Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges

Gaby G. Dagher
Concordia University

Benedikt Bünz
Stanford University

Joseph Bonneau (✉)*
Stanford University

Jeremy Clark
Concordia University

Dan Boneh
Stanford University

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 it offers practical computation times and proof sizes even for a large Bitcoin exchange with millions of customers.

Categories and Subject Descriptors

K.4.4 [**Electronic Commerce**]: Security, Cybercash, digital cash; E.3 [**Data Encryption**]: Public key cryptosystems

Keywords

Bitcoin; Exchange Services; Solvency; Zero Knowledge Protocols

1. INTRODUCTION

Digital currencies enable transactions that are electronically authorized, cleared and settled. After decades of research [7, 5, 2, 25] and failed business ventures attempting to establish a digital currency, Bitcoin [23] 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 [12]. Equipment failure, lost or stolen devices, or Bitcoin-specific malware [18] 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.

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 [22].

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 undermined Mt. Gox's privacy by revealing which 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

*Corresponding Author

CCS'15, October 12–16, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3832-5/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813674 .

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

[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.

by allowing users to independently verify they are in the dataset seen by the auditor, a step taken by Kraken[4] and OKCoin[5].

The notion of a cryptographic proof of liabilities, verifiable by any party with no trusted auditor, was first proposed by Maxwell [29], although this initial proposal leaks 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 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 is a straightforward use of a Merkle tree, a data structure well known by Bitcoin community, Provisions employs somewhat heavier cryptography not found in Bitcoin itself—*e.g.,* homomorphic 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, being computable by a single server in a few hours and requiring proofs which are less than 20 GB in size. Given this practicality and the strong privacy guarantees, 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 confidently which users will check, it runs a high risk of detection if it cheats (see Section 9.2).

Provisions is also limited to proving ownership of accounts with a full public key on the blockchain (not unused pay-to-pub-key-hash or pay-to-script-hash addresses which haven't yet be been used or multi-sig addresses). Removing this limitation is an interesting challenge for future work.

2. BACKGROUND

We assume the reader is familiar with Bitcoin [23]. Bonneau et al. [4] 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 inherent to a single block, representing one snapshot of the state of the system. In the remainder of the paper we leave implicit the proof will be 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 proves the total value of bitcoin it owes to each of its users. In the second, the *proof of assets*, the exchange proves the total value of bitcoin it has signing authority over. If the latter amount 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* [28] 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

[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`

[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.

[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.

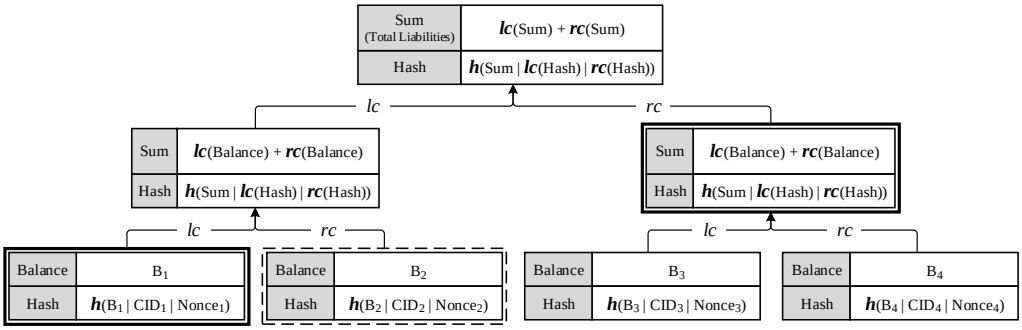

Figure 1: The Merkle tree from the Maxwell protocol [29] for proof of solvency. When a customer desires to verify their account (e.g. dashed line node), only two nodes need to be sent to the customer (bold line nodes).

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 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 as pooling funds ensures that it is not easy for outside observers to link deposits and withdrawals to the same individual [20].

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 [29]) 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 [21] 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 [15].

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.,* password-authenticated) 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 [29], providing a window for each customer to redeem their coins if desired. Again, we assume this is impractical for most 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 any two exchanges attempt to use the same bitcoin address in separate executions of Provisions, they 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 selects a large set of public keys $\mathbf{PK}$ from the blockchain that hold bitcoin to serve as an anonymity set for its own keys. 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, revealing to each user individually the random factors used to commit to the balance for their verification. For each committed balance, it also proves it is a small positive integer. These committed values are summed homomorphically to produce a commitment to the exchange's total liabilities.

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 [6] 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 $Y = xG$).

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 $\mathsf{bal}(y)$ to denote the balance associated with y. We assume $\mathsf{bal}(y)$ is an integer between 0 and MaxBTC for all y. We can represent any bitcoin account with $\mathsf{MaxBTC} = 2^{51}$—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, with ฿$1 = 10^8$ satoshis simply a convention to provide more human-friendly accounting units. 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 [26]. The commitment to a message $m \in \mathbb{Z}_q$ is defined as $\mathsf{com} = g^m \cdot h^r$ 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 [27] or the Chaum-Pedersen proof of representation of a Diffie-Hellman tuple [9], using Fiat-Shamir [13] 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 [16] 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 $\mathcal{E}$ chooses a set of Bitcoin public keys

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

that will serve as an anonymity set (we will discuss choosing this 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

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

Finally, it will be convenient to define

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

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 constructs 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

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

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

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

It remains to prove in zero-knowledge that Z_{Assets} is valid. To do so the exchange publishes a few additional auxiliary 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

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

which is a Pedersen commitment to the quantity $x_i \cdot s_i \in \mathbb{Z}_q$. By setting $\hat{x}_i = x_i \cdot s_i$ the equation can be written as

$$l_i = g^{\hat{x}_i} h^{t_i} \tag{4}$$

Now, to prove that Z_{Assets} is a commitment to the exchange's assets the exchange needs to prove that for every $i \in [1, n]$ it knows $s_i \in \{0, 1\},\ v_i, t_i, \hat{x}_i \in \mathbb{Z}_q$ satisfying conditions (1), (3), and (4). Z_{Assets} can then be computed according to (2).

The exchange proves knowledge of the required values using the Σ-protocol presented in Protocol 1 along with a Σ-protocol to prove that each s_i is binary and known to the exchange. Proving in zero-knowledge that a Pedersen commitment l_i is a commitment to a binary value is a standard zero-knowledge proof and is presented in full version of this paper [11] for completeness.

The protocol can be made non-interactive using the standard Fiat-Shamir heuristic. It therefore suffices to prove that the protocol is honest-verifier zero knowledge. This is captured in the following theorem:

Theorem 1. *The Σ-protocol in Protocol 1 is a honest-verifier zero knowledge proof of knowledge of quantities*

$$\mathsf{Assets} \textit{ and } (s_i \in \{0, 1\},\ \ v_i, t_i, \hat{x}_i \in \mathbb{Z}_q) \textit{ for } i \in [1, n]$$

that satisfy conditions (1),(2)*,* (3) *and* (4) *for all* $i \in [1, n]$.

The proof of Theorem 1 is given in Appendix A.

The proof of knowledge convinces the verifier that Z_{Assets} is a commitment to the exchange's total assets. More precisely, the verifier is convinced that

- Z_{Assets} is a commitment to $\sum_{i=1}^{n} s_i \cdot \text{bal}(y_i) \in \mathbb{Z}_q$ (by equation (2)), where $s_i \in \{0, 1\}$, and
- whenever $s_i = 1$ the exchange knows the corresponding private key $x_i \in \mathbb{Z}_q$. To see why observe that dividing equation (3) by (4) 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.

1. For $i \in [1, n]$
   (a) $\mathcal{E}$ chooses $u_i^{(1)}, u_i^{(2)}, u_i^{(3)}, u_i^{(4)} \xleftarrow{\$} \mathbb{Z}_q$.
   (b) The exchange $\mathcal{E}$ sends to the verifier:
   $$a_i^{(1)} = b_i^{u_i^{(1)}} h^{u_i^{(4)}}, \qquad a_i^{(2)} = y_i^{u_i^{(1)}}$$
   $$a_i^{(3)} = h^{u_i^{(2)}}, \quad a_i^{(4)} = g^{u_i^{(3)}}$$
   (c) The verifier replies with a challenge $c_i \xleftarrow{\$} Z_q$
   (d) $\mathcal{E}$ replies with:
   $$r_{s_i} = u_i^{(1)} + c_i \cdot s_i, \qquad \text{Response for } s_i$$
   $$r_{t_i} = u_i^{(2)} + c_i \cdot t_i, \qquad \text{Response for } t_i$$
   $$r_{\hat{x}_i} = u_i^{(3)} + c_i \cdot \hat{x}_i, \qquad \text{Response for } \hat{x}_i$$
   $$r_{v_i} = u_i^{(4)} + c_i \cdot v_i, \qquad \text{Response for } v_i$$
   (e) The verifier accepts if:
   $$b_i^{r_{s_i}} h^{r_{v_i}} \stackrel{?}{=} p_i^{c_i} a_i^{(1)} \qquad \text{Verify statement (1)}$$
   $$y_i^{r_{s_i}} h^{r_{t_i}} \stackrel{?}{=} l_i^{c_i} a_i^{(2)} a_i^{(3)} \qquad \text{Verify statement (3)}$$
   $$g^{r_{\hat{x}_i}} h^{r_{t_i}} \stackrel{?}{=} l_i^{c_i} a_i^{(3)} a_i^{(4)} \qquad \text{Verify statement (4)}$$
   (f) Run a zero knowledge proof (described in [11]) on l_i to prove knowledge of $s_i \in \{0, 1\}$
2. The verifier computes $Z_{\mathsf{Assets}} = \prod_{i=1}^{n} p_i$ Statement (2)

Protocol 1: Privacy-preserving proof of assets

That the proof is honest-verifier zero knowledge implies that nothing is revealed about the total assets, the s_i, or the x_i, as required.

Proof length. The proof is linear in the anonymity set size n, requiring about $13n$ elements in $\mathbb{Z}_q$. This is feasible even for large anonymity sets. We will discuss practical parameters in Section 10.

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). Perhaps it would be ideal if all entries would correspond to customers (*i.e.,* the mapping were surjective) however this property cannot be enforced—$\mathcal{E}$ can always add fake users to the list, but we ensure that doing so can only increase $\mathcal{E}$'s apparent liabilities.[9]

[9]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.

To verifiably compute 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:
   $$y_{i,k} = g^{x_{i,k}} h^{r_{i,k}}, \quad r_{i,k} \xleftarrow{\$} \mathbb{Z}_q$$
   (c) Compute a non-interactive proof of knowledge Π_i of all $r_{i,k}$ and $x_{i,k}$, and that every $x_{i,k}$ is binary (see [11]).
   (d) Compute a commitment to $\mathcal{C}_i$'s balance as $y_i = \prod_{k=0}^{m-1} (y_{i,k})^{(2^k)} \in G$.
   Then y_i is a Pedersen commitment to $\mathsf{Balance}_i$ because $y_i = g^{\mathsf{balance}_i} h^{r_i}$ where $r_i = \sum_{k=0}^{m-1} r_{i,k} \cdot 2^k$.
   (e) Compute a fresh customer identifier CID_i by committing $\mathcal{C}_i$'s username: choose a nonce $n_i \xleftarrow{\$} \{0,1\}^{256}$ and compute
   $$\mathsf{CID}_i = \mathsf{H}(\mathsf{username}_i \| n_i) \text{ where } \mathsf{H} \text{ is a cryptographically secure hash function such as SHA-256}$$
2. Homomorphically add the commitments to all customers balance into a single commitment to the total liabilities:
   $$Z_{\mathsf{Liabilities}} = \prod_{i=1}^{c} y_i$$
3. Publish the commitment to total liabilities $Z_{\mathsf{Liabilities}}$ and the list $\mathsf{LiabList}$ of all customers' tuples:
   $$\mathsf{LiabList} = \langle \mathsf{CID}_i, y_{i,0}, \ldots, y_{i,m-1}, \Pi_i \rangle \text{ for } i = 1, \ldots, c.$$
4. Every client $\mathcal{C}_i$, upon login, is privately given $\mathsf{username}_i, r_i$ and n_i. The client first computes $\mathsf{CID}_i = \mathsf{H}(\mathsf{username}_i \| n_i)$ and locates its tuple in $\mathsf{LiabList}$. The client then verifies that its balance is included in $Z_{\mathsf{Liabilities}}$ as follows:
   (a) compute $y_i = \prod_{k=0}^{m-1} (y_{i,k})^{(2^k)}$ and verify that $y_i = g^{\mathsf{balance}_i} h^{r_i}$,
   (b) verify that $Z_{\mathsf{Liabilities}} = \prod_{i=1}^{c} y_i$, and
   (c) verify the proof Π_i for $i = 1, \ldots, c$.

   Note that steps (b) and (c) can be carried out by any public auditor and need not be done by every client.

Protocol 2: Privacy-preserving proof of liabilities

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 [28]. To ensure an injective mapping, customers are provided an ID in line 1e which commits 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 email address of a potential customer from determining if that customer is in $\mathsf{LiabList}$ (or if a user is known to be a customer, which CID they correspond to).

The exchange can add arbitrary accounts to the list. However, as long as accounts can only add to the total liabilities (*e.g.,* $\mathcal{E}$ cannot commit to a negative balance and assign it to a fake user account), adding accounts is detrimental to a malicious $\mathcal{E}$'s goal as it could only increase 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 cause a reduction $\bmod q$ where $q \approx 2^{256}$ for our group $G =$secp256k1.

To enforce this, $\mathcal{E}$ provides a range proof (adapted from [19]) for each committed balance showing it is from a 'small' interval between 0 and $\mathsf{MaxBTC} = 2^{51}$. This makes it easy to ensure a modular reduction will never occur, as long as the exchange has fewer than 2^{205} accounts.

The range proof works by providing a bit-by-bit commitment of the account balance in binary representation, proving each bit is a 0 or 1 (using the proof of knowledge, mentioned above, twice with conjunctive logic [10]), and showing how many bits the number contains (an upper-bound on its maximum value). This committed binary representation is homomorphically converted into an integer and homomorphically summed.

In the full version of the paper [11], we prove the following theorem:

Theorem 2. *Protocol 2 is a honest-verifier zero knowledge proof of knowledge of quantities* Liabilities *and*

$$(x_{i,k} \in \{0,1\}, \;\; r_{i,k} \in \mathbb{Z}_q) \;\; \textit{for } i \in [1,c] \textit{ and } k \in [0, m-1]$$

that satisfy the condition

$$Z_{\mathsf{Liabilities}} = \prod_{i=1}^{c} y_i = \prod_{i=1}^{c} \prod_{k=0}^{m-1} (y_{i,k})^{(2^k)} = \prod_{i=1}^{c} \prod_{k=0}^{m-1} (g^{x_{i,k}} h^{r_{i,k}})^{(2^k)}$$

for all $i \in [1,c]$ *and* $k \in [0, m-1]$.

This step leads to the bulk of the proof size (see Section 10). In the full version of the paper [11], we discuss an alternate version of this protocol using zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) [3]. The proof generated by this

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 LiabList of its liabilities.
3. $\mathcal{E}$ computes $Z_{\mathsf{Assets}} \cdot {Z_{\mathsf{Liabilities}}}^{-1} = Z_{\mathsf{Assets-Liabilities}}$.
4. $\mathcal{E}$ proves in zero-knowledge that $Z_{\mathsf{Assets-Liabilities}}$ is a commitment to the value 0.

Protocol 3: Complete privacy-preserving proof of solvency

protocol is 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 step.

5.1 Customer verification

We assume that customers each verify LiabList to confirm the existence of their accounts and the correctness of their balances y_i and ID commitments CID_i. A malicious $\mathcal{E}$ which omits some customers will only be detected if at least one of those customers checks, although this is an inherent limitation given our privacy goals which require that only customers themselves can tell if their balance has been included or not. This limit applies equally, for example, to Maxwell's protocol.

The required checks from individual customers are fortunately 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:

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

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 y_i is indeed a commitment to their true account balance $\mathsf{Balance}_i$. This is shown in Step (4a) and is a simple calculation.

The other two verification steps, (4b) and (4c), 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.

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 $Z_{\mathsf{Assets-Liabilities}}$ is a commitment to 0 (line 4) is a simple Schnorr ZK proof of knowledge of the discrete log of $Z_{\mathsf{Assets-Liabilities}}$ to the base h, since $Z_{\mathsf{Assets-Liabilities}} = g^0 h^k$ for a value k known to the exchange and if $Z_{\mathsf{Assets-Liabilities}}$ 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 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 as an anonymity set.

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 is 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.

We extend Protocol 1 to force every exchange to also compute the list $\mathbf{L} = \{h^{\hat{x}_i} = h^{x_i \cdot s_i} \text{ for } i \in [1, n]\}$ which is randomly permuted and published. Note that when $s_i = 1$ the corresponding element in $\mathbf{L}$ is h^{x_i} and when $s_i = 0$ the corresponding element is simply $1 \in G$, the identity element. Thus $\mathbf{L}$ is a random permu-

tation of the exchange's Bitcoin public keys, but using the base h instead of g.

We require the exchange to prove that $\mathbf{L}$ is correctly constructed (*i.e.,* a permutation of $h^{\hat{x}_1}, \ldots, h^{\hat{x}_n}$) using a zero knowledge proof used as a component of the Neff mix net [24]. That zero-knowledge proof is used to prove that a given list $\ell_2 = \{h^{z_1}, \ldots, h^{z_n}\}$ is a permutation and base change of another given list $\ell_1 = \{g^{z_1}, \ldots, g^{z_n}\}$. This Neff proof thus proves that the published list $\mathbf{L}$ is constructed correctly. It is a simple and efficient proof, requiring $8n$ group elements (8 for each account) and $4n$ additional exponentiations during construction and verification.

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 h^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-trivial intersection.

Since every exchange is required to publish its list $\mathbf{L}$, an auditor can simply 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 $h^{\hat{x}_i} = h^{x_i \cdot s_i}$ 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 Bitcoin addresses beyond the number of addresses. 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 $\langle g, h, h^x \rangle$, 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 bit string of the same length.

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. $\mathsf{output}_{\mathcal{U}}^{\mathsf{ProveSolvency}}(1^k, \mathsf{MaxBTC}, \mathcal{A}', \text{ID}, \ell) \in \{\text{ACCEPT}, \text{REJECT}\}$

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 $Pr[\mathsf{out}_{\mathcal{U}} = \text{ACCEPT}] = 1$.*
2. ***Soundness.*** *If $\mathcal{A}$ and $\mathcal{L}$ are instead not a valid pair, or if $\mathcal{L}[ID] \neq \ell$, then $Pr[\mathsf{out}_{\mathcal{U}} = \text{REJECT}] \geqslant 1 - \nu(k)$.*
3. ***Ownership.*** *For all valid pairs $\mathcal{A}$ and $\mathcal{L}$, if $Pr[\mathsf{out}_{\mathcal{U}} = \text{ACCEPT}] = 1$, 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 the full paper [11]:

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

9. SECURITY DISCUSSION

9.1 Anonymity sets

Although Theorem 3 is true, in the case that the protocol extension of Section 7 is used, 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 [20], has demonstrated that Bitcoin's anonymity is limited (see [4] 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

Although Theorem 3 is true, it may fall short of an ideal level of user verification. Specifically, a proof of solvency *enables* user verification, but it does not guarantee that 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 [8]). 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 [17]. 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.

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 $13n$ 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 [4] 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. There are fewer than 500,000 addresses with a balance of more than 0.1 BTC, which collectively control 99.8% of all bitcoin.[10] 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 block chain), 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. Thus, we expect that fewer than a million addresses are available to be used in the anonymity set in practice 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.[11] 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,[12] 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 extensions would increase the size and construction time of the proof of assets by about $\frac{4}{13} \approx 30\%$. 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.

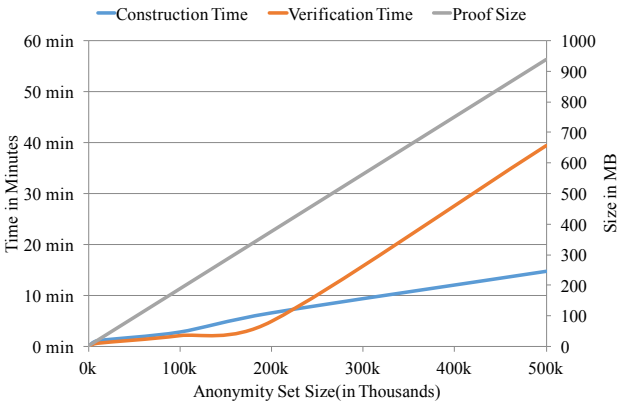

Figure 2: Performance for Protocol 1 (proof of assets).

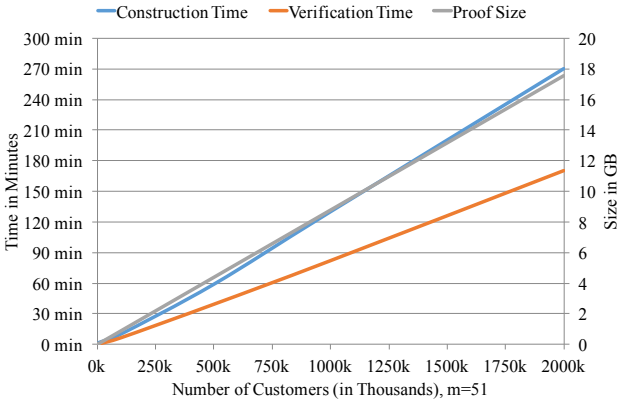

Figure 3: Performance for Protocol 2 (proof of liabilities).

11. CONCLUDING REMARKS

Stu Feldman has outlined a roadmap for technical maturity (as quoted in [14]):

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.

[10] https://bitinfocharts.com/top-100-richest-bitcoin-addresses.html
[11] https://www.coinbase.com/about
[12] https://www.bouncycastle.org/

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, D. Genkin, E. Tromer, and M. Virza. Snarks for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, 2013.

[4] 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.

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

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

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

[8] 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.

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

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

[11] G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges (full version). Technical report, IACR Cryptology ePrint Archive, 2015.

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

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

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

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

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

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

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

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

[20] 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.

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

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

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

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

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

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

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

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

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

APPENDIX

A. PROOF OF ASSETS IS HVZKP

Recall Theorem 1:

Theorem 1. *The Σ-protocol in Protocol 1 is a honest-verifier zero knowledge proof of knowledge of quantities*

$$\mathsf{Assets} \textit{ and } (s_i \in \{0,1\},\ \ v_i, t_i, \hat{x}_i \in \mathbb{Z}_q) \textit{ for } i \in [1,n]$$

that satisfy conditions (1),(2), (3) *and* (4) *for all* $i \in [1,n]$.

In other words, Protocol 1 is honest-verifier zero knowledge of the following relation:

$$\mathsf{PoK}\{(\mathsf{Assets}, s_i \in \{0,1\}, v_i, t_i, \hat{x}_i) : g, h, y_i, \mathrm{bal}(y_i), l_i, p_i\}$$

Proof. By the definition of HVZKP (see full paper [11]), proofs of the following Claims 1.1, 1.2, and 1.3 imply Theorem 1 holds. □

Claim 1.1. ***(Completeness)*** *If $\mathcal{P}$ and a verifier $\mathcal{V}$ follow protocol 1 on input $(g, h, y_i, bal(y_i), l_i, p_i)$ and private input $(\mathsf{Assets}, s_i \in \{0,1\}, v_i, t_i, \hat{x}_i)$, 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. $\mathcal{E}$ can then for any random $u_i^{(\cdot)}$ and challenges c_i compute all responses $r_{(\cdot)}$. Completeness follows because for $i \in [1, n]$:

$$\begin{aligned} b_i^{r_{s_i}} h^{r_{v_i}} &= b_i^{u_i^{(1)}} b_i^{c_i \cdot s_i} h^{u_i^{(4)}} h^{c_i \cdot v_i} \\ &= p_i^{c_i} a_i^{(1)} \\ y_i^{r_{s_i}} h^{r_{t_i}} &= y_i^{u_i^{(1)}} y_i^{c_i \cdot s_i} h^{u_i^{(2)}} h^{c_i \cdot t_i} \\ &= l_i^{c_i} a_i^{(2)} a_i^{(3)} \\ g^{r_{\hat{x}_i}} h^{r_{t_i}} &= g^{u_i^{(3)}} g^{c_i \cdot \hat{x}_i} h^{u_i^{(2)}} h^{c_i \cdot t_i} \\ &= l_i^{c_i} a_i^{(3)} a_i^{(4)} \end{aligned}$$

□

Claim 1.2. ***(Soundness)*** *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 $c_i \neq c_i'$, E can compute $(v_i, s_i, t_i, \hat{x}_i)$.*

Proof. We show in Figure 4 that there exists an extractor E for all efficient provers $\mathcal{P}^\star$ that convince the verifier. Note that

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

$$y_i^{r_{s_i} - r'_{s_i}} h^{r_{t_i} - r'_{t_i}} = l_i^{c - c'}$$

and

$$g^{r_{\hat{x}_i} - r'_{\hat{x}_i}} h^{r_{t_i} - r'_{t_i}} = l_i^{c - c'}$$

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

Additionally in the full version [11], we provide an extractor and simulator for a HVZKP proof that a committed value is binary. Were this extractor run on l_i, it can extract a binary s_i.

□

Claim 1.3. ***(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 5. Note that both the original a's as well as the simulated a's are distributed uniformly at random in G given that the challenges 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

□

1. For $i \in [1, n]$
   (a) Run $P^\star$ to obtain $a_i^{(1)}, a_i^{(2)}, a_i^{(3)}, a_i^{(4)}$
   (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
   $$\begin{aligned} b_i^{r_{s_i}} h^{r_{v_i}} &= p_i^{c_i} a_i^{(1)} \\ y_i^{r_{s_i}} h^{r_{t_i}} &= l_i^{c_i} a_i^{(2)} a_i^{(3)} \\ g^{r_{\hat{x}_i}} h^{r_{t_i}} &= l_i^{c_i} a_i^{(3)} a_i^{(4)} \end{aligned}$$
   (d) Rewind $P^\star$ to right after step 1b of the protocol.
   (e) Send $c' \xleftarrow{\$} Z_q \backslash \{c\}$ to $P^\star$
   (f) $P^\star$ will output $r'_{s_i}, r'_{t_i}, r'_{\hat{x}_i}, r'_{v_i}$ such that
   $$\begin{aligned} b_i^{r'_{s_i}} h^{r'_{v_i}} &= p_i^{c'_i} a_i^{(1)} \\ y_i^{r'_{s_i}} h^{r'_{t_i}} &= l_i^{c'} a_i^{(2)} a_i^{(3)} \\ g^{r'_{\hat{x}_i}} h^{r'_{t_i}} &= l_i^{c'} a_i^{(3)} a_i^{(4)} \end{aligned}$$
   (g) Output
   $$\begin{aligned} v_i &= \frac{r_{v_i} - r'_{v'_i}}{c_i - c'_i} && \bmod q \\ s_i &= \frac{r_{s_i} - r'_{s_i}}{c_i - c'_i} && \bmod q \\ t_i &= \frac{r_{t_i} - r'_{t_i}}{c_i - c'_i} && \bmod q \\ \hat{x}_i &= \frac{r_{\hat{x}_i} - r'_{\hat{x}_i}}{c_i - c'_i} && \bmod q \end{aligned}$$

Figure 4: Extractor for Proof of Assets protocol

1. For each $i \in [1, n]$
   (a) Choose $r_{s_i}, r_{t_i}, r_{\hat{x}_i}, r_{v_i}$ uniformly at random from $\mathbb{Z}_q$
   (b)
   - Let $a_i^{(1)} = b_i^{r_{s_i}} h^{r_{v_i}} p_i^{-c_i}$
   - Let $a_i^{(2)} = y_i^{r_{s_i}}$
   - Let $a_i^{(3)} = h^{r_{t_i}} l_i^{-c_i}$
   - Let $a_i^{(4)} = g^{r_{\hat{x}_i}}$

   (c) Publish $(a_i^{(1)}, a_i^{(2)}, a_i^{(3)}, a_i^{(4)};\ c_i\ ; r_{s_i}, r_{t_i}, r_{\hat{x}_i}, r_{v_i})$ as the transcript.

Figure 5: Simulator for Proof of Assets protocol