Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges

Full Version*

Gaby G. Dagher[2] Benedikt Bünz[1] Joseph Bonneau[†1] Jeremy Clark[2] Dan Boneh[1]

[1]*Stanford University*
[2]*Concordia University*

Oct. 26, 2015

Abstract

Bitcoin exchanges function like banks, securely holding their customers' bitcoins on their behalf. Several exchanges have suffered catastrophic losses with customers permanently losing their savings. A proof of solvency demonstrates that the exchange controls sufficient reserves to settle each customer's account. We introduce Provisions, a privacy-preserving proof of solvency whereby an exchange does not have to disclose its Bitcoin addresses; total holdings or liabilities; or any information about its customers. We also propose an extension which prevents exchanges from colluding to cover for each other's losses. We have implemented Provisions and show that it offers practical computation times and proof sizes even for a large Bitcoin exchange with millions of customers.

1 Introduction

Digital currencies enable transactions that are electronically authorized, cleared and settled. After decades of research [10, 8, 2, 29] and several failed business ventures attempting to establish a digital currency, Bitcoin [27] was proposed and deployed in 2009. While still in its infancy, Bitcoin has achieved unprecedented success, enjoying a multi-billion dollar market capitalization and deployment by large retailers. Bitcoin transactions can be executed at any time by any device in the world with low (sometimes zero) fees.

Users can maintain security of their assets by managing the private keys used to control them. However, managing cryptographic keys is difficult for many users [16]. Equipment failure, lost or stolen devices, or Bitcoin-specific malware [22] could all result in the loss of one's holdings. Many users prefer to keep their holdings with online *exchanges* for a simple user experience similar to online banking—*e.g.,* with passwords, account recovery, velocity limits and customer support. Exchanges, as their name suggest, also provide conversion services between bitcoin[1] and other currencies. Customers can 'withdraw' by instructing the exchange to send the stored bitcoin to a Bitcoin address for which they manage the private key.

*An extended abstract of this work appeared at ACM CCS, 2015.

†Corresponding author, jbonneau@cs.stanford.edu.

[1]Following convention, we refer to the protocol as 'Bitcoin' and the units of currency as 'bitcoin' or ₿.

Unfortunately, storing assets with an exchange leaves users vulnerable to the exchange being hacked and losing its assets. One of the most notorious events in Bitcoin's short but storied history is the collapse and ongoing bankruptcy of the oldest and largest exchange, Mt. Gox, which lost over US$450M in customer assets. A number of other exchanges have lost their customers' Bitcoin holdings and declared bankruptcy due to external theft, internal theft, or technical mistakes [26].

While the vulnerability of an exchange to catastrophic loss can never be fully mitigated, a sensible safeguard is periodic demonstrations that an exchange controls enough bitcoins to settle all of its customers' accounts. Otherwise, an exchange which has (secretly) suffered losses can continue operating until the net withdrawal of Bitcoin exceeds their holdings. Note that while conventional banks typically implement *fractional reserve banking* in which they only retain enough assets to cover a fraction of their liabilities, the Bitcoin community is skeptical of this approach and exchanges are generally expected to be fully solvent at all times.

A rudimentary approach to demonstrating assets is simply to transfer them to a fresh public key. Mt. Gox did so once in 2011 in the face of customer skepticism, moving over ₿420k (then worth over US$7 M) in a single large transaction. However, this demonstration exposed confidential information, such as the size of Mt. Gox's business and the Bitcoin addresses they controlled. It was never repeated.

More importantly, a *proof of reserves* without a corresponding *proof of liabilities* is not sufficient to prove solvency. A proof of liabilities might consist of an audit by a trusted accountant, as done for example by Coinbase[2] and Bitstamp[3]. This might be improved by allowing users to independently verify they are in the dataset seen by the auditor, a step taken by Kraken[4] and OKCoin[5].

A cryptographic proof of liabilities, verifiable by any party with no trusted auditor, was first proposed by Maxwell [33]. However, this initial proposal leaked information about the number and size of customer accounts (see Section 2.2). These privacy issues (as well as those inherent to a simple public proof of assets) have been cited by some exchanges (*e.g.,* Kraken[6]) as a reason to use a trusted auditor instead.

In this paper we propose Provisions, a cryptographic proof of solvency scheme with the following properties:

- no information is revealed about customer holdings
- the value of the exchange's total holdings is kept secret
- the exchange maintains unlinkability from its Bitcoin address(es) through an anonymity set of arbitrary size
- multiple exchanges performing Provisions contemporaneously can prove they are not colluding

While the Maxwell proof of reserves requires only a slightly modified Merkle tree, which is a data structure well known by Bitcoin community, Provisions employs somewhat heavier cryptography not found in Bitcoin itself—*e.g.,* additively homomorphic Pedersen commitments and zero-knowledge proofs. However, we demonstrate that Provisions is efficient enough in practice even for the largest of today's exchanges to conduct a *daily* proof of solvency. The protocol is computable by a single machine in a few hours and yields proofs that are less than 20 GB.

We also consider it crucial that our protocol requires no trusted setup (or "common reference string"), consistent with the de-centralized nature of Bitcoin. More powerful cryptographic tools, such as zero-

[2] A. Antonopoulos, "Coinbase Review," *antonopoulos.com* (Blog), 25 Feb 2014.

[3] E. Spaven, "Bitstamp Passes Audit Overseen by Bitcoin Developer Mike Hearn," *CoinDesk*, 27 May 2014.

[4] N. Hajdarbegovic. "Kraken Bitcoin Exchange Passes 'Proof of Reserves' Cryptographic Audit," *CoinDesk*, 24 Mar 2014.

[5] J. Southurst, "OKCoin Reveals BTC Reserves of 104% as China's Exchanges Undergo Audits," *CoinDesk*, 22 Aug 2014.

[6] "Kraken Proof-of-Reserves Audit Process," `https://www.kraken.com/security/audit`

knowledge SNARKs [4], might offer shorter proofs or improved anonymity, but require a trusted setup which may make them less palatable to the Bitcoin community.

The need to establish the trustworthiness of Bitcoin exchanges should continue to increase as Bitcoin exchanges serve more customers and larger deposits. There are also hints that proofs of solvency will eventually become legally required in some jurisdictions: in September 2015 the U.S. Conference of State Bank Supervisors issued a proposed regulatory requirement which would require Bitcoin exchanges to demonstrate solvency, with cryptographic proofs of solvency the preferred method [13]. We believe the practicality and strong privacy guarantees of Provisions make it a good candidate to fulfill this need. We hope it will become the norm for exchanges to regularly compute a Provisions proof of solvency which might go a long way to restoring confidence in the Bitcoin ecosystem.

Limitations It is important to recognize that no proof of solvency (or any other type of audit) is future proof, as exchanges can still be hacked at any time. Likewise, proving control of a quantity of bitcoin does not guarantee the exchange itself will behave honestly in the future. It may simply abscond with all of its customers funds after completing a Provisions proof. The best we can hope for is efficient enough proofs to enable frequent and continual monitoring of the financial health of exchanges to quickly detect the loss of funds, which Provisions enables.

Provisions also requires customers to check individually that their balance has been included in the proof of liabilities. This appears to be a fundamental limitation given our privacy goals that a user's account balance is not revealed to any other party. On the positive side, as long as *some* users check and the exchange cannot predict which users will check, it runs a high risk of detection if it cheats (see Section 9.2).

Provisions is designed to prove ownership of accounts with a full public key on the blockchain. It cannot prove ownership of unused pay-to-pub-key-hash, unused pay-to-script-hash addresses, or complex multisig addresses. Removing this limitation is an interesting challenge for future work (see Section 4.2).

2 Background

We assume the reader is familiar with Bitcoin [27]. Bonneau et al. [5] provide an extensive survey of Bitcoin, although a deep understanding is not needed for understanding Provisions. The pertinent features are that each unit of bitcoin is usually redeemable by a specified public key[7] and this information is maintained in a public data structure called the blockchain.

Note that the blockchain is an ever-growing log of transactions. Any proof of solvency will be inherently bound to a single block, representing one snapshot of the Bitcoin ledger. In the remainder of the paper we leave implicit that the proof is valid for a specific block number t. It is also possible for the blockchain to fork (or "re-org") in which case an apparently-valid proof at block t may not be valid in the final block number t. As is standard with Bitcoin transactions, the defense against this is to wait until a block is *confirmed* with high probability, typically after observing that 6 followup blocks have been published.

Bitcoin public keys which hold funds are interchangeably called *accounts* or *addresses*. We note here that while we designed Provisions with Bitcoin in mind as it is the dominant cryptocurrency today, it could easily be ported to similar cryptocurrencies which have the above properties.

A proof of solvency consists of two components. In the first, the *proof of liabilities*, the exchange commits to the total quantity of bitcoin it owes to all of its users. In the second, the *proof of assets*, the

[7] Technically, bitcoins are redeemable by a specific transaction script which can encode various spending conditions, though in the vast majority of cases this is simply a public key signature and we will discuss Bitcoin as if this is the only method.

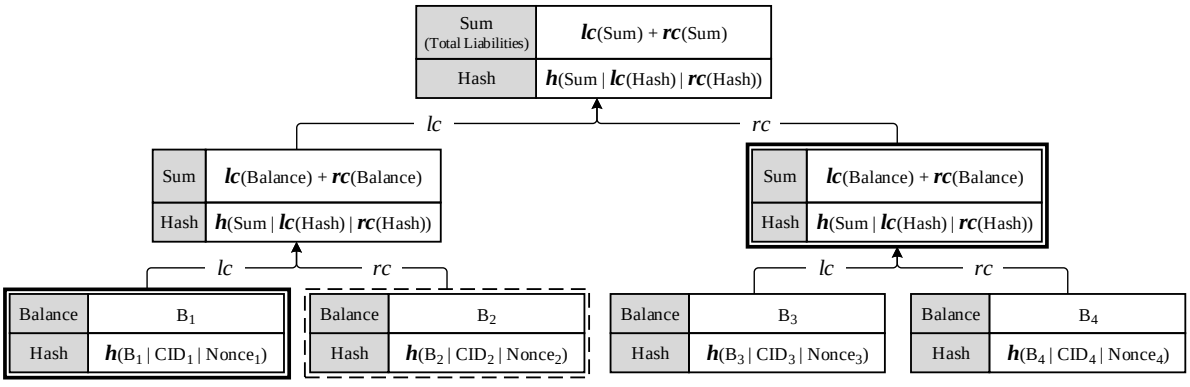

Figure 1: The Merkle tree from the Maxwell protocol [33] for proof of solvency. When a customer desires to verify their account (e.g. dashed line node), only two nodes need to be sent to the customer (bold line nodes).

exchange commits to the total value of bitcoin it has signing authority over. If the latter committed value is greater than or equal to the former, the exchange is considered solvent.

2.1 Exchange structure and holdings

Nearly all large Bitcoin exchanges operate by pooling customers' funds into a small number of large accounts. Typically for security reasons the keys for some of these accounts are kept on offline computers or in hardware security modules, requiring human action to authorize transactions (commonly called *cold storage*).

One might ask why an exchange does not simply maintain a separate Bitcoin address for each customer, enabling *direct monitoring* by each user of their funds on the public blockchain; a simple mechanism that eschews the need for a more complicated cryptographic proof of solvency. By itself, this scheme is not secure, as a malicious exchange might attempt to convince two users with the same balance that a single address is holding funds for both of them (a variation of the *clash attack* [32] discussed later).

This model also has several key practical shortcomings. First, it prevents simple division of money into hot and cold storage. Current exchanges can exist with a limited amount of money in more vulnerable hot storage because, on aggregate, the number of withdrawals in a given day is typically only a small amount of total holdings. This is similar to a large offline bank which does not carry enough cash in ATMs to cover all customer accounts, keeping substantial assets in secure (but less accessible) storage.[8]

Second, pooling assets means that transfers between customers can be efficiently settled by changing each customers' account balance without executing a transaction on the Bitcoin blockchain (incurring a transaction fee and a wait of around an hour for confirmation). Similarly, two exchanges can aggregate multiple transactions between pairs of their customers into a single settlement payment (referred to as *netting*). Minimizing reliance on the blockchain (especially for small transfers) is a key benefit of exchanges. By

[8]Executing Provisions will require computation using all of an exchange's private keys, including those for assets in cold storage. However, this can be done with human intervention at a predictable time and does not require network access to the cold storage.

contrast, maintaining a separate Bitcoin account for each customer requires "hitting the blockchain" with every transaction.

Finally, although it is not typically advertised, exchanges offer a significant privacy benefit to users as pooling funds ensures that it is not easy for outside observers to link deposits and withdrawals to the same individual [24].

Thus, we consider the pooled assets model likely to persist and we have designed Provisions to work in this model. If we combine these factors with maintaining the privacy of an exchange's addresses—proving that one owns (*i.e.,* knows) a private key without disclosing which—zero-knowledge proofs appear inescapable.

2.2 Maxwell's proof of liabilities

Maxwell proposed a protocol (summarized by Wilcox [33]) that enables an exchange to prove its total liabilities while allowing users to verify that their accounts are included in this total. The exchange constructs a binary Merkle hash tree [25] where each leaf node contains a customer's balance, as well as the hash of the balance concatenated with the customer id and a fresh nonce (*i.e.,* a hash-based commitment). Each internal node stores the aggregate balance of its left child (lc) and right child (rc), as well as the hash of its aggregate balance concatenated with the hash of its left and right children. The root node stores the aggregate of all customers' balances, representing the total liabilities, and the exchange broadcasts the root node. This is illustrated in Figure 1.

When a customer wants to verify that their balance is included in the total liabilities declared by the exchange, it is sufficient to send to the customer only part of the hash tree in order to perform the verification. Specifically, the exchange sends to the customer her nonce and the sibling node of each node on the unique path from the customer's leaf node to the root node. The other nodes on the path, including the leaf node itself, do not need to be sent to the customer because they will have sufficient information to reconstruct them. The customer eventually accepts that their balance is included iff their path terminates with the same root broadcast by the exchange.

While elegant, this protocol does not hide the value of the exchange's total liabilities which is published in the root node. While a rough sense of this value may be public knowledge, the exact value may be sensitive commercial data. Furthermore, regular proofs will reveal precise changes in the exchange's holdings.

This protocol also leaks partial information about other customers' balances. For example, if a simple balanced tree is used then each customer's proof reveals the exact balance of the sibling account in the tree (although the account holder remains anonymous). More generally, each sibling node revealed in a given users' path to the root node reveals the total holdings of each customer in that neighboring subtree. This could be mitigated somewhat by using an unbalanced tree so it is not immediately clear how many customers are in any neighboring subtree, but the protocol inherently leaks some information. Provisions removes this problem entirely, revealing no information about any users' assets beyond the fact that the total is less than the exchange's proven reserves.

2.3 Proof of assets

Once an exchange establishes its total liabilities, it must prove it owns sufficient bitcoin to match (or exceed) its liabilities. This proof of assets together with the proof of liabilities forms a proof of solvency. Maxwell's proof of assets does not preserve privacy. Instead, the exchange publicly demonstrates control of a set of addresses holding at least as much bitcoin as the exchange's total liabilities. This demonstration of control might involve moving a challenge amount of bitcoin from each account or signing a challenge message with

the private key associated with each address. Exchanges may be reluctant to do so for privacy and security concerns (revealing their internal division of funds between accounts).

In Provisions, we enable the exchange to prove ownership of an anonymous subset of addresses pulled from the blockchain. The total quantity of bitcoin across these addresses can then be determined, without being revealed, and proved to be equal or greater than the exchange's total liabilities.

2.3.1 Control vs. ownership

Any proof of assets, including Provisions, faces the inherent problem that the ability to use the signing key of an address does not necessarily imply ownership of it. A malicious exchange may collude with one or more bitcoin holders who agree to use their accounts to cover the exchange's liabilities. However, these partners may have no intention of ever making their holdings available to the exchange's customers.

An exchange might try consolidating its holdings into a single address to demonstrate that either exchange or the colluder is risking their bitcoin by placing it under the other's control. However, there is no guarantee that the single address does not implement a shared access structure by a threshold signature scheme [19].

This problem is fundamental, as no system can cryptographically prove its intentions to return something of value to a given user if requested. This customer request will be made without cryptographic authentication (*e.g.,* relying only on password authentication) because by assumption exchange customers are unwilling or unable to manage cryptographic keys. Otherwise, assets could be proved by sending each customer's bitcoins to a 1-out-of-2 multisig address redeemable by either the exchange or the user [33], providing a window for each customer to redeem their coins if desired. Again, we assume this is impractical for the majority of exchange customers.

2.3.2 Collusion attacks

Another potential vulnerability is that a cabal of two or more malicious exchanges might collude by using their own assets to participate in each other's proof of assets, making each exchange appear to control the total amount controlled by the cabal. With a public proof of assets, this would be detected if done simultaneously (because the same addresses would appear in multiple exchanges' proofs) while the transaction graph might reveal if assets are simply being moved around in a shell game.

In Provisions, because the exchange's addresses are kept confidential, detection of this attack becomes more challenging. However, in Section 7 we show an extension to the basic Provisions protocol which enables exchanges to prove that they are not using the same assets as other exchanges running the protocol. To do so, they publish an additional value which is unlinkable to their real Bitcoin address, yet is a deterministic function of (and requires knowledge of) their private key. Thus, if two exchanges attempt to use the same bitcoin address in two different solvency proofs, the re-use can be detected.

This extension imposes a small performance cost (see Section 10.4) and a small impact on the exchange's privacy as it reveals the number of addresses to which the exchange knows the private key (see Section 9.1). Thus we leave it as an extension for now, as it will only become beneficial when multiple exchanges are implementing Provisions and are willing to synchronize their proofs.

3 Protocol overview

The objective of Provisions is to enable an exchange $\mathcal{E}$ to publicly prove that it owns enough bitcoin to cover all its customers' balances such that (1) all customer accounts remain fully confidential, (2) no account

contains a negative balance, (3) the exchange does not reveal its total liabilities or total assets, and (4) the exchange does not reveal its Bitcoin addresses. Provisions consists of three main protocols:

Protocol 1 - Proof of assets. In this protocol, the exchange compiles a large anonymity set of public keys $\mathbf{PK}$ corresponding to addresses on the Bitcoin blockchain. The exchange possesses the private keys to a subset of the public keys in $\mathbf{PK}$. Next, the exchange creates a commitment to its total assets and proves in zero-knowledge that the sum of balances held by the public keys it owns (i.e. public keys for which it knows the secret key) is equal to the committed value. This is done without revealing which public keys it owns.

Protocol 2 - Proof of liabilities. In this protocol, the exchange publishes a commitment to each user's account balance. These committed values are summed homomorphically to produce a commitment to the exchange's total liabilities. The exchange enables each user to privately verify that the commitment to their balance is correct. It also proves that every committed balance is a small positive integer.

Protocol 3 - Proof of solvency. Using the commitments to its total assets and liabilities produced by the above two protocols, the exchange will homomorphically compute a commitment to their difference and prove in zero-knowledge that this final commitment is a commitment to zero. This will prove that the total liabilities is exactly equal to the total assets (or, via a minor modification, that it is strictly less than the total assets).

3.1 Preliminaries & notation

Public parameters. We let g and h be fixed public generators of a group G of prime order q. Our implementation uses the elliptic curve secp256k1 [9] as the group G; this is the group used for Bitcoin ECDSA signatures. Note that this allows us to work with existing Bitcoin public and private keys, although we do not actually perform any ECDSA signatures. While implemented over elliptic curves, we use the more conventional multiplicative notation (*e.g.,* $y = g^x$ instead of $Y = xG$).

We stress that Provisions requires no trusted setup, so that no party (or group of parties) can cause the system to malfunction.

Bitcoin balance lookups. We assume that the Bitcoin blockchain is universally agreed upon and all parties can use it to compute the quantity of bitcoin owned by each address. More precisely, for a Bitcoin public key $y \in G$ we use $\text{bal}(y)$ to denote the balance associated with y. We assume $\text{bal}(y)$ is an integer between 0 and MaxBTC for all y. We can represent any bitcoin account with $\text{MaxBTC} = 2^{51}$—the rules of Bitcoin limit the total currency supply to 21M ₿, each divisible into a maximum of 10^{-8} atomic units called *satoshis*. Note that satoshis are the true units of currency in Bitcoin. Setting $₿1 = 10^8$ satoshis is a convention to provide a more human-friendly accounting unit. In the remainder of this paper when we speak of account balances we will always be working with satoshis.

Pedersen Commitments. Provisions makes heavy use of Pedersen commitments [30]. The commitment to a message $m \in \mathbb{Z}_q$ is defined as $\text{com} = g^m \cdot h^r$ where g and h are fixed public elements of G and the quantity r is chosen at random in $\mathbb{Z}_q$. The generators g and h are chosen once in a way that ensures no one knows their relative discrete logarithm. Specifically, we use the standard g from secp256k1 and derive h deterministically by hashing the string `Provisions`. Recall that Pedersen commitments are perfectly hiding so that com reveals no information about m.

Non-Interactive Zero-Knowledge Proofs (NIZKP). Provisions requires a number of non-interactive zero-knowledge proofs. In all cases, these can be adapted from basic Σ-protocols such as the Schnorr proof of knowledge of a discrete logarithm [31] or the Chaum-Pedersen proof of representation of a Diffie-Hellman tuple [12], using Fiat-Shamir [17] to compile into a non-interactive zero-knowledge protocol (NIZKP). If one wishes to avoid the random oracle model, any alternative Σ-protocol to NIZKP compilation [20] is sufficient.

4 Proof of assets

We begin with Protocol 1 which lets the exchange $\mathcal{E}$ generate a commitment to its total assets along with a zero-knowledge proof that the exchange knows the private keys for a set of Bitcoin addresses whose total value is equal to the committed value.

The exchange $\mathcal{E}$ assembles a set of Bitcoin public keys

$$\mathbf{PK} = \{y_1, \ldots, y_n\} \subseteq G$$

corresponding to addresses on the blockchain that will serve as an anonymity set. Note that these must be complete public keys and not Bitcoin addresses, which are typically hashes of public keys. We discuss this limitation, as well as how to choose this set, in Section 10. We let $x_1, \ldots, x_n \in \mathbb{Z}_q$ be the corresponding secret keys so that $y_i = g^{x_i}$ for $i = 1, \ldots, n$.

Let $\mathbf{S}$ be the exchange's own set of Bitcoin addresses for which it knows the private keys. The anonymity set $\mathbf{PK}$ must of course be a superset of the exchange's own Bitcoin addresses so that $\mathbf{S} \subseteq \mathbf{PK}$.

We use the booleans $s_i \in \{0, 1\}$ to indicate which accounts the exchange controls in $\mathbf{PK}$. We set $s_i = 1$ whenever the exchange knows the private key x_i for Bitcoin public key $y_i \in \mathbf{PK}$. The exchange's total assets can then be expressed as

$$\mathsf{Assets} = \sum_{i=1}^{n} s_i \cdot \mathrm{bal}(y_i)$$

We assume $\mathrm{bal}(y_i) > 0$ for all i. Finally, we define

$$b_i := g^{\mathrm{bal}(y_i)} \quad \text{for } i = 1, \ldots, n.$$

Given the set $\mathbf{PK}$, a verifier can easily compute all the b_i for itself using information in the Bitcoin blockchain.

4.1 Proof of assets Σ-Protocol

The exchange publishes Pedersen commitments to each $s_i \cdot bal(y_i)$ for $i \in [1, n]$ by choosing a random $v_i \in \mathbb{Z}_q$ and computing

$$p_i = b_i^{s_i} \cdot h^{v_i} \ . \tag{1}$$

A homomorphic addition of these commitments yields a Pedersen commitment Z_{Assets} to Assets:

$$Z_{\mathsf{Assets}} := \prod_{i=1}^{n} p_i = \prod_{i=1}^{n} b_i^{s_i} \cdot h^{v_i} = g^{\mathsf{Assets}} h^{(\sum_{i=1}^{n} v_i)} \ . \tag{2}$$

It remains to prove in zero-knowledge that Z_{Assets} is valid. That is, for all $i \in [1, n]$ we have $s_i \in \{0, 1\}$ and when $s_i = 1$ the exchange knows the secret key $x_i \in \mathbb{Z}_q$ for the public key y_i. To prove this, $\mathcal{E}$ publishes a few additional values. For each $i \in [1, n]$ the exchange chooses a random $t_i \in \mathbb{Z}_q$ and publishes

$$l_i = y_i^{s_i} h^{t_i} \ \in G \tag{3}$$

which is a Pedersen commitment for s_i. Equivalently, these l_i can be written as

$$l_i = g^{x_i \cdot s_i} h^{t_i} \tag{4}$$

Public data from blockchain: (y_i, b_i) for $i \in [1, n]$ and g, h
Verifier's input from prover: (p_i, l_i) for $i \in [1, n]$
Prover's input: $s_i \in \{0, 1\}$, $v_i, t_i, \hat{x}_i \in \mathbb{Z}_q$ for $i \in [1, n]$
Protocol:

1. For $i \in [1, n]$

   (a) Prover ($\mathcal{E}$) chooses $u_i^{(1)}, u_i^{(2)}, u_i^{(3)}, u_i^{(4)} \xleftarrow{\$} \mathbb{Z}_q$.

   (b) The prover sends to the verifier:

$$a_i^{(1)} = b_i^{u_i^{(1)}} h^{u_i^{(2)}}, \qquad a_i^{(2)} = y_i^{u_i^{(1)}} h^{u_i^{(3)}}, \qquad a_i^{(3)} = g^{u_i^{(4)}} h^{u_i^{(3)}}$$

   (c) The verifier replies with a challenge $c_i \xleftarrow{\$} Z_q$

   (d) Prover ($\mathcal{E}$) replies with:

$$\begin{aligned} r_{s_i} &= u_i^{(1)} + c_i \cdot s_i, && \text{(response for } s_i\text{)} \\ r_{v_i} &= u_i^{(2)} + c_i \cdot v_i, && \text{(response for } v_i\text{)} \\ r_{t_i} &= u_i^{(3)} + c_i \cdot t_i, && \text{(response for } t_i\text{)} \\ r_{\hat{x}_i} &= u_i^{(4)} + c_i \cdot \hat{x}_i, && \text{(response for } \hat{x}_i\text{)} \end{aligned}$$

   (e) The verifier accepts if:

$$\begin{aligned} b_i^{r_{s_i}} h^{r_{v_i}} &\stackrel{?}{=} p_i^{c_i} a_i^{(1)} && \big(\text{verify statement (1)}\big) \\ y_i^{r_{s_i}} h^{r_{t_i}} &\stackrel{?}{=} l_i^{c_i} a_i^{(2)} && \big(\text{verify statement (3)}\big) \\ g^{r_{\hat{x}_i}} h^{r_{t_i}} &\stackrel{?}{=} l_i^{c_i} a_i^{(3)} && \big(\text{verify statement (5)}\big) \end{aligned}$$

   (f) Run the protocol in Appendix B to prove knowledge of $s_i' \in \{0, 1\}$ and $v_i' \in \mathbb{Z}_q$ satisfying (1). [the binding property of Pedersen commitments ensures that $s_i = s_i'$ and $v_i = v_i'$]

2. The verifier computes $Z_{\mathsf{Assets}} := \prod_{i=1}^{n} p_i$ as in (2)

Protocol 1: Privacy-preserving proof of assets

which is a Pedersen commitment to the quantity $x_i \cdot s_i \in \mathbb{Z}_q$. By setting $\hat{x}_i := x_i \cdot s_i$, we can re-write (4) as

$$l_i = g^{\hat{x}_i} h^{t_i} \tag{5}$$

Now, to prove that Z_{Assets} computed in (2) is a commitment to (a lower bound of) the exchange's assets, the exchange proves knowledge of quantities:

$$s_i \in \{0, 1\} \quad \text{and} \quad v_i, t_i, \hat{x}_i \in \mathbb{Z}_q \quad \text{for } i \in [1, n] \tag{6}$$

satisfying conditions (1), (3), and (5). This convinces the verifier that when $s_i = 1$ the exchange knows the private key $x_i \in \mathbb{Z}_q$ for the public key y_i. To see why, observe that dividing equation (3) by (5) proves that when $s_i = 1$ the exchange knows $\hat{x}_i \in \mathbb{Z}_q$ such that $g^{\hat{x}_i} = y_i$, as required.

The exchange proves knowledge of the required values in (6) using Protocol 1. The protocol makes use of a standard Σ-protocol to prove that each s_i is binary and known to the exchange. The protocol is presented in Appendix B for completeness.

Protocol 1 can be made non-interactive using the Fiat-Shamir heuristic. It therefore suffices to prove that the protocol is honest-verifier zero-knowledge, as shown in the following theorem:

Theorem 1. *For public values g, h and (y_i, b_i, p_i, l_i) for $i \in [1, n]$, the Σ-protocol in Protocol 1 is an honest-verifier zero-knowledge argument of knowledge of quantities*

$$s_i \in \{0, 1\}, \;\; v_i, t_i, \hat{x}_i \in \mathbb{Z}_q \;\; \textit{for } i \in [1, n]$$

satisfying conditions (1)*,* (3) *and* (5) *for all $i \in [1, n]$.*

The proof of Theorem 1 is given in Appendix C. As explained above, this proof of knowledge convinces the verifier that Z_{Assets}, as computed in (2), is a commitment to (a lower bound of) the exchange's total assets. The proof is made non-interactive using the Fiat-Shamir heuristic. Since it is honest-verifier zero-knowledge, it reveals nothing about the total assets, the s_i, or the x_i, as required.

Proof length. The proof size is linear in the size of the anonymity set n. We use the standard Schnorr optimization so that, for each $i \in [1, n]$, the proof contains only two elements in G, namely (p_i, l_i), and eight elements in $\mathbb{Z}_q$, namely the five elements $(c_i, r_{s_i}, r_{t_i}, r_{\hat{x}_i}, r_{v_i})$ plus three additional elements in $\mathbb{Z}_q$ from the protocol in Appendix B. This is feasible even for large anonymity sets. We discuss practical parameters in Section 10.

We can shrink the proof size by $\sim 10\%$ if we choose a single challenge value c for all $i \in [1, n]$. However, we choose a different c_i for each $i \in [1, n]$ so that the verifier can check the proof piecemeal, one i at a time.

4.2 Supported address types

Provisions does not support all address types which are possible in Bitcoin; it only supports addresses which correspond to a single public key. This includes all pay-to-pubkey-hash (P2PKH) and some pay-to-script-hash (P2SH) address, though for these addresses the public key itself will not be revealed until after the address has first been used.[9] Thus, only addresses which have already been used at least once can be utilized as part of the anonymity set.

[9] An exception is pay-to-pubkey transactions, which specify the public key upfront. Unfortunately (from the point of view of Provisions), these transactions have been deprecated and are now rare.

Bitcoin also supports more complicated address types. Each transaction output is technically represented as a small script in a custom Bitcoin script language which can specify arbitrary conditions to redeem the funds. Provisions can only prove the ability to spend from a script that specifies a single public key. The most common case besides a single key are multisig addresses, which specify n keys of which k must sign. Extending Provisions to support multisig addresses is an interesting challenge for future work.

Using a more powerful proof system such as zk-SNARKs [4], it would be possible to demonstrate the ability to satisfy an arbitrary script by compiling a Bitcoin script interpreter circuit. However, this would require a large and complex circuit which would be wasteful given that nearly all Bitcoin addresses conform to one of a very small number of script types. A better design would likely be to compile specific SNARK circuits corresponding to common script types, such as pay-to-pubkey-hash or multisig. Note that SNARKs, which support proofs on any NP statement, can in principle prove a statement of the form "Either I know a public key whose hash is h and I know the private key corresponding to this public key in which case a is a commitment to the balance of the address h, or I don't necessarily know the preimage of h and a is a commitment to th value 0." This would enable a proof of reserves including pay-to-pubkey-hash addresses which have not yet been spent. Implementing such a statement efficiently using zk-SNARKs is an important future research direction.

5 Proof of liabilities

Protocol 2 enables the exchange $\mathcal{E}$ to verifiably commit to its total liabilities and convince all clients that their balances were included in the commitment.

To provide some intuition behind the design of Protocol 2, consider the mapping of real customers to entries on LiabList. Each real customer should have an entry in LiabList (*i.e.,* the mapping is a function) and no distinct customers should be given the same entry (*i.e.,* the mapping should be injective). If two users have the same balance, a malicious $\mathcal{E}$ might try to point both users to the same entry—in the voting literature, this is called a clash attack [32]. To ensure an injective mapping, customers are provided an ID in line (1d) which commits[10] to unique information about the customer $\mathsf{username}_i$ (which may include their username, email address, and/or account number). The commitment is binding, preventing the exchange from opening a CID to distinct data for different users. It is also hiding, preventing an adversary who knows the $\mathsf{username}_i$ of a potential customer from determining if that customer is in LiabList (or if a user is known to be a customer, which CID they correspond to).

We do not require the mapping to be surjective—$\mathcal{E}$ can always add fake users to the list, but we ensure that doing so can only increase $\mathcal{E}$'s apparent liabilities. It might be in $\mathcal{E}$'s interest to include fake users with a zero (or tiny) balance to obscure the total number of customers it truly has. However, we must ensure that any included users (real or fake) can only add to the exchange's total liabilities. That is, $\mathcal{E}$ should not be able include a negative balance to try to decrease its apparent liabilities. Since negative numbers do not technically exist in modular arithmetic, the precise requirement is that when added together, the sum will never exceed q, the order of the group G, which is $q \approx 2^{256}$ for our group $G =$secp256k1.

To enforce this, $\mathcal{E}$ provides a range proof (adapted from [23]) for each committed balance showing it is from a 'small' interval between 0 and $\mathsf{MaxBTC} = 2^{51}$. This ensures that an overflow will never occur as long as the exchange has fewer than 2^{205} accounts. The range proof works by proving that the account balance is at most 51 bits long by committing to the account balance bit-by-bit and proving that each bit is a 0 or 1.

[10] Unlike the other commitments used in Provisions, the commitment scheme used to produce CID_i need only be binding and hiding, not additively homomorphic. We use a simpler hash-based commitment scheme instead of Pedersen commitments.

To verifiably commit to its liabilities, $\mathcal{E}$ does:

1. For each customer $\mathcal{C}_i : 1 \leqslant i \leqslant c$:
   (a) Represent each $\mathcal{C}_i$'s balance $\mathsf{Balance}_i$ as an m-bit binary number (where $m = \lceil \lg_2 \mathsf{MaxBTC} \rceil$):
   $$\mathsf{BinBalance}_i = \langle x_{i,0}, x_{i,1}, \ldots, x_{i,m-1} \rangle, \qquad (\text{then } \mathsf{Balance}_i = \textstyle\sum_{k=0}^{m-1} x_{i,k} \cdot 2^k)$$
   (b) Compute and publish a Pedersen commitment to each $x_{i,k}$ in the group G using generators g and h:
   $$z_{i,k} = g^{x_{i,k}} h^{r_{i,k}}, \quad r_{i,k} \xleftarrow{\$} \mathbb{Z}_q$$
   (c) For every $i \in [1, c]$ construct a proof of knowledge π_i for the statement:
   for all $k \in [0, m-1]$, the exchange $\mathcal{E}$ knows $r_{i,k} \in \mathbb{Z}_q$ and $x_{i,k} \in \{0, 1\}$ such that $z_{i,k} = g^{x_{i,k}} h^{r_{i,k}}$.

   Notice that $z_i := \prod_{k=0}^{m-1} (z_{i,k})^{(2^k)}$ is a Pedersen commitment to $\mathcal{C}_i$'s balance.
   Moreover, $Z_{\mathsf{Liabilities}} := \prod_{i=1}^{c} z_i$ is a Pedersen commitment to the sum of all balances, which is $\mathcal{E}$'s total liabilities.

   (d) Compute a fresh customer identifier CID_i by committing $\mathcal{C}_i$'s username:
   choose a nonce $n_i \xleftarrow{\$} \{0, 1\}^{512}$ and compute
   $$\mathsf{CID}_i := \mathsf{H}(\mathsf{username}_i \| n_i)$$
   where H is a collision-resistant hash function such as SHA-256
2. Publish the following list of liabilities $\mathsf{LiabList}$ of all customers' tuples:
   $$\mathsf{LiabList} = \langle \mathsf{CID}_i,\ z_{i,0}, \ldots, z_{i,m-1},\ \pi_i \rangle \quad \text{for } i = 1, \ldots, c.$$

Every client $\mathcal{C}_i$, for $i \in [1, c]$, can verify that its balance is uniquely included in $\mathsf{LiabList}$ as follows:

1. Client $\mathcal{C}_i$ logs in and is privately given (r_i, n_i).
2. The client recomputes $\mathsf{CID}_i = \mathsf{H}(\mathsf{username}_i \| n_i)$ and verifies that is correctly included in $\mathsf{LiabList}$.
3. The client verifies its own balance as follows:
   compute $z_i := \prod_{k=0}^{m-1} (z_{i,k})^{(2^k)}$ and verify that $z_i = g^{\mathsf{balance}_i} h^{r_i}$.
4. The client validates integrity of the remaining entries in $\mathsf{LiabList}$ by checking the proof π_i for all $i = 1, \ldots, c$.
5. Finally, Client $\mathcal{C}_i$ computes and outputs $Z_{\mathsf{Liabilities}} := \prod_{i=1}^{c} z_i$.

In practice, the client $\mathcal{C}_i$ need only do Steps (1)–(3). Steps (4) and (5) can be carried out by a public auditor (see Section 5.1).

Protocol 2: Privacy-preserving proof of liabilities

This committed binary representation is homomorphically converted into an integer and homomorphically summed.

The range proof in Step (1c) is the bulk of the proof size (see Section 10). We could have used more efficient range proofs [6, 7], but making these protocols non-interactive using the Fiat-Shamir heuristic requires that we use a trusted setup.[11] Our goal is to ensure that Provisions requires no trusted setup, necessitating a range proof that require no trusted setup.

Another alternative is to use zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) [4]. The proof would be significantly shorter (constant in the number of users) at the expense of a large common reference string, the use of heavier cryptographic tools and a trusted setup. Realizing an efficient proof of solvency using zk-SNARKs is an interesting challenge for future work.

The following theorem summarizes the security properties of Protocol 2. We do not provide a formal proof here; the proof follows from the security of the Chaum-Pedersen proof of knowledge of a representation and the security properties of commitments built from collision-resistant hash functions.

Theorem 2.

- *Protocol 2 is zero-knowledge in the following sense: there is an efficient simulator (in the random oracle model) that can simulate all the entries in* LiabList *given only the number of users (but not their balances or usernames).*
- *Protocol 2 is sound in the following sense: if a public auditor successfully executes verification steps (4) and (5) then $\mathcal{E}$ proved knowledge of a pair $(r,\ \mathsf{Liabilities})$ in $\mathbb{Z}_q^2$ such that*

$$Z_{\mathsf{Liabilities}} = g^{\mathsf{Liabilities}} h^r$$

where Liabilities *is equal as an integer to $(\sum_{i=1}^{c} a_i)$ for some a_i in the range $[0, \mathsf{MaxBTC}]$. Moreover, for every client $\mathcal{C}_i$ that successfully executes verification step (3) we have that $a_i = \mathsf{balance}_i$.*

5.1 Customer verification

We assume that customers each check LiabList to verify the presence of their identifier CID_i and the correctness of their committed balance z_i. A malicious $\mathcal{E}$ which omits some customers will be detected if at least one of those customers checks the proof of liabilities. This is an inherent limitation given our privacy goals which require that only customers can tell if their balance has been included or not. This limitation applies equally, for example, to Maxwell's protocol. We discuss this further in Section 9.2. Fortunately the checks required of each individual are quite lightweight. Each customer $\mathcal{C}_i$ receives from $\mathcal{E}$ their $\mathsf{username}_i$, r_i and n_i. They then locate in LiabList, with a hint from $\mathcal{E}$, their tuple:

$$\langle \mathsf{CID}_i, z_{i,0}, \ldots, z_{i,m-1}, \pi_i \rangle$$

Using n_i, they can open their commitment CID_i and verify that it commits to $\mathsf{username}_i$. Next, using r_i the customer checks that z_i is indeed a commitment to their true account balance $\mathsf{Balance}_i$. This is shown in Step (3) and is a simple calculation.

The other two verification steps, (4) and (5), can be carried out by any party—we assume a public auditor will do so on behalf of most customers, so that individuals will typically not verify the entire proof (though they are free do to so). We discuss the cost of verifying the entire proof further in Section 10.

[11] The protocol of [6] can be made to work without a trusted setup, but the resulting protocol is considerably less efficient.

1. $\mathcal{E}$ runs Protocol 1 to verifiably generate a commitment Z_{Assets} to its total assets.
2. $\mathcal{E}$ runs Run Protocol 2 to verifiably generate a commitment $Z_{\mathsf{Liabilities}}$ to its total assets and a list $\mathsf{LiabList}$ of its liabilities.
3. $\mathcal{E}$ computes $Z_{\mathsf{Assets}} \cdot {Z_{\mathsf{Liabilities}}}^{-1} = Z_{\mathsf{Assets}-\mathsf{Liabilities}}$.
4. $\mathcal{E}$ proves in zero-knowledge that $Z_{\mathsf{Assets}-\mathsf{Liabilities}}$ is a commitment to the value 0.

Protocol 3: Complete privacy-preserving proof of solvency

5.1.1 Extended customer verification

One simple extension to the Provisions proof of liabilities is to take the pairs of commitments (CID_i, z_i) to each user's identity and balance and arrange them as the leaves of a Merkle tree. Similar to the Maxwell protocol, each internal node of the tree will contain a hash of its two children and a commitment to the sum of their balances. This would enable each user to verify independently that their balance is included in $Z_{\mathsf{Liabilities}}$ by following their path to the root, checking at each node that the committed balance is the product of its two child nodes' committed balances.

In the Maxwell protocol, the user observes the balance in each sibling node on their path to the root to verify that no negative balances have been included. To achieve the same property with Provisions, a range proof must be given for each sibling node's balance to ensure that it is a small positive number. Thus, for each of the $\log_2 c$ nodes on the user's path to the root they must verify one hash, one product of Pedersen commitments and one range proof. Based on our implementation in Section 10, this would lead to per-user proofs of several hundred kB in size for a large exchange.

Because the proof of assets in Provisions cannot be similarly divided on a per-user basis, verifying the proof of solvency would still require a third party verifier for most users. Generating this tree would double the proof-generation time for the exchange, so we leave it as an optional extension. However, if the exchange implemented a simpler, non-privacy-preserving proof of assets (such as moving money on the blockchain) this extension would allow efficient per-verification of the entire proof of solvency comparable to the Maxwell protocol without assuming any third-party verifiers.

6 Proof of solvency

Protocol 3 specifies how $\mathcal{E}$ can complete the proof of solvency given commitments to total assets and liabilities from Protocols 1 and 2. The proof that $Z_{\mathsf{Assets}-\mathsf{Liabilities}}$ is a commitment to 0 (line 4) is a simple Schnorr ZK proof of knowledge of the discrete log of $Z_{\mathsf{Assets}-\mathsf{Liabilities}}$ to the base h, since $Z_{\mathsf{Assets}-\mathsf{Liabilities}} = g^0 h^k$ for a value k known to the exchange and if $Z_{\mathsf{Assets}-\mathsf{Liabilities}}$ were a commitment to any other value then computing its discrete log to the base h would reveal the discrete log of h relative to g.

Variation for exchanges with a surplus If the exchange is actually running a surplus (total assets are greater than total liabilities), this can easily be handled with a simple modification—the exchange can create a commitment to its surplus, Z_{Surplus}, and apply the same range proof used for customer balances to prove that this is a small positive number. It then replaces line 3 in Protocol 3 with:

$$Z_{\mathsf{Assets}} \cdot {Z_{\mathsf{Liabilities}}}^{-1} \cdot {Z_{\mathsf{Surplus}}}^{-1}$$

This approach reveals that a surplus exists. The exchange can also prove the magnitude of its surplus if desired by opening the commitment Z_{Surplus}. Alternatively, to hide even the existence of any surplus, the exchange could simply move its surplus into a separate address which is not included in the addresses $\mathbf{S}$ used in its proof of assets, or include the value of the surplus in a number of fake customers' accounts which will add to its apparent liabilities.

Variation for fractional-reserve exchanges Fractional reserve banking, in which an exchange promises to keep assets equal to only a fraction ρ of its total liabilities instead of all of them, has been frowned upon by many in the Bitcoin community and not seen significant deployment. However if this approach becomes more popular in the future, it is easy to modify Provisions to handle this case by modifying Protocol 3 to commit to a modified balance $f_i(\mathsf{Balance}_i)$ instead of the customer's true balance $\mathsf{Balance}_i$. Each user can then check during verification that f_i was computed correctly on their true balance. Simple fractional reserves could be implemented by defining $f_i(x) = \rho \cdot x$ for all users. It would also be straightforward to define $f_i(x) = \rho_i \cdot x$ with a different ρ_i for each user if, for example, some users' accounts are fully-guaranteed ($\rho_i = 1$) while others are only fractionally-guaranteed ($\rho_i < 1$). Arbitrary other functions are possible, with a natural example from traditional finance being guaranteeing a user's assets up to some maximum value.

Finally, an exchange can also prove that it is running a surplus of proportion ρ by setting $f_i(x) = (1 + \rho) \cdot x$, with a "fractional surplus" effectively being the inverse of a fractional reserve.

7 Proof of non-collusion

Recall from Section 2.3.2 that the privacy guarantees of Provisions introduce the risk that a cabal of insolvent exchanges colluding by covering each exchanges' individual liabilities with their collective assets. In effect, the assets of a single Bitcoin address can be used in the proof of solvency for multiple exchanges. This can be done by having the exchanges contribute to a set of joint NIZKPs of their keys (*e.g.,* using divertable ZK [1]).

The simplest defense is for each exchange to choose an anonymity set $\mathbf{PK}$ which is smaller than the set of all public keys and where each exchange's set is disjoint from the anonymity set of all other exchanges. This ensures that each exchange is (simultaneously) proving solvency using assets it owns and without the help of other exchanges. The difficulty with this approach is that there may not be sufficiently many addresses on the Bitcoin blockchain to accommodate strong privacy for all the exchanges. In the long run, if exchanges come to collectively control the majority of all bitcoins, we would like them to be able to use each other's addresses in their anonymity sets.

Extension to Proof of Assets We can obtain a stronger defense by extending Protocol 1 with a few additional steps. Our goal is to ensure that the assets of every Bitcoin address are used in at most one proof of solvency. Recall that the exchange has a set of Bitcoin signing keys $\mathbf{PK} = \{y_1, \ldots, y_n\}$ where $y_i = g^{x_i}$ for $i \in [1, n]$. The exchange knows the secret keys x_i for some subset of these public keys. We use indicator variables $s_1, \ldots, s_n \in \{0, 1\}$ such that $s_i = 1$ when the exchange knows the secret key x_i and $s_i = 0$ otherwise.

Let $w \in G$ be an element such that no one knows its discrete-log based g or h.[12] We extend Protocol 1 to force every exchange to also compute the list

$$\mathbf{L} := \left\{ w^{\hat{x}_i} = w^{x_i \cdot s_i} \text{ for } i \in [1, n] \right\}$$

which is randomly permuted and published. When $s_i = 1$ the corresponding element in $\mathbf{L}$ is w^{x_i} and when $s_i = 0$ the corresponding element is simply $1 \in G$, the identity element. Thus $\mathbf{L}$ is a random permutation of the exchange's Bitcoin public keys, but using the base w instead of g.

We require the exchange to prove that $\mathbf{L}$ is correctly constructed (*i.e.,* a permutation of $w^{\hat{x}_1}, \ldots, w^{\hat{x}_n}$) using a zero-knowledge proof of knowledge derived from a component of the Neff mix [28] and presented in Appendix D. The proof of knowledge proves that, given two lists $L_1, L_2 \in G^n$, the exchange knows $(z_1, t_1), \ldots, (z_n, t_n)$ such that

$$L_1 = (g^{z_1} h^{t_1}, \ldots, g^{z_n} h^{t_n}) \text{ and } L_2 = \text{permute}(w^{z_1}, \ldots, w^{z_n}).$$

That is, the proof shows that the list L_2 is a permutation, unblinding and base change of the list L_1. The list L_1 is the list of commitments $(l_1, \ldots, l_n)$ generated in Protocol 1. This Neff-like proof thus proves that the published list $\mathbf{L}$ is constructed correctly. It is a simple and efficient proof requiring about $8n$ elements in $\mathbb{Z}_q$, as explained in Appendix D.

We show below that the list $\mathbf{L}$ reveals no information about the $\mathcal{E}$'s Bitcoin addresses beyond the number of addresses ν controlled by $\mathcal{E}$. Note that ν is not revealed by the basic protocol (Protocol 1). We'll return to the implications of making this information public in Section 9.1 but this is one reason (in addition to added complexity) why we present this as an optional protocol extension.

Now, suppose two exchanges collude and use the same Bitcoin address $y = g^x$ in their proof of solvency. Then w^x will appear in the $\mathbf{L}$ list of both exchanges. In other words, the $\mathbf{L}$ lists of these two exchanges will have a non-empty intersection.

Since every exchange is required to publish its list $\mathbf{L}$, an auditor can check that these lists are mutually disjoint (ignoring the elements $1 \in G$). If so, then the auditor is assured that every Bitcoin address is used in at most one proof of solvency and this holds even if all the exchanges use *the same* anonymity set $\mathbf{PK}$.

An important security requirement is that all exchanges run the extension at the same time—barring this, a simple attack is for exchanges to move bitcoins from one address to another in between runs of the protocol so that the same funds can be used but with a different value for $w^{\hat{x}_i} = w^{x_i \cdot s_i}$ in each $\mathbf{L}$ (since x_i will have changed). Fortunately, the blockchain already provides an easy method of synchronization. Exchanges simply need to agree on a common block number (say, every 240$^{\text{th}}$ block to run the protocol daily) and all run the protocol based on the state of the blockchain up to that block. No further synchronization is required; all exchanges can run the protocol and publish their proofs independently and any assets used by more than one exchange will be detectable.

It remains to argue that the list $\mathbf{L}$ reveals no information about the exchange's set of Bitcoin addresses beyond its size ν. This follows directly from the Decision Diffie-Hellman (DDH) assumption which is believed to hold in the secp256k1 group. DDH states that given the tuple $\langle g, w, w^x \rangle$, the quantity g^x is computationally indistinguishable from a random element of G. Therefore, given the list $\mathbf{L}$ it is not possible to distinguish the n-bit string $(s_1, \ldots, s_n) \in \{0, 1\}^n$ from a random string of the same weight.

[12] Just as we discussed for generating h in Section 3.1, w can be chosen by hashing a specified string.

8 Security definition & proof

We now present a general (not specific to Provisions) definition of a privacy-preserving proof of solvency. We say a function $\nu(k)$ is negligible if for all positive polynomials $p(\cdot)$, there is a sufficiently large k such that $\nu(k) < 1/p(k)$.

Let $\mathcal{A}$ and $\mathcal{A}'$ denote mappings $(y = g^x) \mapsto \text{bal}(y)$ where $\mathcal{A} \subseteq \mathcal{A}'$, y is the public key corresponding to a Bitcoin address with private key x and $\text{bal}(y)$ is the amount of currency, or assets, observably spendable by this key on the blockchain.

Let $\mathcal{L}$ denote a mapping $\text{ID} \mapsto \ell$ where ℓ is the amount of currency, or liabilities, owed by the exchange to each user identified by the unique identity ID.

Definition 1 (Valid Pair)**.** *We say that $\mathcal{A}$ and $\mathcal{L}$ are a* valid pair *with respect to a positive integer* MaxBTC *iff $\forall ID \in \mathcal{L}$,*

1. $\sum_{y \in \mathcal{A}} \mathcal{A}[y] - \sum_{ID \in \mathcal{L}} \mathcal{L}[ID] \geqslant 0$ *and*
2. $0 \leqslant \mathcal{L}[ID] \leqslant \mathsf{MaxBTC}$

Consider an interactive protocol ProveSolvency run between an exchange $\mathcal{E}$ and user $\mathcal{U}$ such that

1. $\mathsf{output}_{\mathcal{E}}^{\mathsf{ProveSolvency}}(1^k, \mathsf{MaxBTC}, \mathcal{A}, \mathcal{L}, \mathcal{A}') = \emptyset$
2. $\mathsf{output}_{\mathcal{U}}^{\mathsf{ProveSolvency}}(1^k, \mathsf{MaxBTC}, \mathcal{A}', \text{ID}, \ell) \in \{\textsc{Accept}, \textsc{Reject}\}$

For brevity, we refer to these as $\mathsf{out}_{\mathcal{E}}$ and $\mathsf{out}_{\mathcal{U}}$ respectively.

Definition 2 (Privacy-Preserving Proof of Solvency)**.** *A privacy-preserving proof of solvency is a probabilistic polynomial-time interactive protocol* ProveSolvency*, with inputs/outputs as above, such that the following properties hold:*

1. ***Correctness.*** *If $\mathcal{A}$ and $\mathcal{L}$ are a valid pair and $\mathcal{L}[ID] = \ell$, then $Pr[\mathsf{out}_{\mathcal{U}} = \textsc{Accept}] = 1$.*
2. ***Soundness.*** *If $\mathcal{A}$ and $\mathcal{L}$ are instead not a valid pair, or if $\mathcal{L}[ID] \neq \ell$, then $Pr[\mathsf{out}_{\mathcal{U}} = \textsc{Reject}] \geqslant 1 - \nu(k)$.*
3. ***Ownership.*** *For all valid pairs $\mathcal{A}$ and $\mathcal{L}$, if $Pr[\mathsf{out}_{\mathcal{U}} = \textsc{Accept}] = 1$, then the exchange must have 'known' the private keys associated with the public keys in $\mathcal{A}$; i.e., there exists an extractor that, given $\mathcal{A}$, $\mathcal{L}$, and rewindable black-box access to $\mathcal{E}$, can produce x for all $y \in \mathcal{A}$.*
4. ***Privacy.*** *A potentially dishonest user interacting with an honest exchange cannot learn anything about a valid pair $\mathcal{A}$ and $\mathcal{L}$ beyond its validity and $\mathcal{L}[ID]$ (and possibly $|\mathcal{A}|$ and $|\mathcal{L}|$); i.e., even a cheating user cannot distinguish between an interaction using the real pair $\mathcal{A}$ and $\mathcal{L}$ and any other (equally sized) valid pair $\hat{\mathcal{A}}$ and $\hat{\mathcal{L}}$ such that $\hat{\mathcal{L}}[ID] = \mathcal{L}[ID]$.*

We prove the following theorem in Appendix E:

Theorem 3. *Provisions, as specified in Protocol 3, is a privacy-preserving proof of solvency.*

9 Security discussion

9.1 Anonymity sets

If the anti-collusion protocol extension of Section 7 is used, then the number of Bitcoin addresses ν controlled by the exchange is revealed as well as the size of the anonymity set $n = |\mathbf{PK}|$ (which includes the ν addresses). For efficiency reasons, exchanges may opt to use smaller anonymity sets than the set of all public keys on the blockchain; in particular, if the number of keys grows unexpectedly in the future. In such a case, the exchange must be aware that this might leak some meaningful information about what $\mathcal{E}$'s total assets are.

Specifically, the adversary can determine that $\mathcal{E}$'s assets consist of one of the $\binom{n}{\nu}$ subsets of the anonymity set $\mathbf{PK}$. We remark that $\mathcal{E}$ can easily control n and can also control ν (by splitting accounts up or by padding ν with zero balance accounts). For practical instances, $\binom{n}{\nu}$ grows quickly—*e.g.,* $\nu = 25$ and $\mu = 250$ already yields $\approx 2^{114}$ candidates. That said, we have no idea what types of external information might be useful for eliminating unlikely or impossible totals from this set (*e.g.,* the adversary's corruption of customers may provides them with a lower bound on the total assets), or for whittling n down by eliminating addresses known or suspected not to be controlled by the exchange. Research on deanonymizing Bitcoin addresses, *e.g.,* through clustering and reidentification [24], has demonstrated that Bitcoin's anonymity is limited (see [5] for a survey).

If an exchange conducts proofs of solvency on a regular basis (or more than once), each anonymity set should be based closely on the anonymity set used previously—choosing independent anonymity sets could reveal the exchange's addresses by intersecting the sets. Exchanges can remove addresses from their anonymity set if the criteria for doing so is independent of whether the exchange owns the address or not. For example, it might remove addresses once the balance is under a certain threshold. However, generally, anonymity sets should grow over time with new addresses (some owned by the exchange and some as cover) being added to the set.

We leave the process of developing and analyzing a heuristic for forming an anonymity set (in terms of size of n and ν and the distribution of amounts across the ν accounts) as future work. For the current state of Bitcoin at the time of writing, we show in Section 10 that it is reasonable for all exchanges to choose an anonymity set equal to most available accounts, sieving out tiny "dust" accounts.

9.2 User Verification

A proof of solvency *enables* user verification, but it does not guarantee that all users actually perform the verification. Consider a malicious $\mathcal{E}$ that does not correctly include some set of users accounts—by either omitting them or zeroing their balances. Assume the exchange has U users, F (for fraudulent) entries, and that a *random* subset $A \subset U$ of users choose to audit the correctness of LiabList. In this case, the probability that an adversary will go undetected is $\binom{U-F}{A}/\binom{U}{A}$, which is closely bounded from above by $\min[(1-A/U)^F, (1-F/U)^A]$ (*cf.* the probability of a malicious election authority being caught modifying ballot receipts in a cryptographic voting system [11]). This probability decreases close-to-exponentially in F and A. Due to the approximation, we conservatively conclude the probability of being caught is high, instead of overwhelming.

Next, one might question the assumption that each customer is equally likely to verify LiabList. However, it is reasonable that the distribution skews in the direction of customers with high balances (and thus more at stake) being more likely to check. This is actually beneficial, because the probability of catching a malicious exchange does not depend on the amount of bitcoin zeroed out. In other words, zeroing out the

largest account is equivalent to zeroing out the smallest in terms of being caught, yet the former action better benefits the adversary's goal of lowering its liabilities.

We also note that Provisions as described does not provide dispute resolution. If a user finds their account missing or balance incorrect, they do not have sufficient cryptographic evidence that this is the case [21]. The issue appears unsolvable cryptographically. Recall that the primary motivation for users keeping funds with an exchange is to avoid needing to remember long-term cryptographic secrets, therefore exchanges must be able to execute user orders and change their balance without cryptographic authentication from the user (*e.g.,* password authentication). Resolving this will likely require legal regulation. Users who dislike an exchange may also falsely claim that verification of their accounts failed, and it is not possible to judge if the user or the exchange is correct in this case based on a Provisions transcript alone.

Lastly, we note that if a user does verify their account, they should use a verification tool other than one provided by the exchange itself; such a tool could be automated to increase participation. All of the issues discussed in this remark deserve followup work to ensure that Provisions is implemented in practice in such a way that users are likely to perform auditing and to do so correctly.

9.3 Access to private keys

One important practical consideration with Provisions is that it requires computation using private keys which exchanges should, by best practice, protect very carefully. These keys might, for example, be kept in *cold storage* in an airgapped hardware security module (HSMs) or even kept on paper in a secure vault. Performing regular Provisions proofs might therefore impact security practices at exchanges. This problem is inherent to any cryptographic proof of assets. If the proof is conducted at a regularly scheduled time, this can be integrated with access control practices using human intervention to access the private keys.

Each private key is needed for an addition in Step (1d) of Protocol 1. This presents a challenge in that some HSMs only support an API to perform a complete ECDSA signature with the private key and do not support an isolated addition with the key. Exchanges interested in implementing Provisions might need to carefully consider if they are using HSMs with a rich enough API to enable the proof-of-assets computation. Alternatively, exchanges could request their HSM providers to implement the required API. This feature could be a way for HSMs providers to differentiate themselves from the competition. An interesting challenge for future research is to design a proof-of-assets protocol that is compatible with a device that only performs complete ECDSA signatures.

The fact that Provisions only requires an addition with the private key also provides an opportunity: while implementing threshold ECDSA (or DSA in general) is quite challenging, it is straightforward to create additive key shares that support private key addition in a threshold fashion. This would enable an exchange implementing Provisions to distribute key shares to multiple servers (or HSMs) which must collaborate to compute a Provisions proof.

10 Implementation

10.1 Asymptotic performance

Provisions scales linearly in proof size, construction and verification time with respect to its inputs: the proof of assets scales with the size of the anonymity set and the proof of liabilities scales with the number of customer accounts. The final proof of solvency given an encryption of the total assets and an encryption of the total liabilities is constant and in practice is negligible. All of the linear parts of the protocol can be run

in parallel and require only associative aggregations to compute homomorphic sums, meaning the protocol is straightforward to parallelize.

Specifically the proof of assets is linear in n, the number of public keys in the anonymity set, regardless of the size of $\mathbf{S}$, the total number of accounts actually owned by $\mathcal{E}$, requiring $10n$ integers from $\mathbb{Z}_q$ in total. The proof of liabilities is linear with respect to the number of customers c. It is dominated by $m+1$ elements from $\mathbb{Z}_q$ used to commit to each bit of each customer's balance, where $m = \lceil \lg_2 \mathsf{MaxBTC} \rceil = 51$. If needed, an exchange could slightly reduce proof sizes by capping the size of assets below or reducing precision. For example, with $m = 32$ the exchange could still include accounts worth up to US$1 billion with precision to the nearest penny. However, we'll assume full precision is desired in our implementation.

Full verification of the protocol requires approximately equal time to the construction of the proof. For customers opting to only validate their own balance's correct inclusion in the proof and trust a third party to run the full verification, verification is much simpler, the customer to check their CID value with a single hash and check that y_i is a correct commitment their balance which requires only $m + 2$ group operations.

10.2 Incremental updates

As described in Section 1 the protocol is intended to be run often (e.g. daily) to give continued proof of solvency. A natural question is whether it is possible to update the proof incrementally. We will consider updates to the anonymity set, to the assets proof and to the liabilities proof separately.

The full set of addresses (anonymity set + owned addresses) used in the proof is public. As such any newly created addresses by the exchange need to be published. To hide these new addresses it is important to additionally add addresses to the anonymity set. As with the anonymity set in general and discussed in Section 9.1 it is important to choose in such a way that the actual addresses are indistinguishable from it. A proper implementation would for example add addresses deterministically (e.g. all addresses with balances over x bitcoin).

The asset proof is almost perfectly separable, in that there is a separate and independent component for each address in the full set of addresses. The components for new addresses and addresses with changed balance need to be updated. However, it is not necessary to update the components of all other addresses. This is especially useful for cold addresses, which do not have a private key easily accessible. The set of addresses which are new or have changed balances is public on the blockchain anyways and thus no additional information is leaked.

The liabilities proof mainly consists of a commitment to each customer's balance and a proof that said balance is within a range. For all new users and users whose balance changed the commitment the proof needs to be redone. For the other users it is not technically necessary to redo the proof. However, not changing the proofs for customers whose balance remained unchanged will leak how many users were actively using their account between the two proofs. If the complete proof were redone then this information would remain private. If an exchange were to accept this privacy leak it could drastically reduce the size of the proof updates.

10.3 Practical parameter sizes

An exchange could achieve optimum anonymity by choosing the anonymity set $\mathbf{PK}$ to be the entire set of unclaimed transaction outputs (called the *UTXO set*) which represents all potentially active Bitcoin accounts. The size of the UTXO set has steadily increased throughout Bitcoin's history [5] and at the time of this writing contains approximately 17M addresses. However, the vast majority of these are "dust" addresses holding only a tiny value. As of October 2015, there are roughly 1.3 Million Addresses with more

than 0.01 BTC, which collectively control 99.97% of all bitcoin.[13] As discussed in Section 4.2, some of these addresses are unusable for the protocol because they do not have public keys available (*i.e.,* they are pay-to-pub-key-hash addresses with only a hash of the public key visible in the blockchain), others have questionable anonymity value as they have never been moved since being mined and exchanges are not expected to be mining their own bitcoin directly. Out of these addresses roughly 431,000 have performed a send action and are thus candidates for the anonymity set. Consequently we tested our implementation with anonymity sets up to 500,000.

On the proof-of-liabilities side, Coinbase is thought to be one the largest exchanges and currently claims roughly 2 million customers.[14] We take as our goal supporting this number of users.

10.4 Implementation & performance tests

To test the performance of our protocol in practice we created a prototype implementation of our protocol in Java 1.8. All cryptographic operations are performed using BouncyCastle,[15] a standard cryptographic library for Java which is also used by the popular bitcoinj implementation of Bitcoin in Java. We performed tests on a commodity server with 2 E5-2680 v2 Xenon processors and 128GB RAM. The max heap size of the JVM was set to the default 256MB. Our implementation assumes a previously downloaded and verified blockchain, to enable efficient balance lookups and selection of an appropriate anonymity set.

Our simulations confirm that Provisions should be practical even for large exchanges desiring strong anonymity and full precision to represent customer accounts. Figure 2 shows proof sizes and computation times for Protocol 1, the proof of assets, varying the anonymity set size n from 10 to 500,000. Figure 3 shows proof sizes and computation times for Protocol 2, the proof of liabilities, varying the number of customers c from 1,000 to 2,000,000. We tested with $m = 51$, supporting full precision of account balances. Reducing m would lead to proportional reductions in proof sizes and construction times. Note that, given realistic parameters today, it appears that the proof of liabilities is the more expensive protocol today for a large exchange.

We report numbers without the protocol extension from Section 7 to ensure assets are not shared between colluding exchanges executing the protocol contemporaneously. This extension requires publishing the element $w^{\hat{x}_i}$ plus eight additional values per address which increases the size and construction time of the proof of assets by about $\sim 80\%$. Because the proof of liabilities is likely much larger, this extension makes only a minor impact on performance.

We omit performance figures for Protocol 3 as this protocol is constant size and negligible compared to Protocols 1 and 2. Similarly, verification time for individual clients depends only m and not the anonymity set or number of other customers. In our implementation it took fewer than 10 ms.

11 Open research challenges

In this section we briefly summarize the open research challenges for proofs of solvency:

- **Multisig addresses** Provisions does not enable multisig addresses (Section 4.2). In principle, the proof of assets of Section 4.1 can be extended to prove control of the required majority of keys in a multisig address (assuming they are all public) without substantially new cryptographic techniques.

[13] `https://bitinfocharts.com/top-100-richest-bitcoin-addresses.html`
[14] `https://www.coinbase.com/about`
[15] `https://www.bouncycastle.org/`

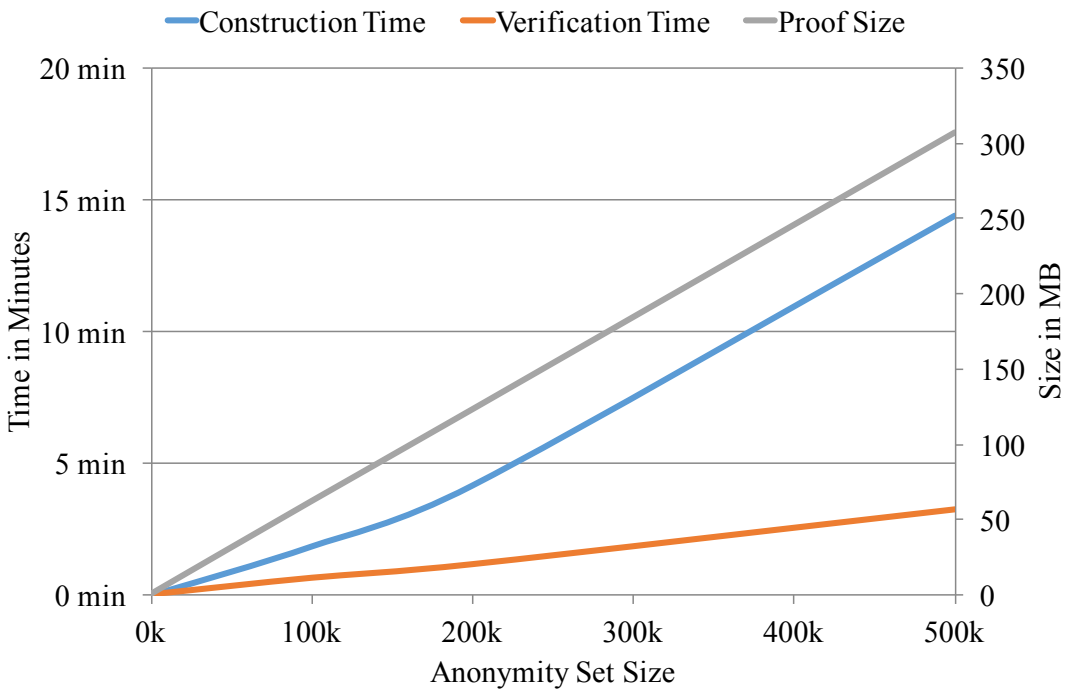

Figure 2: Performance for Protocol 1 (proof of assets).

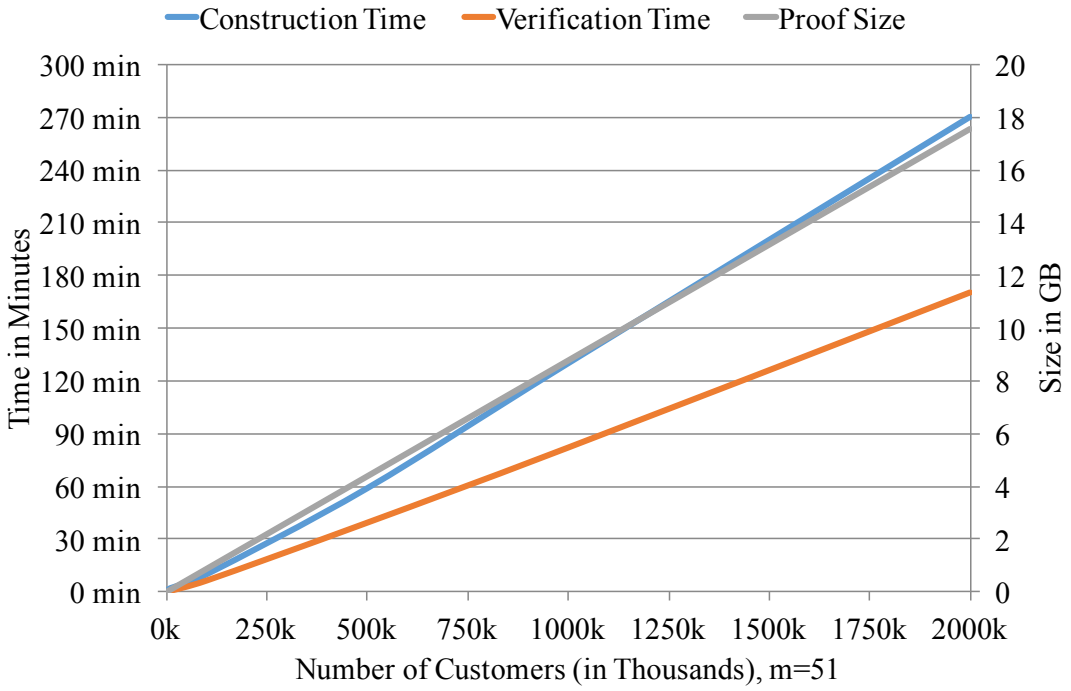

Figure 3: Performance for Protocol 2 (proof of liabilities).

- **Non-public public keys** Provisions does not enable using addresses in the anonymity set for which the public keys are unknown. As discussed in Section 4.2, this prevents using pay-to-pubkey-hash and pay-to-script-hash addresses which are common in Bitcoin. Removing this restriction requires proving knowledge of a hash preimage which will necessitate more powerful proof techniques such as zk-SNARKs.

- **Arbitrary scripts** Bitcoin enables arbitrary scripts for transaction redemption. The most general framework for solvency proofs would enable zero-knowledge proofs of the ability to satisfy *any* redemption script. In principle this could be achieved by compiling an interpreter for Bitcoin scripts in a general-purpose proof system such as zk-SNARKs. The practical value of this may be low as most Bitcoin scripts conform to one of a small number of templates.

- **Zerocash** Zerocash [3] is a proposed cryptocurrency with strong privacy properties. All transactions are zk-SNARKs, revealing no information about senders, recipients, or value. Proofs of solvency for Zerocash would require a substantially different proof of assets. Using zk-SNARKs is the obvious approach since they are already required for the currency. It might be valuable to combine the trusted setup for the currency itself with a trusted setup to enable solvency proofs.

- **Velocity limits** It might be considered valuable for an exchange to demonstrate stability by proving that some portion of its holdings have not been transferred for some period of time. The "age" of each UTXO in Bitcoin is publicly visible, so it would be possible to add this parameter to the proof-of-assets and compute a weighted sum of assets based on age.

- **Compatibility with ECDSA-only HSMs** As discussed in Section 9.3, some exchanges might keep their private keys in HSMs which only export ECDSA signatures and do not enable the raw multiplication needed for the Provisions proof of assets. Designing a new proof of assets around this limitation is an interesting challenge.

12 Concluding Remarks

Stu Feldman has outlined a roadmap for technical maturity (as quoted in [18]):

1. You have a good idea;
2. You can make your idea work;
3. You can convince a (gullible) friend to try it;
4. People stop asking why you are doing it; and
5. Other people are asked why they are not doing it.

Given the shaky track record of Bitcoin exchanges, the onus upon an exchange to perform some kind of audit is nearing level 5. However, cryptographic solvency proofs, like the Maxwell protocol, are lagging behind around level 3. Our belief is that the privacy implications of Maxwell are hindering it—there are good reasons for an exchange not to reveal which addresses it controls, the scale of its total holdings, or potentially leak information about large customers' account sizes. Provisions removes these barriers. While cryptographic proofs of solvency still have inherent limits, namely that control of an address' key at present does not guarantee the future ability to use that key to refund customers, we believe that with Provisions there are no longer good reasons for an exchange *not* to provide regular proofs of solvency to increase customer confidence.

Acknowledgments

We thank the reviewers for their insights. We especially thank our shepherd Sarah Meiklejohn for her constructive feedback and contributions which improved the paper. J. Bonneau is supported by a Secure Usability Fellowship from the Open Technology Fund and Simply Secure as well and is also supported by DARPA. J. Clark acknowledges funding from NSERC, FQRNT and Concordia OVPRGS. D. Boneh acknowledges funding from NSF, DARPA, and a grant from ONR. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[1] O. Baudron, P.-A. Fouque, D. Pointcheval, G. Poupard, and J. Stern. Practical multi-candidate election system. In *ACM PODC*, 2001.

[2] M. Belenkiy. E-Cash. In *Handbook of Financial Cryptography and Security*. CRC, 2011.

[3] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *IEEE Symposium on Security and Privacy*. IEEE, 2014.

[4] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, 2013.

[5] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. *IEEE Symposium on Security and Privacy*, 2015.

[6] F. Boudot. Efficient proofs that a committed number lies in an interval. In *Proc. of EUROCRYPT*, pages 431–444, 2000.

[7] J. Camenisch, R. Chaabouni, and A. Shelat. Efficient protocols for set membership and range proofs. In *ASIACRYPT*, pages 234–252, 2008.

[8] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *EUROCRYPT*, 2005.

[9] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0., 2000.

[10] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.

[11] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity II: end-to-end verifiability for optical scan election systems using invisible ink confirmation codes. In *EVT*, 2008.

[12] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, 1992.

[13] Conference of State Bank Supervisors. State Regulatory Requirements for Virtual Currency Activities Model Regulatory Framework, Sep. 2015.

[14] R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In *Crypto'98*, pages 424–441, 1998.

[15] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *CRYPTO*, 1994.

[16] S. Eskandari, D. Barrera, E. Stobert, and J. Clark. A first look at the usability of bitcoin key management. In *USEC*, 2015.

[17] A. Fiat and A. Shamir. Witness indistinguishable and witness hiding protocols. In *ACM STOC*, 1990.

[18] D. Geer. Technical maturity, reliability, implicit taxes, and wealth creation. *login: The magazine of Usenix & Sage*, 26(8), 2001.

[19] S. Goldfeder. Better wallet security for bitcoin. Technical report, Princeton, March 2014.

[20] C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols*. Springer, 2010.

[21] R. Kusters, T. Truderung, and A. Vogt. Accountability: Definition and relationship to verifiability. In *ACM CCS*, 2010.

[22] P. Litke and J. Stewart. Cryptocurrency-stealing malware landscape. Technical report, Dell SecureWorks Counter Threat Unit, 2014.

[23] W. Mao. Guaranteed correct sharing of integer factorization with off-line shareholders. In *PKC*, 1998.

[24] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *IMC*, 2013.

[25] R. C. Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford, 1979.

[26] T. Moore and N. Christin. Beware the middleman: Empirical analysis of bitcoin-exchange risk. In *Financial Cryptography and Data Security*, 2013.

[27] S. Nakamoto. Bitcoin: A peer-to-peer electionic cash system. Unpublished, 2008.

[28] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *ACM CCS*, 2001.

[29] R. Parhonyi. Micropayment Systems. In *Handbook of Financial Cryptography and Security*. CRC, 2011.

[30] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1992.

[31] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptography*, 4, 1991.

[32] A. Vogt, T. Truderung, and R. Kusters. Clash attacks on the verifiability of e-voting systems. In *IEEE Symposium on Security and Privacy*, 2012.

[33] Z. Wilcox. Proving your bitcoin reserves. https://iwilcox.me.uk/2014/proving-bitcoin-reserves, Feb. 2014.

A Zero-Knowledge

For completeness, we include the definition of an honest-verifier zero-knowledge proof of knowledge (HVZKPK) for Σ-protocols.

Definition 3. *Given a relation R between instance x and (private) witness w, then protocol π between prover P and verifier V is special-honest zero-knowledge if the following hold:*

- ***Completeness:*** *If P and V are honest, then V accepts for all $(x, w) \in R$.*
- ***Honest verifier zero-knowledge:*** *There exists a polynomial time simulator S that can compute w given x and a challenge can create accepting transcripts that are indistinguishable from a transcripts generated by an honest P and W.*
- ***Proof of knowledge:*** *There exists a polynomial time extractor E that given two accepting transcripts that differ in challenges but have the same initial message, outputs a witness w.*

B ZK for Binary Commitment

Protocol 4 is a standard zero-knowledge proof of knowledge that for a given Pedersen commitment $(g, h, l = g^x h^y)$ the prover $\mathcal{E}$ knows $x \in \{0, 1\}$ and $y \in \mathbb{Z}_q$. The protocol is based on [15] and also discussed in [6, Sec. 1.2.1]. The protocol can be made non-interactive using the Fiat-Shamir heuristic. The proof is just four elements in $\mathbb{Z}_q$, namely (c, c_1, r_0, r_1).

Verifier's input: $g, h, l \in G$
Prover's input: $x \in \{0, 1\}$ and $y \in \mathbb{Z}_q$ s.t. $l = g^x h^y$

1. $\mathcal{E}$ chooses $u_0, u_1, c_f \xleftarrow{\$} \mathbb{Z}_q$ and sends
$$a_0 = h^{u_0} g^{-x \cdot c_f}, \quad a_1 = h^{u_1} g^{(1-x)c_f}$$
2. The verifier responds with challenge $c \xleftarrow{\$} \mathbb{Z}_q$
3. $\mathcal{E}$ computes $c_1 = x \cdot (c - c_f) + (1 - x) \cdot c_f$
and responds with
$$c_1, \quad r_0 = u_0 + (c - c_1) \cdot y, \quad r_1 = u_1 + c_1 \cdot y$$
4. The verifier accepts if
$$h^{r_0} \stackrel{?}{=} a_0 (l)^{c - c_1} \quad \text{and} \quad h^{r_1} \stackrel{?}{=} a_1 (l g^{-1})^{c_1}$$

Protocol 4: Knowledge of Binary Number Protocol

For completeness, security is shown in Claims B.1, B.2, and B.3 below.

Claim B.1. ***(Completeness)*** *If $\mathcal{P}$ and a verifier $\mathcal{V}$ follow protocol 4 on input (g, h, l) and private input (x, y) where $x \in \{0, 1\}$ then $\mathcal{V}$ always accepts.*

Proof. Suppose that $\mathcal{E}$ knows x and $x = 0 \vee x = 1$. Then $\mathcal{E}$ can always compute a_0, a_1, r_0, r_1 and c_1. Let $c_t = c - c_f$. If $\mathcal{E}$ follows the protocol then an honest verifier will find that:

$$\begin{aligned} a_0(lg)^{c-c_1} &= h^{u_0} g^{-xc_f} (g^x h^y)^{c-c_1} \\ &= h^{u_0} h^{(c-c_1)y} g^{-xc_f} g^{xc} g^{-c_1 x} \\ &= h^{u_0} h^{(c-c_1)y} g^{-xc_f} g^{xc} g^{-xc_t} \\ &\qquad (\text{because } x^2 = x \iff x \in 0, 1) \\ &= h^{u_0} h^{(c-c_1)y} g^{x \cdot (c - c_f - c_t)} \\ &\qquad (\text{because } c = c_f + c_t) \\ &= h^{u_0} h^{(c-c_1)y} = h^{r_0} \end{aligned}$$

and

$$\begin{aligned} a_1(lg^{-1})^{c_1} &= h^{u_1} g^{(1-x)c_f} (g^x h^y g^{-1})^{xc_t + (1-x)c_f} \\ &= h^{u_1} h^{c_1 y} g^{(1-x)c_f} g^{xc_t} g^{-xc_t} g^{-(1-x)c_f} \\ &\qquad (\text{because } x^2 = x \iff x \in 0, 1) \\ &= h^{u_1} h^{c_1 y} = h^{r_1} \end{aligned}$$

and thus accept the transcript.

□

Claim B.2. ***(proof of knowledge)*** *There exists a polynomial-time algorithm (extractor E) for Protocol 4 that extracts $x \in \{0, 1\}$ and $y \in \mathbb{Z}_q$ such that $l = g^x h^y$.*

1. Run $P^\star$ to obtain a_0, a_1
2. Send $c \xleftarrow{\$} \mathbb{Z}$ to $P^\star$
3. $P^\star$ outputs c_1, r_0, r_1 such that $h^{r_0} = a_0 l^{c-c_1}$ and $h^{r_1} = a_1 (lg^{-1})^{c_1}$
4. Rewind $P^\star$ to right after step 1 of the protocol.
5. Send $c' \xleftarrow{\$} \mathbb{Z}_q \backslash \{c\}$ to $P^\star$
6. $P^\star$ will output c_1', r_0', r_1' such that $h^{r_0'} = a_0 l^{c'-c_1'}$ and $h^{r_1'} = a_1 (lg^{-1})^{c_1'}$
7. If $c_1' \neq c_1$ then output $y = \frac{r_1 - r_1'}{c_1 - c_1'}$ and $x = 1$ otherwise output $y = \frac{r_0 - r_0'}{c - c'}$ and $x = 0$

Figure 4: Extractor for Binary Proof

Proof. An extractor is given in Figure 4 that extracts y and either $x = 1$ or $x = 0$ from any efficient prover $P^\star$ that convinces the verifier. □

Claim B.3. ***(Honest Verifier Zero-knowledge)*** *There exists a probabilistic polynomial-time simulator S that, given (g, h, l) and random challenge c, can produce a transcript that has the same distribution as a transcript between $\mathcal{P}$ and an honest verifier.*

1. Choose c_1, r_0 and r_1 uniformly at random from $\mathbb{Z}_q$
2. Let $a_0 = h^{r_0}/l^{c-c_1}$ and $a_1 = h^{r_1}/(lg^{-1})^{c_1}$
3. Publish $(a_0, a_1; c, c_1; r_0, r_1)$

Figure 5: Simulator for Binary Proof

Proof. Finally, we prove that the protocol is honest-verifier zero-knowledge, i.e. there exists a simulator that, given the statement $l = g^x h^y, x \in \{0, 1\}$ and a uniformly random challenge c, can create transcripts that come from the same distribution as transcripts between $\mathcal{E}$ an an honest verifier. The simulator is given in Figure 5.

Given a uniformly random challenge c, both a_0 and a_1 are distributed uniformly at random in G, subject to the verification conditions. Thus the simulator produces transcripts from the same distribution as an honest verifier. □

C Proof of assets is HVZKPK

Theorem 1 states that Protocol 1 is an honest-verifier zero-knowledge argument of knowledge. The proof follows from the following three claims. We denote the prover by $\mathcal{P}$ and verifier by $\mathcal{V}$.

Claim C.1. ***(Completeness)*** *If $\mathcal{P}$ and $\mathcal{V}$ follow protocol 1 on input $(g, h, y_i, b_i, l_i, p_i)$ and private input $(s_i \in \{0, 1\},\ v_i, t_i, \hat{x}_i)$, then $\mathcal{V}$ always accepts.*

Proof. Suppose $\mathcal{E}$ knows v_i , s_i and t_i. By assumption $\mathcal{E}$ followed the protocol and all p_i and l_i are, thus, well formed. $\mathcal{P}$ can then for any random $u_i^{(\cdot)}$ and challenges c_i compute all responses $r_{(\cdot)}$. Completeness follows by a simple calculation. □

Claim C.2. ***(Argument of knowledge)*** *Let $\mathcal{P}^\star$ be an efficient prover. There exists a polynomial-time algorithm (extractor E) for Protocol 1 such that for each $i \in [1, n]$ and any pair of accepting transcripts with the same $a_i^{(\cdot)}$ and $c_i \neq c_i'$, $\mathcal{P}^\star$ can output $(s_i \in \{0, 1\},\ v_i, t_i, \hat{x}_i)$ satisfying conditions* (1)*,* (3) *and* (5) *for all $i \in [1, n]$.*

Proof. We show in Figure 6 that there exists an extractor E for all efficient provers $\mathcal{P}^\star$ that convince the verifier. Note that

$$\prod_{i=1}^{n} (b_i^{r_{s_i} - r'_{s_i}} h^{r_{v_i} - r'_{v_i}})^{\frac{1}{c_i - c'_i}} = \prod_{i=1}^{n} p_i = Z_{\mathsf{Assets}}$$

$$(y_i^{r_{s_i} - r'_{s_i}} h^{r_{t_i} - r'_{t_i}})^{\frac{1}{c_i - c'_i}} = l_i$$

1. For $i \in [1, n]$
   (a) Run $P^\star$ to obtain $a_i^{(1)}, a_i^{(2)}, a_i^{(3)}$
   (b) Send $c_i \xleftarrow{\$} \mathbb{Z}_q$ to $P^\star$
   (c) $P^\star$ will output $r_{s_i}, r_{t_i}, r_{\hat{x}_i}, r_{v_i}$ such that

$$\begin{aligned} b_i^{r_{s_i}} h^{r_{v_i}} &= p_i^{c_i} a_i^{(1)} \\ y_i^{r_{s_i}} h^{r_{t_i}} &= l_i^{c_i} a_i^{(2)} \\ g^{r_{\hat{x}_i}} h^{r_{t_i}} &= l_i^{c_i} a_i^{(3)} \end{aligned}$$

   (d) Rewind $\mathcal{P}^\star$ to right after step 1b of the protocol.
   (e) Send $c_i' \xleftarrow{\$} Z_q \backslash \{c_i\}$ to $P^\star$
   (f) $\mathcal{P}^\star$ will output $r'_{s_i}, r'_{t_i}, r'_{\hat{x}_i}, r'_{v_i}$ such that

$$\begin{aligned} b_i^{r'_{s_i}} h^{r'_{v_i}} &= p_i^{c'_i} a_i^{(1)} \\ y_i^{r'_{s_i}} h^{r'_{t_i}} &= l_i^{c'} a_i^{(2)} \\ g^{r'_{\hat{x}_i}} h^{r'_{t_i}} &= l_i^{c'} a_i^{(3)} \end{aligned}$$

   (g) Output: $v_i = \frac{r_{v_i} - r'_{v'_i}}{c_i - c'_i} \mod q$, $s_i = \frac{r_{s_i} - r'_{s_i}}{c_i - c'_i} \mod q$, $t_i = \frac{r_{t_i} - r'_{t_i}}{c_i - c'_i} \mod q$, $\hat{x}_i = \frac{r_{\hat{x}_i} - r'_{\hat{x}_i}}{c_i - c'_i} \mod q$

Figure 6: Extractor for Proof of Assets protocol

and

$$(g^{r_{\hat{x}_i} - r'_{\hat{x}_i}} h^{r_{t_i} - r'_{t_i}})^{\frac{1}{c_i - c'_i}} = l_i$$

Since $c_i \neq c_i'$ we can conclude that E gives valid outputs and thus that the protocol is sound.

Additionally if the Protocol 4 was run on p_i, the extractor described in B.2 can extract a binary s_i' and $v_i' \in \mathbb{Z}_q$. Since $\mathcal{P}^*$ is efficient, the binding property of Pedersen commitments ensures that $s_i = s_i'$ and $v_i = v_i'$. □

Claim C.3. ***(Honest Verifier Zero-knowledge)*** *There exists a probabilistic polynomial-time simulator S that, given $(g, h, y_i, bal(y_i), l_i, p_i$ and random challenge c_i for each $i \in [1, n]$, can produce a transcript that has the same distribution as a transcript between $\mathcal{P}$ and an honest verifier.*

Proof. A simulator is given in Figure 7. Note that both the original a's as well as the simulated a's are distributed uniformly at random in G given that the challenge c_i was chosen uniformly at random. Given uniformly chosen u's the responses in the protocol are uniform in $\mathbb{Z}_q$. The simulated responses are uniformly drawn. The probability of a simulated transcript thus equals the probability of an actual transcript. □

D Proof of permutation

Our extension in Section 7 requires a way for the exchange to prove that, given two lists $L_1, L_2 \in G^n$, the list $L_2 \in G^n$ is a permutation, unblinding and base change of the list $L_1 \in G^n$. More precisely, for $g, h, w \in G$ and $L_1, L_2 \in G^n$ the exchange needs to prove knowledge of $(z_1, t_1), \ldots, (z_n, t_n)$ such that

$$L_1 = (g^{z_1} h^{t_1}, \ldots, g^{z_n} h^{t_n}) \quad \text{and} \quad L_2 = \text{permute}(w^{z_1}, \ldots, w^{z_n}).$$

For convenience, we slightly re-write this statement as follows: given g, h, w, L_1, L_2 the exchange must prove knowledge of $(x_1, t_1, y_1), \ldots, (x_n, t_n, y_n)$ such that

$$\begin{aligned} &L_1 = (g^{x_1} h^{t_1}, \ldots, g^{x_n} h^{t_n}), \quad L_2 = (w^{y_1}, \ldots, w^{y_n}), \text{ and} \\ &(y_1, \ldots, y_n) \text{ is a permutation of } (x_1, \ldots, x_n). \end{aligned} \tag{7}$$

The protocol to do so is based on an idea used in the Neff mix net [28]. It makes use of the following idea: define two polynomials

$$p_1(r) := \prod_{i=1}^{n} (r - x_i) \qquad \text{and} \qquad p_2(r) := \prod_{i=1}^{n} (r - y_i).$$

If $(y_1, \ldots, y_n)$ is a permutation of $(x_1, \ldots, x_n)$ then p_1 and p_2 are the exact same polynomial. Otherwise, they are different polynomials of degree-n and therefore agree on at most n points. To test that $Y := (y_1, \ldots, y_n)$ is a permutation of $X := (x_1, \ldots, x_n)$ the verifier can choose a random point $r \in \mathbb{Z}_q$ and check that $p_1(r) = p_2(r)$. If equality holds the verifier can safely conclude that the lists are the same. To see why, observe that if Y is not a permutation of X then the only way that the verifier would be fooled is if r happens to be one of the points where p_1 agrees with p_2. Since that are at most n such points, the probability that r is one of them is at most n/q, which is negligible.

To turn this idea into an argument of knowledge for (7) the verifier first sends a random $r \in \mathbb{Z}_q$ to the exchange. The exchange chooses a random $s_n \in \mathbb{Z}_q$ and publishes the quantities

$$A_n := g^{p_1(r)} h^{s_n} \quad \text{and} \quad B_n := w^{p_2(r)}$$

along with a proof that (1) both quantities are constructed correctly and (2) $p_1(r) = p_2(r)$. The latter proof is a standard Chaum-Pedersen proof of representation, namely that the exchange knows two values $a_n, s_n \in \mathbb{Z}_q$ such that

$$A_n = g^{a_n} h^{s_n} \quad \text{and} \quad B_n = w^{a_n}. \tag{8}$$

1. For each $i \in [1, n]$
   (a) Choose $r_{s_i}, r_{t_i}, r_{\hat{x}_i}, r_{v_i}$ and the challenge c_i uniformly at random from $\mathbb{Z}_q$
   (b) Let: $a_i^{(1)} = b_i^{r_{s_i}} h^{r_{v_i}} p_i^{-c_i}$, $a_i^{(2)} = y_i^{r_{s_i}} h^{r_{t_i}} l_i^{-c_i}$, $a_i^{(3)} = g^{r_{\hat{x}_i}} h^{r_{t_i}} l_i^{-c_i}$
   (c) Publish $(a_i^{(1)}, a_i^{(2)}, a_i^{(3)};\ c_i\ ; r_{s_i}, r_{t_i}, r_{\hat{x}_i}, r_{v_i})$ as the transcript.

Figure 7: Simulator for Proof of Assets protocol

Proving that A_n and B_n are constructed correctly requires n additional sub-proofs. Let $s_1 := -t_1$ and choose random $s_2, \ldots, s_n \in \mathbb{Z}_q$. The exchange publishes the $2n$ quantities

$$A_i := g^{\prod_{i=1}^{i}(r-x_i)} \cdot h^{s_i} \quad \text{and} \quad B_i := w^{\prod_{i=1}^{i}(r-y_i)}.$$

Notice that A_n and B_n are the same as their definition in (8).

The verifier can easily check that A_1 and B_1 are constructed correctly by checking that $A_1 = g^r / L_1[1]$ and $B_1 = w^r / L_2[1]$.

Now, suppose that for some $i \geqslant 1$ the verifier is convinced that A_i, B_i are constructed correctly. Convincing the verifier that A_{i+1}, B_{i+1} are constructed correctly amounts to proving the following facts:

1. Suppose A_i is a Pedersen commitment to a number $a_i \in \mathbb{Z}_q$ and $L_1[i+1]$ is a Pedersen commitment to a number $x_{i+1} \in \mathbb{Z}_q$. Then A_{i+1} is a Pedersen commitment to a number $a_i(r - x_{i+1})$.
2. $(w,\ L_2[i+1],\ B_i,\ (B_i)^r / B_{i+1})$ is a Diffie-Hellman tuple.

The first can be done using a protocol of Cramer and Damgård [14, Lemma 2.6] and the second is a Chaum-Pedersen proof [12]. They amount to proving knowledge of six quantities

$$a_i, s_i, x_{i+1}, t_{i+1}, v_{i+1}, b_i \ \in \mathbb{Z}_q \tag{9}$$

such that

$$\begin{aligned}
A_i &= g^{a_i} \cdot h^{s_i} \\
L_1[i+1] &= g^{x_{i+1}} \cdot h^{t_{i+1}} \\
A_{i+1} &= (A_i)^r \cdot L_1[i+1]^{-a_i} \cdot h^{v_{i+1}} && (= g^{a_i(r-x_{i+1})} h^{(\cdots)}) \\
B_i &= w^{b_i} \\
B_{i+1} &= (B_i)^r \cdot L_2[i+1]^{-b_i} && (= w^{b_i(r-y_{i+1})})
\end{aligned}$$

Proving knowledge of such six quantities is a standard Chaum-Pedersen proof of knowledge. Note that an honest exchange would use the following value for v_{i+1} in the proof to ensure that $A_{i+1} = g^{a_{i+1}} \cdot h^{s_{i+1}}$:

$$v_{i+1} := s_{i+1} - r s_i + a_i t_{i+1}$$

We see that for each $i \in [1, n]$ the exchange must publish A_i, B_i plus six additional elements in $\mathbb{Z}_q$ (the length of the Chaum-Pedersen proof for proving knowledge of the elements in (9), after applying the Fiat-Shamir heuristic).

This completes our description of the protocol for ensuring that the list L_2 is a permutation, unblinding and base change of the list L_1.

To conclude this section we note that this argument of knowledge can be made shorter if the lists L_1 and L_2 are first mapped to a pairing-friendly group (the mapping is done by proving equivalence between the pairing-friendly group elements and the original lists L_1 and L_2). We did not pursue this optimization here since we want to keep all arithmetic in the standard Bitcoin group to make the scheme more acceptable to the Bitcoin community.

E Proof of Solvency

Recall that in the definition of a privacy-preserving proof of solvency (Definition 2), $\mathcal{A}$ denotes a list of ordered pairs $\langle y_i = g_i^x, \mathrm{bal}(y_i)\rangle$ where y_i is the public key corresponding to a Bitcoin address with private key x_i and bal(y_i) is the amount of currency, or assets, observably spendable by this key on the blockchain. $\mathcal{L}$ denotes a list of integers l_j (which are taken as the amount of currency, or liabilities, belonging to a set of pseudonymous users).

Recall Theorem 3:

Theorem 1. *Provisions, as specified in Protocol 3, is a privacy-preserving proof of solvency.*

Proof. By Definition 2, proofs of the following Claims 1.1–1.5 imply Theorem 3 holds. □

Claim 1.1. ***(Solvency)*** *Protocol 3 ensures for all i and j:*

1. $\sum \mathrm{bal}(y_i) - \sum l_j = 0$
2. $0 \leqslant \mathrm{bal}(y_i) \leqslant \mathsf{MaxBTC}$

Proof. Toward proving Condition 1 of Claim 1.1, we note that as a corollary of the special soundness property of the proof of assets HVZKP captured by Theorem 1, Z_{Assets} is a commitment to $\sum \mathrm{bal}(y_i)$. Similarly by the special soundness of the proof of liabilities (Theorem 2), $Z_{\mathsf{Liabilities}}$ is a commitment to $\sum l_j$. By the property of homomorphic subtraction of Pedersen commitments, the value of $Z_{\mathsf{Assets}} \cdot {Z_{\mathsf{Liabilities}}}^{-1}$ computed in Protocol 3 is $\sum \mathrm{bal}(y_i) - \sum l_j$. The exact values of $\sum \mathrm{bal}(y_i)$ and $\sum l_j$ are extractable and thus an extractor can determine the truth of $\mathrm{bal}(y_i) - \sum l_j = 0$ for any malicious prover.

Condition 2 of Claim 1.1 is a corollary of the special soundness property of the proof of liabilities HVZKP captured by Theorem 2.

□

Claim 1.2. ***(Verification)*** *If user ID_k accepts T then $l_k \in \mathcal{L}$ and l_k is unique to ID_k; for all k.*

Proof. The first condition, $l_k \in \mathcal{L}$, is a corollary of the special soundness property of the proof of liabilities HVZKP captured by Theorem 2.

The second condition, that l_k is unique to ID_k, follows from the observation that l_k is unique to the commitment CID_k as they are published together by the exchange. The uniqueness of CID_k per ID_k is reduced directly to the binding property of the commitment scheme used to compute CID_k. In other words, if l_k was acceptable to user $\mathrm{ID}_{j\neq k}$, then either the exchange provided a different transcript that bound l_k to $\mathrm{CID}_{j\neq k}$ (which contradicts the property of broadcasting the transcript) or CID_k de-commits to $\mathrm{ID}_{j\neq k}$ (which contradicts the binding property of the commitment).

□

Claim 1.3. ***(Ownership)*** *Protocol 3 requires the exchange to be able to extract x_i for every $y_i \in \mathcal{A}$*

Proof. Claim 1.3 is a corollary of the special soundness property of the proof of assets HVZKP captured by Theorem 1. □

Claim 1.4. ***(User Privacy)*** *Protocol 3 ensures for user k, no non-negligible information is learned from T about $\langle \mathrm{ID}_{j\neq k}, l_{j\neq k}\rangle$ beyond Property 1.*

Proof. The first condition, $\mathrm{ID}_{j\neq k}$, of claim 1.4 can be reduced directly to the hiding property of the commitment scheme used to compute $\mathsf{CID}_{j\neq k}$.

The second condition, $l_{j\neq k}$, of claim 1.4 is a corollary of the zero-knowledge property of the proof of liabilities HVZKP captured by Theorem 2. □

Claim 1.5. ***(Exchange Privacy)*** *Protocol 3 ensures non-negligible information is learned about A and L beyond: $|A|$, $|L|$, and what is learned through Properties 1-4.*

Proof. Claim 1.4 is a fairly direct corollary of the zero-knowledge property of: the proof of assets HVZKP captured by Theorem 1, and the proofs of liabilities HVZKP captured by Theorem 2. This property extends to Protocol 3 as follows: Having called the simulator for both Z_{Assets} and $Z_{\mathsf{Liabilities}}$, a simulator for Protocol 3 computes the value $Z_{\mathsf{Assets}} \cdot {Z_{\mathsf{Liabilities}}}^{-1}$ and simulates it is a commitment to the value 0 using 'half' of the simulator in Figure 5 (*i.e.,* ignores r_1, c_1 and a_1 and sets $c_0 = c$).

□