the content using pk_A. Also note that source need not be a single transaction; user A is free to use multiple transactions in order to fund their transaction to B. In addition to the N BTC transfer from A to B, there is often C BTC amount specified in the transaction for a particular address, where C denotes the amount of change that will be given to this address as a result of the transaction. It is not required that the address to which C is addressed is the same as the address of A, though this often happens in practice. Figure 1 illustrates the input and output relation of our transaction from A to B, and Figure 2 illustrates the steps used in constructing this transaction. Note that, in both cases, source is comprised of two transactions T_1 and T_2, and the resulting transaction is denoted as T_3.

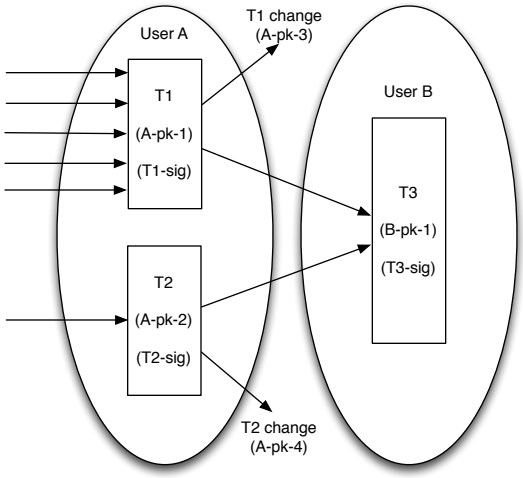


Fig. 1. Visual depiction of the input and output elements of a transaction from user A to user B.

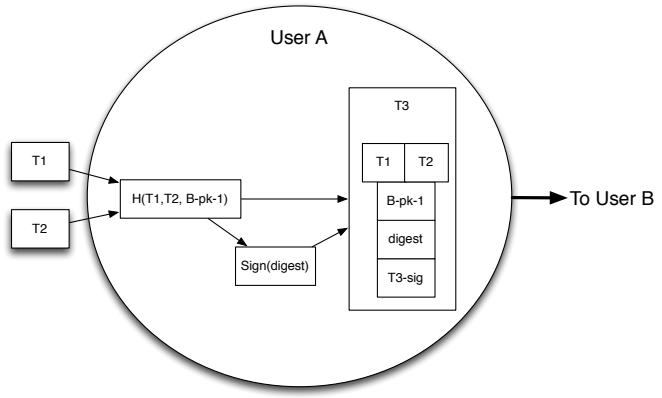


Fig. 2. Visual depction of the steps to create a transaction T_3 from user A to user B using two input transactions, T_1 and T_2.

After a transaction has been created, it is broadcasted in the network. In order to prevent double spending, nodes must confirm this transaction and append it to the chain of accepted transactions in the system's history. This procedure is based on the aforementioned proof-of-work, which works as follows. Bitcoin miners will collect unconfirmed transactions into a buffer, along with the longest chain of system-wide accepted transactions, and compute a Merkle hash of the transactions and digest of the chain. The output digest of this Merkle hash, referred to as the challenge c in the proof-of-work protocol, is then used to find the proof p. Together, c and p have the property that, when concatenated and hashed using a cryptographically strong collision-resistant hash function H, the leading B bits of the output $x = H(c||p)$ are all 0. That is, $x = \{0,1\}^B\{0,1\}^{256-B}$. Given the collision resistant properties of H, finding a valid proof p for the challenge c is comptuationally difficult. Figure 3 illustrates the construction of c and p using a previously confirmed block chain B.

Once a miner finds a proof, it is broadcasted to the other nodes in the network along with the input transactions used by the miner, who can then easily recompute the challenge c and verify the correctness of p. Once verified, this new transaction "block" is appended to the block chain which the miner used in finding the proof. Figure 4 illustrates a snippet of the block chain, where the challenge c is the digest of the previous block and the proof p are embedded in each block. Miners will continually use the longest block chain to gather and verify transaction blocks. Since there is a particular subset of BTCs

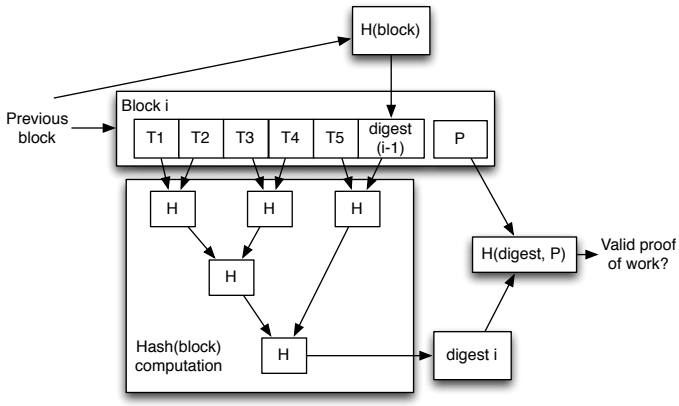


Fig. 3. Proof-of-work computational procedure using the transactions of a block, the digest of the previous block, and the sampled proof p.

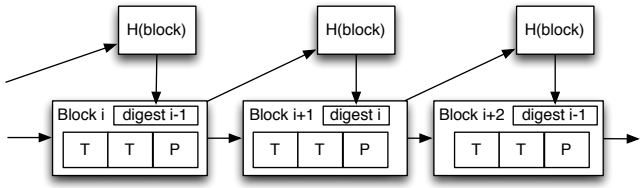


Fig. 4. A snippet of a Bitcoin transaction block chain, illustrating the groupings of transactions into a blocks, the declaration of the proof of the work p, and the digest of the previous block linking the blocks together.

in each transaction that are paid to the miner who provides the proof-of-work for a block containing that transaction, referred to as the transaction fee, miners are financially incentivized to collect more transactions into a block and continually "mine" for valid proofs-of-work.

B. Bitcoin Privacy Measures

As the topic of the survey indicates, Bitcoin has serious privacy flaws. However, there are several standard practices that Bitcoin clients and users are recommended to follow in order to improve their overall privacy and decrease the likelihood of becoming the target of privacy- or anonymity-based attacks. First, clients (and users) should specify *shadow addresses* to collect change from a transaction [**?**]. Such addresses are distinct from the user's address associated used at the time of the transaction. Furthermore, since change need not always be returned to the user who provided the BTC funds, this disjoint address obfuscates the link between the address and the original user, thus helping improve overall privacy. Secondly, it is recommended that all users maintain and continually swap their addresses, and as a result, the underlying public and private key pairs, in order to deter attacks that stem from address re-use. We discuss attacks of this nature in the following sections.

III. Definitions and Adversarial Models

Privacy refers to the inability of an adversary A to link a user U with any transactions involving U. The Bitcoin block chain links all transactions to addresses, therefore privacy is only preserved if A is unable to link U to any of the addresses involved in any transaction involving U. The inability of A to link U with any address defines the property *address unlinkability*. As shown below, analysis of privacy will be centered around address unlinkability.

In contrast to privacy is *anonymity*, which is captured and quantified with respect to *unlinkability* and *address* or *user profile indistinguishability* [5]. Activity unlinkability refers to the fact that an adversary A should not be able to link any set of transactions to any user. Given a record of all transactions, such as the Bitcoin block chain, A should not be able to identify any user. This is a much stronger security notion as it protects all users from being identified from any set of transactions. Furthermore, since transactions are linked to addresses, A should not be able to identify any user from a given set of addresses; this property is referred to as address unlinkability. Similar to address unlinkability is user profile indistinguishability. In fact, one may view it as an addition to the prior definition in that user profile indistinguishability holds if, given two addresses, an adversary A should not be able to determine if they have a common owner. Put another way, Bitcoin enjoys a measure of profile indistinguishability if an adversary is not able to group the addresses or transactions corresponding based on the original, underlying Bitcoin users.

Adversaries seeking to attack Bitcoin privacy or anonymity clearly have several distinct advantages. First, transactions are publicly broadcast throughout the network, and as users of the Bitcoin system or passive bystanders, they will therefore have access to this log. Additionally, adversaries may have access to the addresses associated with particular vendors that partake in transactions. That is, they may be able to identify and group transactions made by vendors (or other users) whose addresses are acquired via external means. Finally, for practical reasons, we also enforce that all adversaries are computationally bounded, i.e., any algorithm they may run or attack they may leverage must be carried out in polynomial time. Without this restriction it would be possible for an attacker to forge signatures, double-spend confirmed transactions by re-doing proofs-of-work, etc., among other scenarios.

IV. Deanonymization Attacks and Their Implications

In this section we survey deanonymization attacks on Bitcoin, which can roughly be characterized into the following four categories based on the source of information:

1) Collapsing multi-input transactions to the same owning entity or user,
2) Bitcoin value flow through the transaction graph (defined below),
3) Side-channel attacks that rely on timing and network-layer (e.g., IP addresses) information, and
4) Auxiliary information linkability (e.g., entity-to-address linking using information acquired via external mediums).

Our analysis is based on the small body of work primarily analyzing the first three classes of attacks. The last class is generally not discussed in scholarly work, but rather in popular media. Before discussing such attacks, we first describe the attack vectors leveraged by researchers and malicious users while forming such attacks.

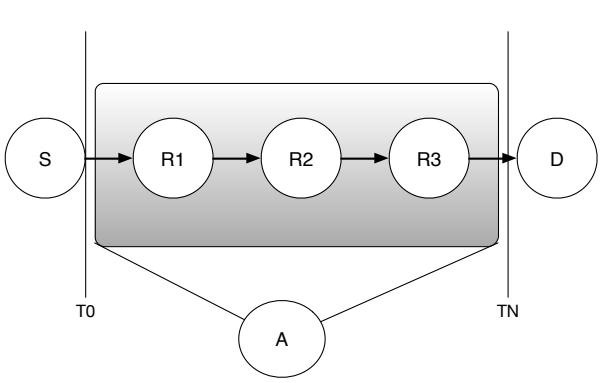
A. Attack Vectors

Given the design of the Bitcoin system, it may seem surprising that the surface for privacy- and anonymity-targeting attacks is quite large. In fact, there is a large amount of information available to attackers that may be, and has been, exploited to carry out such attacks. Perhaps most fruitful are the *transaction* and *user* graphs that can be constructed via network and transaction analysis. The transaction graph is a directed graph $\mathcal{T}$ with a vertex set $V(\mathcal{T})$ containing all transactions in the Bitcoin history and edge set $E(\mathcal{T})$ containing directed edges between the source (sender) and target (recipient) for each transaction. The user graph is yet another directed graph $\mathcal{U}$ with a vertex set $V(\mathcal{U})$ corresponding to physical users, or entities, partaking in the Bitcoin system and edge set $E(\mathcal{U})$ corresponding to the flow of Bitcoins or funds between two users. With sufficient network analysis (i.e., eavesdropping on Bitcoin traffic in the network), one may also construct a *network address* graph, which is similar to the user graph with the exception that vertices represent physical IPs instead of particular users.

B. Detailed Anonymization Attacks and Analysis Techniques

Intuitively, a successful attack on the anonymity of Bitcoin yields a mapping between Bitcoin addresses, or public keys, to their respective owners. Depending on the success criteria for such an attack, the attacker may seek to find a single mapping for a particular user or, quite oppositely, a mapping for as many users as possible. Accordingly, there has been substantial research investigating the degree to which anonymity is achieved [7], [8], [9], [6], [5]; proposed solutions presented in the literature are discussed in the following section.

Although the original Bitcoin proposal suggests that anonymity, or rather pseudonominity, is possible due to the expected difficulty of associating Bitcoin addresses with their physical owners, there have been ample works attempting to disprove this claim. In fact, one of the earliest anonymity analyses was done by Dan Kaminsky in 2011, a well-known security consultant and researcher. He assessed the anonymity claims of Bitcoin by performing the following experiment: He began by opening up a connection to a large set of peers in the Bitcoin network. Under the assumption that the first appearance of a transaction stems from the original sender (unless an anonymizing layer such as Tor is used to obscure the source), it became fairly easy to combine the peer connections with some elementary traffic analysis to derive a mapping between Bitcoin addresses and IP-layer addresses [10]. While IPs are rarely static due to DHCP and mobile devices, it is not a significant leap to suspect that such knowledge could reveal the underlying user's identity, especially if offline information is used (e.g., if the attacker is able to correlate the IP addresses to users using public forums or other applications). With this observation, a more active deanonymization attack, as carried out by Kaminsky [10], [7], would involve malicious nodes



scanning for Bitcoin clients listening to port TCP/8333 and open a direct connection. While proxy services like Tor can hide outbound connections, an inbound connection will not be obfuscated. By listening to transaction announcements over time, the client that first reports a transaction is the one that initiated it. This allows the malicious nodes to link transactions to IP addresses.

While using Tor enables one to obfuscate the source of a transaction, it does not solve the problem entirely for several reasons. First, Tor does not provide perfect source secrecy. Tor is inherently susceptible to *end-to-end correlation attacks* in which the attacker controls both ends of a Tor circuit used to send data. Second, Tor was designed for low-latency, high throughput anonymous traffic. As such, it is susceptible to side channel timing attacks [11].

Trivial attacks on privacy and anonymity involve using the Bitcoin block chain to follow all the transactions associated with that address. As user commonly have many addresses, a more sophisticated attack requires the adversary to link the known address with other hidden addresses and then analyze the transactions associated with those addresses. The two major heuristics for linking addresses are *multi-address transactions* and *shadow or change addresses*. Multi-address transactions are transactions with more than one source. Currently, Bitcoin allows for users to use more than one source address in a transaction, but does not allow multiple users to pay for one transaction. For example, suppose addr_A has 3 Bitcoins (BTC) and addr_B has 2 BTC. The user uses both addresses to pay 4 BTC to addr_C and puts the remainder of 1 BTC to addr_D. Only one user can be the input to any transaction, therefore in this example, addr_A and addr_B belong to the same user. *Shadow addresses* or *change accounts* are accounts created for change from a transaction. In the transaction above, addr_D is the shadow account that belongs to the same user that controls addr_A and addr_B. Although not directly related to the Bitcoin system, the way Bitcoin clients handle shadow accounts can break address indistinguishability [4]. However, because shadow accounts rely on user behavior instead of an inherent property of the Bitcoin system, the shadow account heuristic is not as robust [9]. Using these two heuristics, researchers have been able cluster addresses with a common owner in a user graph where every node is a user and every edge is a transaction [6], [7], [9]. In any node where the user has revealed ownership of an address, the user's anonymity has been lost.

Motivated by this observation, and relying on the accessibility of publicly accessible off-network information relating to the Bitcoin system, Reid and Harrigen [7] followed up on Kaminsky's work with an analysis of Bitcoin anonymity. To do so, they constructed the Bitcoin transaction and user graphs (see Section IV-A) as an alternative representation of the flow of information in Bitcoin, and use *passive* analysis techniques to derive user-to-address mappings.

As a currency system, Bitcoin cannot have perfect privacy nor anonymity. Although the information originates outside the Bitcoin network, some address ownership information are public knowledge. For instances, a store needs to have a publicly identifiable address in order to accept payment for goods or services. Users may also disclose address ownership when asking for donations or posting on Bitcoin forums [9]. Large centralized Bitcoin services such as the Mt. Gox exchange service are also able to associate users with addresses as part of their service.

[5]

V. Proposed Solutions

Generally speaking, solutions to address the anonymity issues in Bitcoin roughyl fall into one of the following three categories:

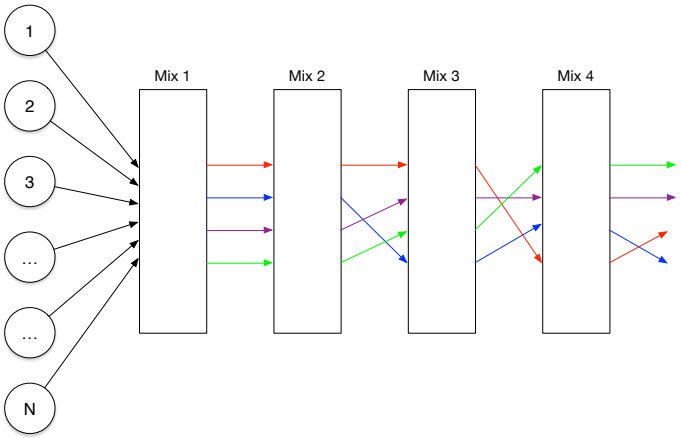
1) Use an anonymity layer to obfsucate the source IP address of Bitcoin users,
2) Use mix networks or mixing services to unlink transactions from their owners, and
3) Modify the Bitcoin protocol using certain cryptographic enhancements to improve cryptographic guarantees of anonymity

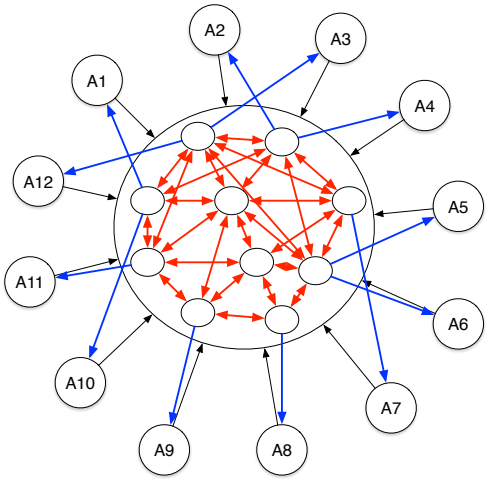
A. Anonymity Layers

The basic design and use of mix networks is credited to Chaum and dates back to more than three decades [2]. The essential idea of a mix network is illustrated in Figure V-B. Senders wrap their message in layers of encryption using the public keys of randomly selected mixing service nodes (mix node), who then receive a set of encrypted messages, decrypt and shuffle the messages, and then send them to another mix node or the intended recipient. Clearly, the role of each mix node is to hide the source of original messages by obfuscating their flow through the network.

B. Coin Mixing Services

With respect to Bitcoin, mixing services are distinct services separate from traditional mixnets. In particular, mixers are used to anonymously aggregate coins from clients, mix them internally (i.e., among several different Bitcoin addresses), and then redistribute coins back to the sources in equal denominations after some predefined amount of time. This conceptual distinction is illustrated in Figure **??**. The obvious goal is to make break, to the maximum extent possible, the link between the change address and signing address, thus improving the sender's overall anonymity. Also, it is important to emphasize the element of time with mixing services; contrary to anonymity layers like Tor, which were designed for low-latency, high throughput anonymous connections and are





therefore susceptible to timing side channel attacks, mixing services that introduce a mandatory delay prohibit (to some degree) such attacks.

Unfortunately, Bitcoin mixing services naturally require a certain level of trust on behalf of the user. In particular, there is nothing in the current Bitcoin system that prevents a malicious mixing service from aggregating a large collection of coins and then abandoning their duty as a mixing service, thereby stealing all of its user's coins. Of course, such financial losses can be minimized if the user only sends funds to the mixing services in small amounts, but it does not eliminate the problem (users would then need to sequentially small small amounts to the mixer, thus drastically increasing the payout time). Therefore, we identify mixing *theft* as a primary threat to users, as well as deanonymization, since the mixing service naturally learns the client's input transaction-signing and output change address.

Given these problems with simple mixing services in the context of Bitcoin, there has been several attempts to modify their usage in order to avoid the aforementioned threats. One work by Barber et al. [8] explored such a modification under the assumption that there does not exist an underlying trust architecture that can be relied upon to prevent the attacks;

their conclusion was that the only feasible alternative is to use a *fair exchange protocol* between the users and mixing service to ensure that all input coins are eventually paid back. Unfortunately, their proposed protocol must be integrated with the existing Bitcoin protocol - it is not natively supported. For brefity, we highlight the main features of their approach here. The fair exchange protocol between a sender and mixing service, denoted as parties A and B, respectively, requires three types of transactions:

1) Commitment transaction to commit a party to a coin exchange.
2) Refund transaction to refund a party's committed coins at a future date in case the other party aborts the protocol.
3) Claim transaction to claim the other party's committed money.

After establishing a set of shared secrets, A and B then begin the fair exchange protocol, which proceeds as follows:

1) B generates a committment and refund transaction, with the help of A, and broadcasts both transactions. The committment transaction is constructed such that its respective funds can be redeemed using either the corresponding refund transaction (after the protocol "times out", or one party fails to complete their duties within a pre-defined) or a to-be-generated claim transaction. [1]
2) Simultaneously, A generates a committment and refund transaction, also with the help of B, and broadcasts both to the network.
3) After both parties are committed to the exchange, they must claim their coins. In order to do so fairly, B first claims A's coins using A's refund transaction, and by doing so, enables A to claim B's coins through B's refund transaction. This refund process works as follows: B modifies the output of A's refund script to be redirected to B's recently generated address, and also modifies the input of the script to reveal the "secret" values that enable A to modify her refund transcation. Once B publishes his claim transaction, thus taking A's funds, A can then receive the secret values from B's claim transaction to modify B's refund transaction output to point to her own recently generated address. A then broadcasts her claim transaction to claim B's coins, thus completing the exchange.
4) If a protocol "timeout" occurs, then the refund transactions were never properly modified by both parties to steal the other party's coins, and no exchange took place.

With an existing public key infrastructure and means to securely transfer the secrets used in the transaction generation, this protocol will prevent malicious mixing services from stealing user's coins. However, it requires modifications to the underlying Bitcoin protocol that aren't readily supported. Namely, it requires that transactions support the notion of timelock (i.e., points in time when they can no longer be changed).

[1]The refund transactions are constructed with the output coins being redirected back to their original owner so that the coins are not lost in the event that the protocol does not complete.

Motivated by the desire to achieve the same level of trust without modifying the underlying Bitcoin protocol, Bonneau et el. [14] recently proposed Mixcoin, a natively-supported protocol for building accountability into mixing services. The principal idea used to achieve mixing accountability is to rely on simple economics, rather than cryptographic solutions, to ensure that mixing services will benefit more from honest participation in mixing rather than from theft. The authors are primarily interested in two main malicious mixing service threats: theft and client deanonymization. Properly aligned financial incentives ensure that theft is not likely to happen (else the mixing service's reputation is permanently destoryed). However, as the authors note, there is not (protocol-level) mechanism that can be used to gaurantee that a mixing service is not linking client input and output addresses; software implementations of mixing services may (purposefully or accidentally) store this information and thus render it susceptible to exposure if compromised.

The key to achieving accountability through financial incentives is that mixing nodes will provide warranties stating that they claim to complete a mixing operation prior to some agreed-upon deadline. If the mixing node operates dishonestly or steals the client's funds, the client then simply broadcasts the warranty to the network, which is signed by the mixing node using a long-term private key, which can then be verified by all nodes in the network using the corresponding mixing node's public key. The net effect is that no more (intelligent) clients will do business with the mixing node. Since mixing nodes are paid for their efforts through a mixing fee, with an appropriate fee selection it is therefore economically sensible to behave honestly rather than attempt to steal funds.

The fundamental protocol used to achieve this accountability operates as follows:

1) A client A generates a tuple $T = (v, \mathsf{addr}_{\mathsf{in}}, \mathsf{addr}_{\mathsf{out}}, t_1, n)$, where v is the BTC amount to send, t_1 is a time by which A will transfer the v funds, and n is a random nonce. This tuple is then sent to the mixing node M.
2) If M accepts the parameters, a fresh escrow address $\mathsf{addr}_{\mathsf{escrow}}$ is generated at which the funds can be acquired by M, a deadline t_2 by which the funds will be transferred, and a mixing fee rate p. These parameters are then signed using the long-term key K_M to generate a warranty, and both are then sent to A.
3) If A accepts t_2 and p, A transfers the funds to $\mathsf{addr}_{\mathsf{escrow}}$.
4) After receipt of the funds at $\mathsf{addr}_{\mathsf{escrow}}$, M will return the same amount of funds, minus the agreed-upon transaction fee, to $\mathsf{addr}_{\mathsf{out}}$.
5) Both parties delete the parameters used in the transaction.

Observe that a client may choose to opt out of the protocol at any point prior to the transfer of the funds by A. If M does not act honestly, A simply broadcasts the signed warranty to other nodes so that they may see A was cheated. It is also important to note that, just as in traditional mixing

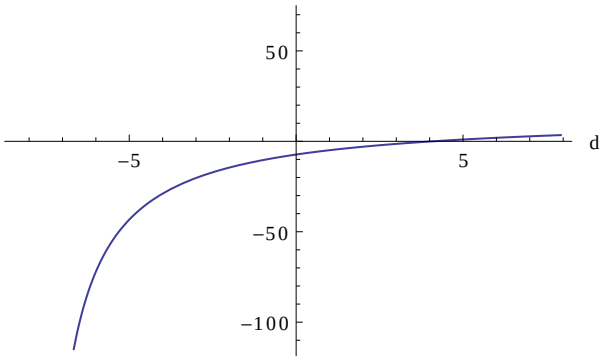
schemes, these mixing nodes may be used sequentially to increase anonymity through multiple mixing rounds. This is done by using the output address of the i-th round of mixing as the input address of the $(i+1)$-th round, thereby creating a chain through which the client's funds flow through potentially separate mixes.

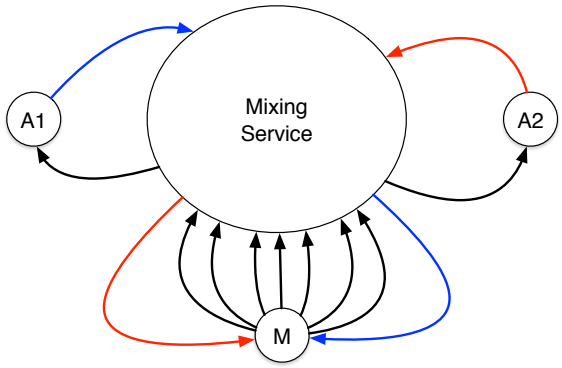
While simple, there are several underlying issues that must be addressed in practice to ensure anonymity:

- All Bitcoin chunk sizes should be uniform;
- The mixing service should choose escrow addresses at random when transferring funds from a client's input address to their specified output address;
- Transaction fees should be randomized (from an appropriate source of randomness, such as a Beacon [15] or some PRG computed based on the transaction block chain);
- There should never be two outstanding warranties for the same input address issues at the same time;
- Clients should not publicize their freshly created input addresses until mixing has initiated, else a malicious party could perform a DoS attack (due to the above requirement) by requesting a warranty for some observed or predicted input address;
- Clients should use separate mixes when aggregating their mixed funds to make a transaction, or else their anonymity set reduces to the intersection of the anonymity set for each of the mixed funds;

A brief analysis of the anonymity properties provided by Mixcoin was also presented. Of the most trivial observations are that mixing message replay is impossible due to the single-warranty issuance property stated above, and also that blocking client-to-mix or mix-to-client transactions is not feasible under the standard assumption that the majority of Bitcoin users and miners are not malicious. They also derived a value for the maximum mix delay δ_{max} (i.e., the time between receiving input funds and transferring output funds) to optimize the client's anonymity set. If a mix receives a input transactions at a stable rate where Q chunks are mixed in a single round (i.e., the total number of funds mixed is equal to $Q \times v$, where v is the baseline uniform mixing input), then the anonymity set of each client is roughly $Q(\delta_{max} - w + 1)$, since fresh input and output keys are used for each of the Q chunks for a particular user, and where w is the minimum difference between t_2 and t_1 (observe that the final mix delay, uniformly generated, thus falls in the interval $[w, \delta_{max}]$). For $98 \leq Q \leq 4096$, it was found that $\delta_{max} = 7$ since the anonymity set grows by $1 + \frac{\log_2 Q(\delta_{max}-w+1)}{1+w+\frac{\delta_{max}-w}{2}}$; see Figure **??**.

It should also be clear that the anonymity set of a particular chunk *is not* the size of the fund chunks in the mixing node. Rather, it is the size of the set of fund chunks that were mixed at the same time, i.e., those which overlap in the same round within the mixing service. For this reason, longer mixing delays or multiple mixing nodes in sequence benefit all mixing participants, and it is conjectured that these delays will converge to reasonably steady values in practice. Simulations performed by the authors showed that as the





number of overlapping mixing rounds for a set of m mixes increases, where each round transfers Q chunks, the statistical distance between the PDF of the anonymity set of each chunk rapidly approaches the uniform distribution. Put simply, this means that out of all Qr chunks (where r is the number of rounds), the probability that an adversary can correctly associate a chunk with its input client is roughly $(Qr)^{-1}$.

There is one more subtle yet important issue concerning the use of these mixing services. Recall that the input from one client A_1 may be used to provide output funds for another client A_2, and each of these inputs are associated with their providers, A_2 will learn that A_1 has interacted with the mixing service. Therefore, to deanonymize clients, an active adversary can request the mixing of many funds in hopes of probabilistically determining whether or not a particular user A has used a mixing service. This type of deanonymization attack is probabilistic because the input provided by A is only routed to the adversary's output according according to some probability distribution. Fortunately, since each use of the mixing service costs a certain fee, an attacker will, on average, have to invest a great deal of funds in hopes of learning whether or not A has interacted with the mixing service. Therefore, as previously mentioned, clients should use separate mixes whenever possible if their mixed funds (chunks) are to be used together in a single transaction. Beyond the immediate fact that her anonymity set reduces to the intersection of the anonymity set for each mixed chunk, it also reduces the likelihood that compromised or malicious mixes can deanonymize the client.

Finally, we note that there are several protocol-level side channel attacks that can be leveraged by an adversary to

decrease a client's anonymity, as follows:

- Passive observance of a client requesting a certain amount of mixed funds and then witnessing that same amount appear in a single transaction in the future effectively links the later transaction to the observed client (probabilistically).
- Bitcoin transactions (and therefore, mixed funds) are timestamped, and the provenance information of transaction block chains can therefore be used to link clients to future transactions if both the client who originally owned the unmixed funds and the time at which those funds were mixed is known. This can be avoided by paying with funds that were mixed at the same time, paying to re-mix all chunks every time a payment is to be made, and time-delayed continual mixing, among other possibilities. Observe that the client will incur additional mixing costs for the latter two solutions, which may or may not be acceptable given the level of anonymity desired, and so the client will have to intelligently choose when to mix funds at her own discretion.

C. Cryptographically-Enhanced Protocols for Anonymity on Top of Bitcoin

The underlying idea behind the Zerocoin protocol is amazingly simple and intuitive, and it stems from the fact that Bitcoin transactions are publicly linked together and therefore susceptible to passive network analysis techniques (as previously discussed above). Zerocoin breaks the links between transactions using cryptographic techniques that yield the same property of Bitcoin transaction chains (namely, the flow of coins and current value of a transaction). Ultimately, Zerocoins are not identified by public keys as in the case of the backing Bitcoin currency; rather, they are identified by a commitment C to randomly generated secrets (i.e., a serial number S associated with the coin and "opening value" r) only maintained by the owner of the coins. To do this, Zerocoin leverages accumulators, or primitives that enable a party to sign a collection of objects as opposed to a single object and accumulate the result into a single signature digest (in this case, the collection would be the set of previous transactions), and zero-knowledge proofs.

The use of these cryptographic techniques as follows. Whenever a user wants to spend Zerocoins they receive associated with some committment C, the owner must reveal S publicly and then provide a proof that they are also in possession of r capable of opening some commitment $C_i \in \{C_0, \ldots, C_{n-1}\}$, the set of all committments made in the system. The owner uses r to generate a (zero-knowledge) "signature of knowledge" π that is equivalent to stating that the owner can open some committment (coin) $C_i \in \{C_0, \ldots, C_{n-1}\}$, without revealing which coin they actually own (hence, the signature of knowledge is actually a zero-knowledge proof). Technically, to generate this proof, the owner accumulates the results of all coins used to finance the transaction into an RSA-based accumulator, produces an accumulator witness w (the accumulated total of all bitcoins in the transaction minus the value of the Zerocoins to be spent), and finally, generates π that enables other entities to *publicly* verify that the value in the accumulator was generated by the entity who derived this proof and was in possession of the Zerocoins coins (committments) necessary to fund this transaction. Zerocoin peers can then verify an accumulated value, representing virtually the same information as a Bitcoin transaction graph, using π.

From an anonymity perspective, Zerocoin ensures that given two Zerocoins and one "spend" coin, one is not able to determine, using publicly available knowledge, which coin was spent (i.e., the adversary's advantage in correctly identifying the spent coin is negligible in the security parameter of the system). With respect to the size of anonymity set, this means that $k = 2$, which is ideal. However, in practice, Zerocoin is inherently limited in that the size of the spender anonymity set can be significaly reduced under certain circumstances. For example, if a user mints and subsequently spends 5 Zerocoins, it is clear to an adversary participating in the system that all 5 coins were spent since they can simply verify the transactions; the adversary does not, however, know which coin was spent in which transaction, implying that the anonymity set cardinality is $k = 5$. Now, if the same user not mints and spends another coin, one would hope that the anonymity set would increase to $k = 6$. Clearly, this is not the case, as the attacker can easily deduce that the last minted coin was spent in the last transaction, and thus $k = 1$. Therefore, the anonymity set cardinality for a single coin is clearly bounded below by the number of coins minted between the time of the candidate coin's mint and spend, and upper bounded by the total number of minted coins in the system. Other anonymity issues of Zerocoin relate to the fact that all minted and spent coins are public knowledge, and that all transaction denominations are also publicly available. These can easily be addressed in practice, however.

The mathematical details of accumulation and witness verification are not discussed for brevity. However, we note that security of both schemes relies on the hardness of the Strong RSA and Discrete Logarithm assumptions. As such, the operations required for accumulation and verification come at the cost of additional computational overhead. Furthermore, Zerocoin requires a preliminary, potentially offline, setup phase in which the parameters for the accumulator and zero-knowledge scheme. Aware of these pitfalls, the authors propose a variety of optimizations for improving Zerocoin performance. Namely, they introduce the notion of accumulator checkpoints, which are essentially timestamps that cryptographically capture the value of an accumulator after each transaction in a recently mined block, thus removing the need for a verifier to recompute the entire accumulator value from scratch upon every invocation.

Secondly, the authors discuss methods in which the zero-knowledge proof scheme can be improved. Their preliminary experiments revealed that the size of proofs often exceeded the MTU of Bitcoin transactions, which prohibits its immediate use since Zerocoin relies on Bitcoin for the initial source of Zerocoin funds (i.e., Bitcoins are used to mint Zerocoins) and transaction block graphs to timestamp the state of the system. Rather than modify this MUT limitation so that proofs can be stored alongside accumulator checkpoints in the transaction block chain, the authors propose to store and access proofs using a distributed hash table or non block-chained storage mechanism in Bitcoin. With respect to the computational cost

of proofs, the authors propose a distribtued verification strategy so as to not make every node verify the proof of every new transaction block. Doing so would be a large computational burden on verifiers, which should be quick, as opposed to miners, whose efforts are justified via transaction fees. Internally, the chosen zero-knowledge signature of knowledge scheme consists of n repetitions of the same proof to reduce the probability of forgery to $\mathcal{O}(2^{-n})$, and so by requiring that each node *randomly* selecting a subset of these n proofs to compute and it is possible to achieve the same proof security guarantee with a significantly reduced amount of duplicated effort.

Pinocchio Coin [13] is an attempt at optimizing the Zerocoin protocol. Whereas Zerocoin uses an accumulator based on the Strong RSA assumption and proofs founded on double discrete logarithms, Pinocchio Coin uses elliptic curves and bilinear pairings to achieve virtually the same behavior with significantly less overhead. As such, Pinocchio Coin follows the same process as Zerocoin for setup, mint, spend, and verify coins: The setup phase involves creating a suitable pairing-friendly elliptic curve in the security parameter and then configuring the parameters for the Pinocchio proof system. As previously stated, minting, spending, and verification are analogous to the Zerocoin protocol with the exception that proofs are generated without the need to accumulate past commitments (though the proof generation still requires all commitments $C_0, \ldots, C_{n-1}$ as input to ensure correctness). The output of the spend procedure

The Pinocchio proof of work was originally designed as a generic proof of work to be used in applications such as cloud computing. The system takes a set of operations and converts them into proof of work that can be seen as a zero knowledge proof. The primary advantage of using Pinocchio Coin over Zerocoin is that the size of each proof is significantly smaller; Pinocchio proofs are less than 400 bytes as opposed to Zerocoin's 50kB proofs, which are well within the Bitcoin transaction size limits. One potential disadvantage of using Pinocchio Coin are that there is no security analysis as of yet [13]. Futhermore, although Pinocchio serves as an efficient general proof compiler, it is unknown if specialized systems, such as ZKPDL, would exhibit better performance in the Zerocoin framework.

CoinJoin [12] is a method to have multi-user inputs in a Bitcoin transaction. Users agree on a common output size and then combine all their transactions of that size into one big transaction. The aim of this method is to obfuscate which user is the input to an output and prevent the association of multiple addresses in a transaction to one user.

It is not be clear which user is associated to a certain output if multiple transactions with the same sized output are combined together. Any user could have paid for any output. For instance, if Alice has a 1 Bitcoin transaction to Carol and Bob has a 1 Bitcoin transaction to Dave, they can combine their transactions together to create one transaction with two outputs for 1 Bitcoin each. Now it is unclear if Alice paid Carol or Dave. With more simultaneous users, the ability to accurately track transactions would decrease.

CoinJoin allows for multiple inputs per user and combines them all into one transaction. With enough users, the transaction would hide which of the many accounts belong to which user. Although some analysis can be done on the inputs and outputs, it would not have nearly the same accuracy as the current heuristic of one user per transaction.

The simplest implementation of CoinJoin can be done with a "meet-up" server to coordinate transactions. A decentralized version can be used, but the issues with coordinating transactions cause complexity.

The most notable feature of CoinJoin, aside from privacy, is that it works today *without* any modification to the Bitocoin protocol, which is untrue for solutions akin to Zerocoin. The transactions from CoinJoin are identical to normal Bitcoin transactions. A potential development involves moving away from a centralized server that knows all the mappings to one that doesn't and eventually a de-centralized system. It is also unknown how many sessions does the protection of privacy extends. Finally, CoinJoin similar to other financial systems, CoinJoin does not hide the user's IP address; an anonymity layer such as Tor is needed to obfuscate this information from a sender's peers.

VI. Open Problems

TODO: identify issues in the bitcoin protocol that need to be addressed, and propose some solutions here

VII. Privacy-Preserving Alternatives

TODO: summary of related work and different currency systems (e.g., zerocoin) that achieve privacy

References

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Consulted* 1 (2008).

[2] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM* 24(2) (1981), 84-90.

[3] David L. Chaum. Blind Signatures for Untraceable Payments. *Crypto* 82 (1982).

[4] Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. *IEEE Symposium on Security and Privacy* (2013).

[5] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating User Privacy in Bitcoin. *IACR Cryptology ePrint Archive* 596 (2012).

[6] Dorit Ron and Adi Shamir. Quantitative Analysis of the Full Bitcoin Transaction Graph. *IACR Cryptology ePrint Archive* 584 (2012).

[7] Fergal Reid and Martin Harrigan. An Analysis of Anonymity in the Bitcoin System. *Security and Privacy in Social Networks*, Springer New York (2013), 197-223.

[8] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to Better – How to Make Bitcoin a Better Currency. *Financial Cryptography and Data Security*, Springer Berlin Heidelberg (2012), 399-414.

[9] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoyy, Geoffrey M. Voelker, and Stefan Savage. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, ACM (2013).

[10] Dan Kaminsky. Black Ops of TCP/IP Presentation. *Black Hat, Chaos Communication Camp* (2011).

[11] Tor. Bitcoin Wiki. Available online at https://en.bitcoin.it/wiki/Tor. Last accessed: 1/25/14.

[12] G. Maxwell. Coinjoin: Bitcoin Privacy for the Real World. Available online at https://bitcointalk.org/index.php?topic=279249.0. Last accessed: 1/31/14.

[13] George Danezis, Cédric Fournet, Markulf Kohlweiss, Bryan Parno. Pinocchio Coin: Building Zerocoin from a Succinct Pairing-Based Proof System. In *Proceedings of the First ACM Workshop on Language Support for Privacy-Enhancing Technologies*, ACM (2013).

[14] Joseph Bonneau, Arvind Narayan, Andrew Miller, Jeremy Clark, and Joshua A. Kroll. Mixcoin - Anonymity for Bitcoin with accountable mixes. Preprint available online at: http://cs.umd.edu/~amiller/mix.pdf.

[15] NIST Randomness Beacon. Available online at: http://www.nist.gov/itl/csd/ct/nist_beacon.cfm. Last accessed: 2/5/14.