Hiding Transaction Amounts and Balances in Bitcoin

Elli Androulaki[1] and Ghassan O. Karame[2]

[1] IBM Research Zurich, CH-8803 Rüschlikon, Switzerland
lli@zurich.ibm.com
[2] NEC Laboratories Europe, 69115 Heidelberg, Germany
ghassan.karame@neclab.eu

Abstract. Bitcoin is gaining increasing adoption and popularity nowadays. In spite of its reliance on pseudonyms, Bitcoin raises a number of privacy concerns due to the fact that all of the transactions that take place in the system are publicly announced.
The literature contains a number of proposals that aim at evaluating and enhancing user privacy in Bitcoin. To the best of our knowledge, ZeroCoin (ZC) is the first proposal which prevents the public tracing of coin expenditure in Bitcoin by leveraging zero-knowledge proofs of knowledge and one-way accumulators. While ZeroCoin hardens the traceability of coins, it does not hide the amount per transaction, nor does it prevent the leakage of the balances of Bitcoin addresses. In this paper, we propose, EZC, an extension of ZeroCoin which *(i)* enables the construction of multi-valued ZCs whose values are only known to the sender and recipient of the transaction and *(ii)* supports the expenditure of ZCs among users in the Bitcoin system, without the need to convert them back to Bitcoins. By doing so, EZC hides transaction values and address balances in Bitcoin, for those users who opt-out from exchanging their coins to BTCs. We performed a preliminary assessment of the performance of EZC; our findings suggest that EZC improves the communication overhead incurred in ZeroCoin.

1 Introduction

First introduced in 2008, Bitcoin is the most widely adopted digital currency in history. Indicatively, Bitcoin is currently integrated across a number of businesses [1] and has several exchange markets (e.g., MtGox [2], Bitstamp [3]).

Bitcoin is a Proof-of-Work (PoW) based currency which allows users to generate digital coins by performing computations. Users execute payments by digitally signing their transactions and are prevented from double-spending their coins through a distributed time-stamping service [38]. This service operates on top of the Bitcoin Peer-to-Peer (P2P) network which ensures that all transactions and their order of execution are available to all Bitcoin users. In this way, Bitcoin transactions form chains of digital signatures, which enables the public tracing of the expenditure of individual coins (BTCs).

The literature contains a number of proposals that analyze the privacy offered in Bitcoin [4, 34, 37]. ZeroCoin [32] is the first proposal to enhance the privacy in Bitcoin and has received increasing attention recently; ZeroCoin leverages zero-knowledge proofs of knowledge (ZKPoK) protocols and cryptographic accumulators in order to hide the expenditure of coins. More specifically, ZeroCoin transforms each single BTC in the system into a ZeroCoin (ZC); this ZC can be proven (using zero-knowledge techniques) to originate from a valid, and unspent BTC, but it is computationally infeasible for any adversary to trace the ZC to the corresponding BTC.

Here, each ZC corresponds to one BTC (or a predefined number of BTCs); transactions whose values are larger than the ZC's would therefore result in several back-to-back ZeroCoin transactions. This results in significant overhead in propagating the corresponding transactions in the network and including them in valid blocks. Moreover, it is easy to see that while ZeroCoin indeed prevents the traceability of coins, it does not conceal the transaction amounts; multiple ZC payments for the same transaction are likely to be linked in time and the total amount per payment can be recovered (since each ZC corresponds to a single BTC). Furthermore, ZeroCoin does not hide the total number of BTCs redeemed by Bitcoin addresses, when the owners of these addresses transform their ZCs back to BTCs. A recent study [4] has shown that tracing coin expenditure is not the only source of information leakage in Bitcoin. More specifically, Androulaki *et al.* have shown that behavior-based clustering algorithms can be used to acquire considerable information about the user profiles in Bitcoin [4]. These algorithms mainly leverage user spending patterns, such as transaction amounts, transaction times, etc., in order to profile users. Clearly, ZeroCoin does not prevent such analysis, since the transaction times, transaction amounts, and address balances can still be derived from the block chain.

In this work, we address this problem and we propose an enhanced variant of ZeroCoin, dubbed EZC, which builds upon ZeroCoin to hide the transaction amount and address balances from the network. Similarly to ZeroCoin, EZC leverages accumulators and ZKPoK protocols to construct multi-valued ZCs. The resulting coins can be either spent as regular Bitcoins, or can be spent directly in the network without transforming them back to the corresponding BTC coins. Our construct ensures that the transaction amount is never revealed to any user in the system (except for the sender and recipient). Since the coins created in EZC do not have to be exchanged back to BTCs, our scheme also prevents the leakage of the balances of address who opt-out from exchanging their coins to BTCs. We analyze the security and privacy provisions of our proposal and we show that it incurs in considerably less communication overhead when compared to ZeroCoin, given the current usage patterns of Bitcoin.

The remainder of this paper is organized as follows. In Section 2, we briefly overview the main operations in Bitcoin and Zerocoin. In Section 3, we introduce our adversarial model, and the buildings blocks that we will use in the paper. In Section 4, we introduce and analyze our extended version of ZeroCoin, EZC.

In Section 5, we overview related work in the area and we conclude the paper in Section 6.

2 Background & Problem Description

In this section, we overview the main operations in Bitcoin and ZeroCoin, respectively. We also discuss the main shortcomings of ZeroCoin.

2.1 Bitcoin

Bitcoin is a P2P payment system [38] 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*. Generally, each peer has hundreds of different Bitcoin addresses that are all stored and managed by its (digital) wallet. 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 [38]. Any peer can verify the correctness of each Bitcoin transaction 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 a hash-based proof-of-work (PoW) scheme. More specifically, to generate a block, Bitcoin peers 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, peers then include it (as well as the additional fields) in a new block thus allowing any entity to publicly verify the PoW. This process is referred to as *block mining*. Upon successfully generating a block, a peer is granted a number of BTCs. This provides an incentive for peers 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" (that is, the block contains correctly formed transactions that have not been previously spent, and has a correct PoW), 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. Bitcoin relies on this mechanism to resist double-spending attacks. In fact, for malicious users to double-spend a BTC, they would not only have to redo all the work required to compute the block where that BTC was spent, but also they would need to recompute all the subsequent blocks in the chain. Further details on Bitcoin can be found in [5, 6, 38].

2.2 ZeroCoin

ZeroCoin was introduced by Miers *et al.* in [32] to prevent the public tracing of coin expenditure in the Bitcoin network.

ZeroCoin is a cryptographic extension to Bitcoin and leverages ZKPoP protocols and cryptographic accumulators. More specifically, ZeroCoin transforms each single BTC in the system into a ZeroCoin coin, referred to in the sequel by ZC, by adding it to a cryptographic coin mixer (essentially an accumulator that is publicly available). The resulting ZCs can be proven in zero-knowledge to have originated from a valid and unspent BTC, i.e., that they are part of the unspent subset of coins in the mixer. In this way an entity is prevented from linking a transaction with the BTC (and the corresponding address) that generated the zc used therein. In other words, ZeroCoin ensures that the origin of a zc is hidden among all BTCs that were converted to zcs. In addition, ZeroCoin preserves the security guarantees of Bitcoin (e.g., the doublespending resistance). That is, no party can spend more BTCs or ZCs than the ones he/she possesses. We refer the reader to Section 3.2 for a detailed presentation of the operations and security provisions of ZeroCoin.

2.3 Problem Description

As mentioned earlier, ZeroCoin prevents the linking of a given coin spending (or transaction) to the BTC associated with it; however, ZeroCoin does not entirely prevent the possible linking of different transactions.

Recently, Androulaki *et al.* have shown that behavior-based clustering algorithms leak considerable information about user profiles in Bitcoin [4] . These algorithms mainly leverage user spending patterns, such as transaction amounts, transaction times, etc. in order to profile users and link their addresses.

ZeroCoin does not prevent such analysis, since the transaction times and transaction amounts can still be acquired from the block chain. In fact, since each ZC corresponds to exactly one BTC, any payment for a value that exceeds one BTC will incur the issuing of multiple ZC transactions. At spending time, multiple ZC transactions will be broadcasted in the network back to back in time which *(i)* can be correlated in time by a user connected to the Bitcoin P2P network in order to acquire the actual transaction amounts and *(ii)* can be used to link all the recipient addresses. Indeed, this would give the adversary $\mathcal{A}$ a considerable advantage in linking two different addresses together.

In Section 4, we propose an extension to ZeroCoin which hardens behavior-based analysis by preventing the leakage of transaction amounts. As a by-product, our proposed extension also prevents the leakage of address balances.

3 Model & Building Blocks

In existing centralized payment systems [15, 17, 22], user privacy is often measured with respect to the honest-but-curious centralized entity (e.g., Bank of

Mint) that maintains the accounts of individuals. In these systems, privacy typically means guaranteeing the payer/payee anonymity with respect to the bank. However, existing privacy-preserving solutions in this area indirectly assume that although the bank can have complete view of daily or monthly withdrawals and deposits of individuals, it is not aware of *all* transactions that take place within the system.

In an open payment system, such as Bitcoin, this model is clearly not applicable. In particular, the centralized entity is substituted by the distributed time-stamping server which is governed by the "majority of the available computation power", and has the ability to confirm or reject transactions. This distributed mechanism requires that participants check the validity of *all* transactions that occur in the system. Therefore, the privacy adversary in this case should be adjusted to account for public view of all payments, although it may not be able to link payments to individuals. For instance, in Bitcoin, a user is only aware of the pseudonym (address) of the person he/she sends a payment to/receives a payment from, but does not know other addresses that pertain to that person.

In what follows, we introduce our adversary model and we introduce the various building blocks that we use in the paper.

3.1 Threat Model and Requirements

We observe the public log of Bitcoin, denoted by pubLog, within a period of time Δt. During this period, a number of users participate in pubLog. We assume that within Δt, a total of $\mathrm{n_T}$ 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 i and S_i and R_i denote the sets of senders' addresses and recipients' addresses, respectively.

We assume that the adversary $\mathcal{A}$ is motivated to acquire information about the addresses/transactions pertaining to all or to a subset of Bitcoin users. As such, $\mathcal{A}$ does not only have access to pubLog, but is also *part of the Bitcoin system* and can participate in one or more transactions through Bitcoin. Furthermore, we assume that $\mathcal{A}$ can have access to the (public) addresses of some vendors along with (statistical) information such as the pricing of items or the number of their clients within a specified amount of time. We, however, assume that $\mathcal{A}$ is computationally bounded and as such cannot construct ill-formed Bitcoin blocks, double-spend confirmed transactions, or forge signatures, etc.

Given the aforementioned adversarial model, we identify the following security notions for Bitcoin: *balance*, *anonymity*, and *activity unlinkability*. Informally, the balance property requires that no p.p.t. adversary who has legitimately acquired a set of BTCs can spend more BTCs to other users than the ones that she possesses [32]. On the other hand, the unlinkability property refers to the fact that an adversary $\mathcal{A}$ should not be able to link two different transactions that pertain to a user of her choice. Finally, anonymity refers to the fact that the

spending of a coin should not be linked to a particular coin. We refer the reader to Section 4.3 for the definitions of the aforementioned security properties.

3.2 Building Blocks

In the sequel, we will make use of the following building blocks.

Zero Knowledge Proofs of Knowledge and Signatures of Knowledge: Our protocols use zero-knowledge proofs, i.e., protocols that can be used by a prover to prove knowledge of a committed value v, without leaking any information on the value she proves knowledge of. We instantiate such zero-knowledge proofs using the technique of Schnorr [39], and its extensions [16, 19, 24, 29], and convert them into non-interactive proofs by applying the Fiat-Shamir heuristic [28]. In the latter case, we refer to the resulting non-interactive proofs as signatures of knowledge as defined in [21].

In signatures of knowledge scheme, the knowledge of the (secret) committed value v is used as signing key. The unforgeability property of these schemes implies that no one but the party that has knowledge of v is able to provide a valid signature on any message, i.e., a signature for which the signature verification algorithm accepts.

In the following, we will use the notation of Camenisch and Stadler [11,19,21] when referring to these proofs. Namely, NIZKPoK$\{(\alpha, \beta) : h = g^{\alpha} \wedge c = g^{\beta}\}$ denotes a non-interactive zero-knowledge proof of knowledge of the elements α and β that satisfy both $h = g^{\alpha}$ and $c = g^{\beta}$. All values not enclosed in ()s are known to the verifier. Similarly, the extension ZKSoK[m]$\{(\alpha, \beta) : h = g^{\alpha} \wedge c = g^{\beta}\}$ indicates a signature of knowledge on message m.

Accumulators: Cryptographic accumulators basically constitute one-way membership functions; these functions can be used to answer a query whether a given candidate belongs to a set without revealing any meaningful information about the other set members. We make use of the accumulator by Camenisch and Lysyanskaya [18] that supports the following operations:

- $\{\mathrm{N}, \mathrm{u}\} \leftarrow \mathsf{ACC.Setup}(k)$. On input a security parameter k, sample primes p, q (with polynomial dependence on the security parameter), Setup computes the RSA modulus $\mathrm{N} = pq$, and chooses value $\mathrm{u} \in \mathrm{QR}_{\mathrm{N}}, \neq 1$. Finally, Setup outputs (N, u), which we will refer to as params.
- $\{\mathsf{Acc}\} \leftarrow \mathsf{ACC.Accumulate}(\mathrm{params}, \mathrm{PN})$. On input params and a set of prime numbers $\mathrm{PN} = \{p_1, ..., p_n | p_i \in [\mathrm{A}, \mathrm{B}]\}$, where A and B can be chosen with arbitrary polynomial dependence on k, as long as $2 < \mathrm{A}$ and $\mathrm{B} < \mathrm{A}^2$ (see [20] for more details), ACC.Accumulate computes the accumulator $\mathsf{Acc} = p_1 p_2 \cdots p_n \pmod{\mathrm{N}}$.
- $\omega \leftarrow \mathsf{ACC.GenWitness}(\mathrm{params}, \mathrm{v}, \mathrm{PN})$. On input params $= (\mathrm{N}, \mathrm{u})$, a set of prime numbers PN as described above, and a value $\mathrm{v} \in \mathrm{PN}$, the witness ω is the accumulation of all the values in Acc besides v, i.e., $\omega = \mathsf{ACC.Accumulate}(\mathrm{params}, \mathsf{Acc}|\mathrm{v})$.

– $\{0,1\} \leftarrow \mathsf{ACC.Verify}(\text{params}, \mathsf{Acc}, \text{v}, \omega)$. On input params (N, u), an element v, and witness ω, ACC.Verify computes $\mathsf{Acc}' \leftarrow \omega^{\text{v}}(\text{modN})$, and outputs 1 if and only if $\mathsf{Acc}' = \mathsf{Acc}$, v is prime, and $\text{v} \in [\text{A}, \text{B}]$.

Accumulators in [18] satisfy the strong collision-resistance property if the Strong RSA assumption is hard. Informally, this ensures that no p.p.t. adversary can produce a pair (v,ω) such that $\text{v} \notin$ PN and yet ACC.Verify is satisfied.

Camenisch and Lysyanskaya [18] describe an efficient zero-knowledge proof of knowledge which proves that a committed value is contained in an accumulator. Similar to [32], we convert this into a non-interactive proof using the Fiat-Shamir transform and refer to the resulting proof using the following notation:

$$\text{NIZKPoK}(\nu, \omega) : \text{Acc.Verify}((\text{N}, \text{u}), \mathsf{Acc}, \nu, \omega) = 1.$$

ZeroCoin Operations: Our proposal, EZC, builds upon the ZeroCoin algorithms, which we describe below.

ZeroCoin consists of the following operations: Setup, where the system parameters are set, the Mint operation, where a Bitcoin (BTC) is converted to a ZeroCoin (ZC), the Spend operation, where a ZC is spent, i.e., deposited to Bitcoin address, and (automatically) converted to a (part of) BTC, and the Verify operation, through which the peers of Bitcoin can verify the validity of the ZC transaction and include it in a block. More specifically,

– $\mathsf{params} \leftarrow \mathsf{ZC.Setup}(1^k)$, where k is the security parameter. params include a group $\mathcal{G}$ of RSA modulus and of order o, and its generators $g, h : \langle g \rangle = \langle h \rangle = \mathcal{G}$.
– $\{\text{pub}_{\text{zc}}, \text{sec}_{\text{zc}}\} \leftarrow \mathsf{ZC.Mint}(\mathsf{params}, \text{btc})$, tthrough which a BTC, btc, is converted to a ZeroCoin, zc of fixed value. The latter is associated to a secret token sec_{zc} and a public token pub_{zc}. ZC.Mint is reflected to a Bitcoin transaction (e.g., Mint transaction), where the converted btc is the transaction input and pub_{zc} that uniquely defines zc is the transaction output. Mint transactions are broadcasted in the entire network. Upon receiving the Mint transaction, the miners verify that the input btc is valid, i.e., that it is owned by the address who has signed the transaction and that it has not been spent before by that address. If btc is valid, then the Mint transaction (and thus pub_{zc}) is included in the next block using the same mining process as in Bitcoin. The public ZC-coin tokens that are included in blocks of the longest blockchain are automatically considered to be part of a public accumulator Acc.[3]. The public token related to zc, pub_{zc} is actually a Pedersen commitment to a serial number s of the form $\text{zc} = g^s \cdot h^r$, where $s, r \leftarrow_R Z_o$. The secret information associated with zc is set to $\text{sec}_{\text{zc}} = (s, r)$ and is partially revealed in ZC.Spend operation.

[3] Note that the accumulator value is computed by the peers locally. Given the public parameters of the accumulator, and the confirmed ZCs, each peer can locally compute the accumulator value at any point in time.

– $\{\pi, s\} \leftarrow \mathsf{ZC.Spend}(\mathsf{params}, \mathrm{sec}_{\mathrm{zc}}, \mathrm{pub}_{\mathrm{zc}}, \mathsf{Acc})$, which is performed by the peer who wishes to spend a ZC coin, zc, with public information $\mathrm{pub}_{\mathrm{zc}}$. To do so, the peer reveals s and computes a ZKPoK π that s corresponds to a ZC-coin which has been confirmed into a block (i.e., is part of the accumulator). As such, ZeroCoin Spend transactions can be integrated in existing Bitcoin transactions as follows: (s, π) constitute the transaction input, while the Bitcoin address(es) that will receive the spent ZC coin, will constitute the transaction output(s). This special transaction is signed using the zero-knowledge π, and is released to the network to be confirmed into a block.
– $\{\pi, s\} \leftarrow \mathsf{ZC.Verify}(\mathsf{params}, s, \pi, \mathsf{Acc})$, which can be executed by every peer in the Bitcoin network to verify that the ZC-coin associated to serial number s and signature of knowledge derived from the ZKPoK π has not been spent before, i.e., that the signature of knowledge verification accepts and that the serial number s has not been used in another confirmed spending. Verified pairs of (π, s) are included by the Bitcoin miners in the next generated block, to establish the spending of the corresponding ZC-coin to all Bitcoin peers.

4 EZC: An Extension of ZeroCoin

In what follows, we propose, EZC, an extended variant of ZeroCoin that *(i)* is not restricted to values equivalent to 1 BTC and can thus cope with any transaction amount v, and *(ii)* allows the direct spending of the v-valued ZCs into ZCs (i.e., without transforming them to BTCs first) while completely concealing the transaction amounts. Figure 1 compares the main operations of EZC to ZC.

4.1 Overview

We now start by describing the main intuition behind EZC. In the following, we refer to an EZC coin by eZC.

EZC supports the following operations: *(i)* EZC.Mint, where arbitrary valued BTCs are converted to an EZC coin (eZC)[4], *(ii)* EZC.SpendEZCToBTC, where eZCs are spent in the form of BTCs, *(iii)* EZC.SpendEZCToEZC, where the payment recipient receives her payment in the form of eZC(s), and *(iv)* EZC.SpendBTCToBTC where the payer provides BTC payments and the recipient receives its payment in the form of BTCs. Similarly to Bitcoin, the validity of each transaction is checked by the network peers, who subsequently work towards confirming valid transactions into blocks as in Bitcoin.

As in ZeroCoin, EZC generates parameters for a dynamic public accumulator $\mathrm{Acc}_{\mathrm{EZC}}$ which absorbs all properly minted and confirmed eZCs. In particular, an eZC is added in this accumulator whenever a Mint transaction is validated, i.e., included in a block, while eZCs in $\mathrm{Acc}_{\mathrm{EZC}}$ can be spent only once.

[4] This operation may also be thought of as a Bitcoin Spend operation where BTCs are spent in the form of eZC(s).

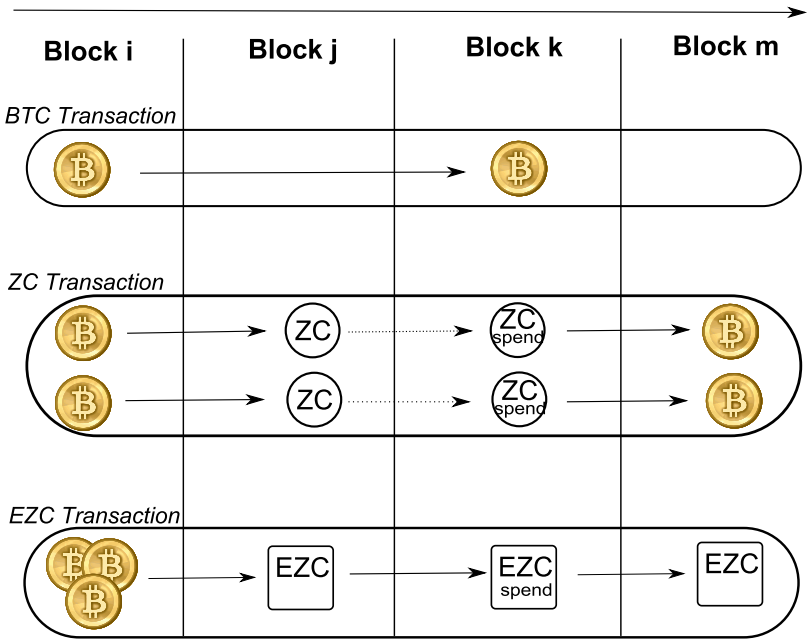

Fig. 1: Comparison between EZC and ZC. Each ZC corresponds to a single BTC and can only be spent in the form of BTCs. EZC, on the other hand, enables the construction of a (multi-valued) eZC, and can be spent in eZCs without the need to transform them back to BTCs.

In EZC, the Mint transaction is constructed in a similar way to the corresponding transaction in ZeroCoin, and thus consists of an input (in BTCs) and an output which includes information related to the created eZCs. However, the coins generated through EZC.Mint can accommodate any payment value val. More specifically, the output of a Mint transaction in EZC, consists of a commitment c to val and to a serial number ser, and a zero-knowledge proof of c's correctness. As we show later, val is revealed by the peer who runs EZC.Mint to all the peers in the network but ser is kept private until the minted eZC is spent. The Mint transaction in EZC is signed using the private keys corresponding to the input Bitcoin address(es). The correctness of EZC.Mint transactions is checked by the rest of the peers in the network and valid transactions are included in the longest block chain, in which case the EZC.Mint transactions are deemed confirmed. After the confirmation of an EZC.Mint transaction, the commitment c is considered to be a valid member of Acc_{EZC}.

To spend an eZC in the form of BTCs of value val, the eZC-owner—who knows ser and the opening of c—constructs a proof π that ser corresponds to a commitment to a value val that is a member of Acc_{EZC}. She then constructs an EZC.SpendEZCtoBTC transaction by providing a signature of knowledge of π on a conventional Bitcoin transaction output where val is assigned to one or more Bitcoin addresses. Note that for peers of the network to be able to verify

the correctness of such a transaction, the serial *ser* is revealed; nevertheless, no entity is able to link it to a particular EZC.Mint transaction, and, thus, to the btc-s that created it.

On the other hand, to spend an eZC in the form of a fresh eZC of the same or smaller value val', the payer reveals *ser*, and engages in a similar set of operations as in EZC.SpendEZCtoBTC to construct π. However, to accommodate the creation of the recipient's eZC, the two parties construct a commitment c' to a freshly generated serial ser' and to the payment amount val'. Here, π contains a proof that the payment amount does not exceed the value of the payer's coin. Finally, π is used to produce a signature of knowledge on the output commitment c' into an EZC.SpendEZCtoEZC transaction, which is released to the network with *ser*. As soon as the latter is confirmed, c' is considered by the peers to be a member of $\mathrm{Acc}_{\mathrm{EZC}}$. Here, as opposed to EZC.SpendEZCtoBTC, where *val* is revealed to the peers, *val* is kept private between the payer and the payment recipient, while ser' and the opening of c' is only known to the payment recipient.

4.2 Protocol Specification

In what follows, we detail the operations in EZC. In the following, we denote the public information (token) associated with an coin of EZC, eZC, by $\mathrm{pub}_{\mathrm{eZC}}$, and the corresponding private information (token) by $\mathrm{sec}_{\mathrm{eZC}}$.

Setup. The setup consists of operation EZC.Setup, which runs with input a security parameter λ and produces the system parameters params:

$$\{\mathrm{params}\} \leftarrow \mathsf{EZC.Setup}(1^{\lambda})$$

More specifically, EZC.Setup runs ACC.Setup(λ) to obtain (N, u), and generates primes p, q, such that $\mathrm{p} = 2^f \cdot \mathrm{q} + 1, f \geq 1$. It then picks g, h, and w such that $\mathcal{G} = \langle g \rangle = \langle h \rangle = \langle w \rangle \subset Z_{\mathrm{q}}^*$. Finally, it sets $\mathrm{params} = \{\mathrm{N}, \mathrm{u}, \mathrm{p}, \mathrm{q}, g, h, w\}$.

Coin Conversion. Coin conversion is achieved using the EZC.Mint operation, which is executed by the owner u of a set of BTCs $\mathcal{I}_{\mathrm{BTC}}$, that are converted into an eZC with public information $\mathrm{pub}_{\mathrm{eZC}}$, and private information $\mathrm{sec}_{\mathrm{eZC}}$:

$$\{\pi, \mathrm{pub}_{\mathrm{eZC}}, u(\mathrm{sec}_{\mathrm{eZC}})\} \leftarrow \mathsf{EZC.Mint}(\mathcal{I}_{\mathrm{BTC}}, \mathrm{params}).$$

Here, u picks $ser, r \leftarrow_R \mathcal{G}$, where *ser* is the serial number of the generated eZC, and computes $\mathrm{pub}_{\mathrm{eZC}} = g^{\mathrm{ser}} \cdot h^r \cdot w^{val}$, such that $\mathrm{pub}_{\mathrm{eZC}}$ is prime, and a ZKPoK π asserting that $\mathrm{pub}_{\mathrm{eZC}}$ is correctly formed:

$$\mathrm{NIZKPoK}(\alpha, \beta) : \mathrm{pub}_{\mathrm{eZC}} = g^{\alpha} \cdot h^{val} \cdot w^{\beta}.$$

Note that, the EZC.Mint transaction is constructed similarly to a standard Bitcoin transaction, where the BTCs are used as input, and $\langle \mathrm{pub}_{\mathrm{eZC}}, \pi \rangle$ is used as output. Subsequently, peers verify that $\mathrm{pub}_{\mathrm{eZC}}$ is correctly formed, by running the ZKPoK verification protocol for π, and by confirming that the input BTCs

were not spent in the past, as is currently done in Bitcoin. If the transaction is deemed valid by the majority of the computation power of the network, pub_{eZC} is included in the block chain, and pub_{eZC} is considered as a valid member of the public accumulator Acc_{EZC}. User u's private output is $\text{sec}_{\text{eZC}} = \langle \mathit{ser}, r \rangle$, while $\{\text{sec}_{\text{eZC}}, \mathit{val}\}$ is stored in u's local memory.

Spending EZC**(s) to BTCs:** This is performed using the EZC.SpendEZCToBTC operation, which takes as input $(\text{sec}_{\text{eZC}_S}, \text{pub}_{\text{eZC}_S})$, and spends them in BTCs of value val to a set of Bitcoin addresses, $\mathcal{O}_{\text{BTC}}$.

$$\mathcal{O}_{\text{BTC}} \leftarrow \mathsf{EZC.SpendToBTC}[\mathsf{params}, \mathit{ser}_S, u_S(\text{sec}_{\text{eZC}_S}, \text{pub}_{\text{eZC}_S})].$$

Here, the sender u_S first computes the public accumulator value Acc_{EZC} locally, by running $\mathsf{ACC.Accumulate}(\text{N}, \text{u}, \{\text{pub}_{\text{eZC}}\}_{\forall \in \mathsf{pubLog}})$ for the set of EZC commitments that have appeared in the output of a transaction in the longest block chain. The sender retrieves $\text{sec}_{\text{eZC}_S}$ from her local memory, and runs $\mathsf{ACC.GenWitness}(\mathsf{params}, \{\text{pub}_{\text{eZC}}\}_{\forall \in \mathsf{pubLog}}, \text{pub}_{\text{eZC}_S})$, to compute the witness w_S for $\text{pub}_{\text{eZC}_S}$'s membership in Acc_{EZC}. Furthermore, u_S computes a ZKPoK π to show that ser_S corresponds to an eZC whose public information (here, $\text{pub}_{\text{eZC}_S}$) is part of Acc_{EZC}, and that it corresponds to a value val. Finally, it converts π to a signature of knowledge on $\mathcal{O}_{\text{BTC}}$:

$$\text{ZKSoK}[\mathcal{O}_{\text{BTC}}](\alpha, \beta, \gamma) : \alpha = g^{\mathit{ser}_S} h^{\mathit{val}} w^{\beta} \wedge \mathsf{Acc.Verify}(\text{N}, \text{u}, \text{Acc}_{\text{EZC}}, \alpha, \gamma) = 1\}.$$

Finally, u_S announces the corresponding signature within a transaction to the EZC network, which, after confirming the transaction's correctness, it includes the latter into a block. [5]

Spending EZC**(s) to** EZC**(s).** This is achieved through the EZC.SpendEZCToEZC operation, which is an interactive process between a payment sender u_S and a payment recipient u_R. EZC.SpendEZCToEZC takes as input the information associated to an eZC of u_S, e.g., $\text{sec}_{\text{eZC}_S}$, and $\text{pub}_{\text{eZC}_S}$, and spends it in the form of a new eZC that belongs to u_R, eZC_R; if change should be incorporated in the payment, EZC.SpendEZCToEZC outputs additionally another eZC that would belong to u_S; we denote this eZC by eZC'_S:

$$\{\langle \text{pub}'_{\text{eZC}_S}, \text{pub}_{\text{eZC}_R}, u_S(\text{sec}'_{\text{eZC}_S}), u_R(\text{sec}_{\text{eZC}_R}) \rangle / \perp\} \leftarrow \mathsf{EZC.SpendEZCToEZC}$$
$$(\mathsf{params}, \mathit{ser}_S,\ \text{Acc}_{\text{EZC}},\ u_S(\mathit{val}_R,\ \mathit{val}_S,\ r_S,\ \mathit{ser}'_S,\ r'_S), u_R(\mathit{val}_R, \mathit{ser}_R, r_R)).$$

Here, u_R's private input $\text{sec}_{\text{eZC}R}$ consists of a serial number ser_R for her new coin, and a random number $\text{r}_\text{S} \in Z_{(p-1)/2}$ which will be used in her new coin's commitment. Assuming that $\langle \mathit{ser}_S, r_S, \mathit{val}_S \rangle$ is the entry for eZC_S in u_S's local memory, u_S announces the serial number ser_S of eZC_S, and privately contributes r_S and val_S to compute the eZC_S validity proof as in EZC.SpendEZCtoBTC.

[5] Note that, if fees are to be supported, the fee amount should be explicitly stated within the message in the signature (transaction).

Finally, u_S's private input includes the values $\langle \mathit{ser}'_S, r'_S \rangle$ used for $\text{eZC}_{S'}$'s construction. We emphasize that $\text{sec}_{\text{eZC}R}$ should be kept private even towards u_S so as the latter is not able to trace further spendings of eZC_R.

In more detail, the payment sender u_S and recipient u_R engage in the following sequence of actions:

1. u_S proves eZC_S's validity: u_S runs $\mathsf{ACC.Accumulate}(\text{N}, \text{u}, \{\text{pub}_{\text{eZC}}\}_{\forall \mathsf{pubLog}})$ for the set of eZC-commitments that appear in the EZC-blockchain, to compute the current public accumulator value Acc_{EZC}. Subsequently, u_S runs $\mathsf{ACC.GenWitness}(\mathsf{params}, \text{pub}_{\text{eZC}_S}, \text{sec}_{\text{eZC}_S}, \text{Acc}_{\text{EZC}})$, to extract a witness w_S that eZC_S has been confirmed into a block. Then, u_S computes π as described in the previous section, i.e.:

   $$\text{NIZKPoK}(\alpha, \beta, \gamma, \delta) : \alpha = g^{\mathit{ser}_S} \cdot h^{\beta} \cdot w^{\gamma} \wedge \mathsf{ACC.Verify}(\text{N}, \text{u}, \text{Acc}_{\text{EZC}}, \alpha, \delta) \ = \ 1.$$

2. u_S mints eZC'_S: u_S picks $\mathit{ser}'_S \leftarrow_R Z_{\text{p}-1}$, and $r'_S \leftarrow_R Z_{\text{p}-1}$, and computes the public information associated to eZC'_S, as $\text{pub}'_{\text{eZC}_S} = g^{\mathit{ser}'_S} h^{\mathit{val}'_S} w^{r'_S}$, such that $\text{pub}'_{\text{eZC}_S}$ is prime and $\mathit{val}'_S = \mathit{val}_S - \mathit{val}_R$ is the change value. Note that $\text{pub}'_{\text{eZC}_S}$ would be part of the transaction output. Finally, u_S updates π to include a proof that $\text{pub}'_{\text{eZC}_S}$ is properly formed:
   $\text{NIZKPoK}(\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta)$:

   $$\alpha = g^{\mathit{ser}_S} \cdot h^{\beta} \cdot w^{\gamma} \ \wedge \ \mathsf{ACC.Verify}(\text{N}, \text{u}, \ \text{Acc}_{\text{EZC}}, \ \alpha, \ \delta) \ = \ 1 \ \wedge$$

   $$\text{pub}'_{\text{eZC}_S} = g^{\epsilon} \cdot h^{\zeta} \cdot w^{\eta} \ \wedge \ \zeta \in Z_{(\text{p}-1)/2}.$$

3. u_S enables u_R to privately mint for the payment coin eZC_R. Thus, u_S picks $r_{SR} \leftarrow_R Z_{p-1}$, computes the auxiliary token $\ell_{SR} = h^{\mathit{val}_R} w^{r_{SR}}$, and updates ZKPoK π so as to include a proof of correctness of ℓ_{SR}:

   $$\text{NIZKPoK}(\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta) : \alpha = g^{\mathit{ser}_S} \cdot h^{\beta} \cdot w^{\gamma} \wedge \mathsf{ACC.Verify}(\text{N}, \text{u}, \ \text{Acc}_{\text{EZC}}, \ \alpha, \ \delta) \ = \ 1 \wedge \text{pub}'_{\text{eZC}_S} = g^{\epsilon} \cdot h^{\zeta} \cdot w^{\eta} \wedge \zeta \in Z_{(\text{p}-1)/2} \wedge \ell_{SR} \cdot \text{pub}'_{\text{eZC}_S} = g^{\epsilon} \cdot h^{\beta} \cdot w^{\theta}.$$

   Finally, u_S sends $\langle \pi, \ell_{SR}, r_{SR} \rangle$ to u_R. ℓ_{SR} is used by u_R, for the latter to prove that her privately minted eZC, is of value val_R without revealing val_R (to a third party), or the secret token of its minted eZC to u_S.
4. u_R mints eZC_R: u_R picks $\mathit{ser}_R \leftarrow_R Z_{\text{p}-1}$, and $r_R \leftarrow_R Z_{\text{p}-1}$ and computes $\text{pub}_{\text{eZC}_R} = g^{\mathit{ser}_R} \cdot h^{\mathit{val}_R} \cdot w^{r_R}$ as described previously; she extends π to include proof of correctness of $\text{pub}_{\text{eZC}_R}$ and collaborates with u_S to converts π into a ZKSoK to sign ser_S and $\text{pub}_{\text{eZC}R}$ and $\text{pub}_{\text{eZC}}{}'_S$ in the longest blockchain resulting into another transaction:

   $$\text{ZKSoK}[\mathit{ser}_S, \text{pub}'_{\text{eZC}_S}, \text{pub}_{\text{eZC}_R}](\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \mu, \rho) : \alpha = g^{\mathit{ser}_S} \cdot h^{\beta} \cdot w^{\gamma} \wedge \mathsf{ACC.Verify}(\text{N}, \text{u}, \ \text{Acc}_{\text{EZC}}, \ \alpha, \ \delta) \ = \ 1 \wedge \text{pub}'_{\text{eZC}_S} = g^{\epsilon} \cdot h^{\zeta} \cdot w^{\eta} \wedge \ell_{SR} \cdot \text{pub}'_{\text{eZC}_S} = g^{\epsilon} \cdot h^{\beta} \cdot w^{\theta} \wedge \ell_{SR} = h^{\iota} \cdot w^{\kappa} \wedge \text{pub}_{\text{eZC}_R} = g^{\mu} \cdot h^{\iota} \cdot w^{\rho} \wedge \zeta \in Z_{(p-1)/2} \wedge \iota \in Z_{(p-1)/2}.$$

The resulting transaction is announced to the network of EZC peers, who upon correct verification of its correctness, work towards its inclusion into a block. After such a transaction is included into a block, $\text{pub}_{\text{eZC}_R}$ and $\text{pub}_{\text{eZC}_S}'$ are considered members of Acc_{EZC}.

4.3 Security Analysis

In this section, we instantiate the security properties described in Section 3.1 in the context of EZC, and we show how these properties are achieved in our scheme.

Anonymity: We adopt the definition of anonymity as presented in Zero-Coin, where the spending of an eZC should not be linked to a particular Mint operation— and the BTCs—which were involved in the eZCs' creation operation with non-negligible advantage.

More formally, the EZC anonymity game takes place between an adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ as follows. Initially, the challenger runs EZC.Setup to generate the system parameters. We assume that $\mathcal{C}$ has correctly acquired two sets of BTCs, $\mathcal{I}_{\text{BTC}}^{(0)}$, and $\mathcal{I}_{\text{BTC}}^{(1)}$, which he converts through EZC.Mint into eZC_0 $(\text{pub}_{\text{eZC}_0}, \text{sec}_{\text{eZC}_0})$, and eZC_1 $(\text{pub}_{\text{eZC}_1}, \text{sec}_{\text{eZC}_1})$. Challenger $\mathcal{C}$ sends $(\text{pub}_{\text{eZC}_0}, \text{pub}_{\text{eZC}_1})$ to $\mathcal{A}$. Then, he picks $\mathscr{b} \leftarrow_R \{0,1\}$, and computes EZC.SpendEZCToEZC or EZC.SpendEZCToBTC for $\text{eZC}_{\mathscr{b}}$, outputting $\langle ser_{\mathscr{b}}, \pi_{\mathscr{b}} \rangle$, which is sent to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs her guess $\mathscr{b}'$ on $\mathscr{b}$, and wins the game if $\mathscr{b}' = \mathscr{b}$.

We say that EZC satisfies coin-anonymity property if and only if $\mathcal{A}$ has a negligible advantage over winning in the anonymity game described above.

Claim. If the underlying commitment scheme is perfectly-hiding, and the signature of knowledge π is at least computationally zero-knowledge, then EZC provides coin-anonymity.

Proof Sketch. If there is an adversary $\mathcal{A}$ who could win the coin-anonymity game with non-negligible advantage over choosing her answer at random, then such an adversary could be used by another adversary $\mathcal{S}$ to break the zero-knowledge property of the signature scheme on a commitment of a hidden value.

Transaction Unlinkability: According to this property, it should be computationally infeasible for a third party to decide whether two eZC-transactions EZC.SpendEZCToEZC (or EZC.SpendEZCToBTC) belong to the same user or not.

More formally, we construct the eZC-transaction unlinkability game in EZC, adapted from [4], as follows. $\mathcal{A}$ chooses an transaction of type EZC.SpendEZCtoEZC from pubLog denoted by τ_0. Let u_S, u_R denote the sender and recipient of τ_0. $\mathcal{C}$ picks $\mathscr{b} \leftarrow_R \{0,1\}$; if $\mathscr{b} = 0$, $\mathcal{C}$ randomly chooses $\tau_1 \neq \tau_0$ from $\text{T}_{u_S} \cup \text{T}_{u_R}$, where T_u denotes the set of transactions in which u has participated in; otherwise $\mathcal{C}$ picks τ_1 randomly from $\{\text{pubLog} - \{\text{T}_S \cup \text{T}_{u_R}\}\}$ to ensure that there are no transactions pertaining to u_S or u_R. $\mathcal{C}$ then sends τ_0, τ_1 to $\mathcal{A}$. The adversary responds with her guess on $\mathscr{b}$, $\mathscr{b}'$, and wins if and only if $\mathscr{b} = \mathscr{b}'$.

Claim. If the underlying commitment scheme is perfectly-hiding, and the signature of knowledge π is at least computationally zero-knowledge, then EZC provides eZC-transaction unlinkability.

Proof Sketch. We point out that at an eZC creation, the eZC-owner contributes random numbers. Therefore, eZCs pertaining to the same user are statistically independent. It remains to show that it is computationally infeasible for a third party that does not know the secret token of an eZC to link the spending of the eZC (EZC.SpendEZCToEZC or EZC.SpendEZCToBTC) to another EZC.SpendEZCToEZC transaction. As mentioned earlier, this follows from the security of the underlying signature of knowledge. In particular, the zero-knowledge and proof-of-knowledge properties of the underlying signature of knowledge guarantee that as long as the signature is valid, *(i)* the signer of the transaction *knows* the secret token of the spent eZC and to which public token it corresponds to, and *(ii)* that he/she leaks no information related to the parts of the tokens that remain concealed (i.e., the randomness r, the value incorporated in the coin val, and pub_{eZC}). Given that in the EZC.SpendEZCtoEZC transaction, the signature of knowledge is constructed collaboratively by the sender and recipient of the payment, the former cannot trace the spending of the payment eZC (since the sender does not obtain the secret token of the recipient eZC). We emphasize that our analysis does not account for any possible linking that can be performed by analyzing the choice of Bitcoin addresses or of the payment amount in an EZC.SpendEZCToBTC (since this transaction reveals the addresses and the amounts). On the contrary, owing to the zero-knowledge property of the signature of knowledge used within the EZC.SpendEZCtoEZC operation, it is computationally infeasible for a third party to infer the payment amount in a given EZC.SpendEZCtoEZC transaction.

Balance: The balance property requires that no user or coalition of users can spend eZCs with higher value than their balance, i.e., the value of eZCs which they possess. More formally, we consider the following game for balance. The challenger runs EZC.Setup and sets the system parameters. The adversary participates in a set of n EZC.Mint operations, by providing Bitcoin inputs $\{\mathcal{I}_{\text{BTC}}^{(i)}\}_{i=1}^{i=n}$ that were executed correctly and receive $\{\text{eZC}_i\}_{i=1}^{i=n}$ ($\{\text{pub}_{\text{eZC}_i}$, $\text{sec}_{\text{eZC}_i}\}_{i=1}^{i=n}$) in response. Let ser_i, val_i denote the serial number and value of eZC_i. The adversary $\mathcal{A}$ engages in a series of eZC-spendings resulting in a set of m eZCs $\{\text{eZC}'_i\}_{i=1}^{i=m}$ corresponding to $\{\text{pub}'_{\text{eZC}_i}, \text{sec}'_{\text{eZC}_i}\}_{i=1}^{i=n}$ with values $\{val'_i\}_{i=1}^{i=m}$. The adversary wins the game if and only if $\sum_{i=1}^{n} val_i < \sum_{i=1}^{i=m} val'_i$. Clearly, we require that no probabilistic polynomial time algorithm $\mathcal{A}$ wins the balance game with non-negligible probability.

Claim. If the underlying zero-knowledge signature of knowledge scheme is sound, and the construction of the dynamic accumulator used is secure, EZC satisfies balance of payments.

Proof Sketch. Balance in EZC is satisfied by the security properties of the dynamic accumulator and the used ZKPoK schemes. More specifically, dynamic

accumulators guarantee that no user can produce a valid witness of a coin which has not been previously added to the accumulator. This prevents the user from spending a coin which has never been confirmed. Thus, to double-spend a coin, the adversary should present the same serial number twice which would immediately reveal her intentions.

On the other hand, the security of ZKPoK protocols guarantee that if the protocol succeeds then the user indeed has knowledge of the secret values, and that the user does not overspend the value of a given eZC. More specifically, the ZKPoK used within our EZC guarantees, that in each transaction the amount of the input equals the output amount (either in BTCs or in EZCs). This is guaranteed by requiring that the amount included in the payment eZC and the eZCused for change is of a value below $\frac{(\mathrm{p}-1)}{2}$. Such a requirement avoids attacks where the sender and recipient collude and generate arbitrary payment amounts whose sum wraps around the prime modulus p. Note that, for reasonable choices of p, $\frac{(\mathrm{p}-1)}{2}$ corresponds to large payment values, which never typically occur in Bitcoin.

4.4 Performance Comparison to ZeroCoin

By enabling the construction of multi-valued ZCs, EZC clearly results in a smaller number of broadcasted transactions in the network when compared to ZeroCoin. That is, for a user to spend a transaction worth v BTCs in ZeroCoin, the user client actually broadcasts v separate Spend transactions. EZC only incurs, on the other hand, a single EZC.SpendZCtoZC transaction in order to spend the eZC whose value is worth v BTCs.

To better assess the performance gains brought by EZC when compared to ZeroCoin, we computed the average transaction amounts incurred in Bitcoin. For that purpose, we modified the block chain parser in [7] and we parsed the first 239,200 Bitcoin blocks (June 2013). Our results show that transactions in Bitcoin currently spend 9.47 BTCs on average. This means that EZC results in 9.47 times less transactions broadcast in the network, on average, when compared to ZeroCoin.

On the other hand, EZC incurs similar computational costs when compared to ZeroCoin in both the Mint and the Setup operations. Spending coins in EZC is more computationally expensive than ZeroCoin, since EZC requires an additional zero-knowledge proof that the input coins amount to the output coins. We argue, however, that this overhead is much lower than that incurred by performing ZC spendings given the current usage patterns in Bitcoin. Indeed, since current Bitcoin transactions transfer on average 9.47 BTCs, an average spending in ZC will result in the need to construct and verify almost 10 zero-knowledge proofs of knowledge.

5 Related Work

Bitcoin has received considerable attention in the literature. In [10], Elias investigates the legal aspects of privacy in Bitcoin. In [12], Babaioff *et al.* address the

lack of incentives for Bitcoin users to include recently announced transactions in a block, while in [9], Syed *et al.* propose a user-friendly technique for managing Bitcoin wallets. In [8], Karame *et al.* thoroughly investigate double-spending attacks in Bitcoin and show that double-spending fast payments in Bitcoin can be performed in spite of the measures recommended by Bitcoin developers. In [13], Welten *et al.* compile countermeasures to detect double spending attacks. In [25], Wattenhofer *et al.* connect to a subset of the Bitcoin network and measure the propagation delay of blocks.

Clark *et al.* [23] propose the use of the Bitcoin PoW to construct verifiable commitment schemes. Reid and Harrigan [36] analyze the flow Bitcoin transactions in a small part of Bitcoin log, and show that external information, i.e., publicly announced addresses, can be used to link identities and organizations to some transactions.

In [4], Androulaki *et al.* evaluate user privacy in Bitcoin and show that Bitcoin leaks considerable information about the profiles of user. In an attempt to deal with the privacy leaks in ZeroCoin, Garman *et al.* briefly describe a ZKPoK based technique which enables the construction of transactions between anonymous coins [30]. Nevertheless, the scheme of [30] is only described at high level, and their anonymous coins' transactions only refer to transactions that self-spend coins (i.e., that originate from a user to herself); in this case, the output coins are allowed to be traced by the sender. In [37], Ron and Shamir this paper 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 in order to better protect their privacy. In [34], Ober *et al.* studied the time-evolution properties of Bitcoin by analyzing its transaction graph. Finally, in [33], Moore and Christin study the economic risks that investors face due to Bitcoin exchanges.

ECash [15,17,22] 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. In [35], Pfitzmann *et al.* define unlinkability and privacy in pseudonymous systems. Dwork [27] defined differential privacy and quantified the information leakage from the query access of individuals. In [41], Shokri *et al.* quantify location privacy by assessing the error of the adversarial estimate from the ground truth. In [26, 40] the authors further introduce entropy-based metrics to assess the communication privacy in anonymous networks.

In [14], Belenkiy *et al.*, introduce an ECash-based P2P payment scheme that provides accountability at the cost of privacy. In [31], Karame *et al.* propose a novel micropayment model based on verifiable microcomputations.

6 Conclusion

In this paper, we presented EZC which builds atop Bitcoin in order to hide the transaction amounts and address balances in the system. More specifically, EZC enables the construction of a multi-valued ZeroCoin, which can be either spent as a regular Bitcoin, or can be spent directly in another ZC to ZC transaction. ESZ strengthens user privacy in the system, and minimizes information leakage that might arise from behavior based analysis.

We performed a preliminary assessment of the performance of EZC; our findings suggest that EZC considerably improves the communication overhead incurred in ZeroCoin. In the near future, we plan to implement EZC and integrate it within the Bitcoin system in order to better evaluate its performance.

Acknowledgements

The author would like to thank Srdjan Capkun for the valuable discussions. The author would also like to thank Kristiyan Haralambiev for the various discussions on ZKPoP, and Arthur Gervais for his help in parsing the Bitcoin block chain.

References

1. Trade - Bitcoin, Available from `https://en.bitcoin.it/wiki/Trade`.
2. Mt. Gox – WIkepedia, Available from `http://en.wikipedia.org/wiki/Mt._Gox`.
3. Bitcoin Charts, Available from `http://bitcoincharts.com/`.
4. Evaluating User Privacy in Bitcoin, Financial Cryptography and Data Security Conference (FC), 2013, Available from `http://eprint.iacr.org/2012/596.pdf`.
5. Protocol Rules – Bitcoin, Available from `https://en.bitcoin.it/wiki/Protocol_rules`.
6. Protocol Specifications – Bitcoin, Available from `https://en.bitcoin.it/wiki/Protocol_specification`.
7. znort987 Bitcoin Blockchain parser, Available from `https://github.com/znort987/blockparser`.
8. Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin , Available from `http://eprint.iacr.org/2012/248.pdf`.
9. Bitcoin Gateway, A Peer-to-peer Bitcoin Vault and Payment Network, 2011. Available from `http://arimaa.com/bitcoin/`.
10. Bitcoin: Tempering the Digital Ring of Gyges or Implausible Pecuniary Privacy, 2011. Available from `http://ssrn.com/abstract=1937769ordoi:10.2139/ssrn.1937769`.
11. Man Ho Au, Willy Susilo, and Yi Mu. Proof-of-Knowledge of Representation of Committed Value and Its Applications. In *ACISP*, 2010.
12. M. Babaioff, S. Dobzinski, S. Oren, and A. Zohar. On Bitcoin and Red Balloons. *CoRR*, 2011.
13. T. Bamert, C.Decker, L. Elsen, R. Wattenhofer, and S. Welten. Have a Snack, Pay with Bitcoins. In *13-th IEEE International Conference on Peer-to-Peer Computing*, 2013.
14. M. Belenkiy, M. Chase, C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. Making P2P Accountable without Losing Privacy. In *Proceedings of WPES*, 2007.

15. S. Brands. Electronic Cash on the Internet. In *Proceedings of the Symposium on the Network and Distributed System Security*, 1995.
16. Stefan Brands. Rapid Demonstration of Linear Relations Connected by Boolean Operators. In *EUROCRYPT*, 1997.
17. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact E-Cash. In *Proceedings of Advances in Cryptology - EUROCRYPT*, 2005.
18. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. pages 61–76, 2002.
19. Jan Camenisch. *Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem*. PhD thesis, ETH Zurich, 1998. ETH Series in Information Security and Cryptography.
20. Jan Camenisch and Anna Lyasyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials, 2002.
21. Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *CRYPTO*, 2006.
22. D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *Proceedings on Advances in Cryptology - CRYPTO*, 1990.
23. J. Clark and A. Essex. (Short Paper) CommitCoin: Carbon Dating Commitments with Bitcoin. In *Proceedings of Financial Cryptography and Data Security*, 2012.
24. Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.
25. C. Decker and R. Wattenhofer. Information Propagation in the Bitcoin Network. In *13-th IEEE International Conference on Peer-to-Peer Computing*, 2013.
26. C. Diaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*, April 2002.
27. C. Dwork. Differential privacy: a survey of results. In *Proceedings of the 5th international conference on Theory and applications of models of computation*, TAMC'08, 2008.
28. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. pages 186–194, 1986.
29. Amos Fiat and Adi Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *CRYPTO*, 1986.
30. Christina Garman, Matthew Green, Ian Meiers, and Aviel Rubin. Rational zero: Economic security for zerocoin with everlasting anonymity. In *Financial Cryptography and Data Security Conference*, 2014.
31. G. Karame, A. Francillon, and S. Čapkun. Pay as you Browse: Microcomputations as Micropayments in Web-based Services. In *Proceedings of WWW*, 2011.
32. Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. 2013.
33. Tyler Moore and Nicolas Christin. Beware the middleman: Empirical analysis of bitcoin-exchange risk. In *Proceedings of Financial Cryptography 2013*, April 2013.
34. Micha Ober, Stefan Katzenbeisser, and Kay Hamacher. Structure and anonymity of the bitcoin transaction graph. *Future Internet*, 5(2):237–250, 2013.
35. Andreas Pfitzmann and Marit Hansen. Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management A Consolidated Proposal for Terminology. *Fachterminologie Datenschutz und Datensicherheit*, pages 111–144, 2008.
36. F. Reid and M. Harrigan. An Analysis of Anonymity in the Bitcoin System. *CoRR*, 2011.

37. Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. Financial Crypto 2013, 2013. http://eprint.iacr.org/2012/584.pdf.
38. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.
39. C.-P. Schnorr. Efficient signature generation for smart cards. pages 239–252, 1991.
40. A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In Roger Dingledine and Paul Syverson, editors, *Proceedings of Privacy Enhancing Technologies Workshop (PET 2002)*. Springer-Verlag, LNCS 2482, April 2002.
41. R. Shokri, G. Theodorakopoulos, J. Le Boudec, and J. P. Hubaux. Quantifying location privacy. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2011.