Instant Zero Knowledge Proof of Reserve

Xiang Fu[0000−0002−6608−1654]
Xiang.Fu@hofstra.edu, Hofstra University

July 26, 2023

Keywords: Proof of Reserve, Proof of Asset, Solvency, Lookup Argument, Zero Knowledge, Aggregated Non-Membership Proof, Zero Knowledge Set Disjoint Proof

Abstract

We present two zero knowledge protocols that allow one to assert solvency of a financial organization *instantly* with high throughput. The scheme is enabled by the recent breakthrough in lookup argument, i.e., after a pre-processing step, the prover cost can be independent of the lookup table size for subsequent queries. We extend the $\mathfrak{cq}$ protocol [EFG22] and develop an aggregated non-membership proof for zero knowledge sets. Based on it, we design two instant proof-of-reserve protocols. One is non-intrusive, which works for crypto-currencies such as BTC where transaction details are public. It has $O(n \log(n))$ prover complexity and $O(1)$ proof size/verifier complexity, where n is the number of transactions assembled in a cycle. The other works for privacy preserving platforms where the blockchain has no knowledge of transaction details. By sacrificing non-intrusiveness, the second protocol achieves $O(1)$ complexity for both the prover and verifier.

1 Introduction

First defined in Provisions [DBB$^+$15], the zero knowledge (zk) solvency problem is to prove in zero knowledge that the *asset* of an organization is greater than its *liability*. Due to the volatility of financial market, we are interested in a variation called *instant zk-solvency* problem, i.e., to prove solvency of an organization instantly, e.g., at the frequency of $1Hz$. This seems like a daunting task that requires very expensive computing resources. For instance, in [DBB$^+$15] to generate and then verify a single zk-solvency proof for an anonymity set size of $500k$ BTC addresses needs about 1 hour. In our work, we are using the entire 80 million BTC addresses as the anonymity set, and the design goal is to support 2000 transactions per second (TPS). We build our protocols over the recent breakthrough in lookup arguments [ZBK$^+$22, PK22, GK22, ZGK$^+$22, EFG22]: if the container table is preprocessed, then all lookup arguments can be generated at a cost independent of the lookup table size. We assume that the registration and verification of liability is handled by frameworks such as DAPOL(+) [CLMN20, JC21], then we just need to focus on the instant zk-proof for asset only. In the rest of the paper we use terms "proof-of-reserve" and "proof-of-asset" interchangeably.

We first consider the zk-asset protocols for open cryptocurrencies such as BTC where transaction details are public. Here, we abstract away platform specifics and use "account" to denote e.g., BTC address. We observe that an organization can pre-compute a fixed set $\mathbf{T}$ of accounts that includes both those currently in use and those to be used in future.[1] We will leverage this fact later in applying lookup arguments. At each transaction cycle, the market discloses two vectors $\mathbf{a}$ and $\boldsymbol{\Delta}$ where $\mathbf{a}_i$ is an account and $\boldsymbol{\Delta}_i$ is its balance change. Given the Pedersen commitments to $\mathbf{a}$, $\boldsymbol{\Delta}$, and $\mathbf{T}$, we present a zk-proof π_{ASSET1}, which convinces the verifier that $\mathbf{C}_v$ commits to a value v that accumulates the balance change of those accounts in the intersection of $\mathbf{a}$ and $\mathbf{T}$ as sets.

The main challenge here is the concrete performance, as for the state of the art, to prove 2^{20} R1CS constraints needs about 10^1 to 10^2 seconds for the vast majority of proof systems. A recent research line in lookup arguments [GW20, ZBK$^+$22, PK22, GK22, ZGK$^+$22, Hab22, EFG22] has shed a light. For the most recent protocol $\mathfrak{cq}$, after running a preprocessing of $O(N\mathsf{log}(N))$ on $\mathbf{T}$, the prover cost is $O(n\mathsf{log}(n))$ field and $O(n)$ group operations to show that all elements of a table of size n belong to $\mathbf{T}$. Built upon the $\mathfrak{cq}$ protocol, we are able to develop an aggregated non-membership proof for zero knowledge sets, i.e., to show that the intersection of two committed zk-sets is empty. Then, the account set $\mathbf{a}$ can be split into two disjoint sets $\mathbf{a} \cap \mathbf{T}$ and $\mathbf{T} - \mathbf{a}$, as sets. Balance can be accumulated over $\mathbf{a} \cap \mathbf{T}$ only. The first protocol has $O(n\mathsf{log}(n))$ prover complexity and $O(1)$ proof size.

The asset proof for privacy preserving cryptocurrencies such as Monero [YSL$^+$] and ZeroCoin [MGGR13] is more delicate, as the blockchain itself may only have partial or no knowledge of transaction details at all. In the past, there are platform specific asset proof protocols such as [DV19, DBV21], but they are costly in proof generation and verification. Our solution is orthogonal to the platform and is efficient and instant. The basic idea is to require each transaction participant to submit an additional zk-proof correlating a hiding commitment of the organization ID and balance change to the transaction request. Then the blockchain can collect the two tables of commitments of IDs and balance changes, make them public and generate vector commitments to them. These two tables are preprocessed and treated as lookup table. Then the prover runs a 1-query lookup argument against these tables. This results a protocol costs $O(1)$ at both the prover and verifier.

[1] $\mathbf{T}$ can be expanded periodically.

2 Preliminaries

Let $\mathcal{G}$ be a generator of bilinear groups, i.e., $(p, \mathbf{g}_1, \mathbf{g}_2, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$. Here $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ all have prime order p, with $\mathbf{g}_1$ ($\mathbf{g}_2$) as the generator of $\mathbb{G}_1$ ($\mathbb{G}_2$). $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is the bilinear map s.t. for any $a, b \in \mathbb{Z}_p$: $e({\mathbf{g}_1}^a, {\mathbf{g}_2}^b) = e(\mathbf{g}_1, \mathbf{g}_2)^{ab}$ and $e(\mathbf{g}_1, \mathbf{g}_2)$ is the generator of $\mathbb{G}_T$. Following the notations in Groth16 [Gro16], we write $\mathbb{G}_1$ and $\mathbb{G}_2$ as additive groups. That is: given $a \in \mathbb{Z}_p$, we denote ${\mathbf{g}_1}^a$ as $[a]_1$, and similarly are group elements in $\mathbb{G}_2$ and $\mathbb{G}_T$ denoted. For instance, ${\mathbf{g}_1}^a {\mathbf{g}_1}^b$ is written as $[a]_1 + [b]_1$ or $[a+b]_1$, $({\mathbf{g}_2}^a)^b$ as $[ab]_2$, and $e({\mathbf{g}_1}^a, {\mathbf{g}_2}^b)$ is denoted as $[a]_1 \cdot [b]_2$ or $[ab]_T$.

We use $[n]$ to denote range $[1, n]$. We use bold symbols to denote a vector, e.g., $\mathbf{T} \in \mathbb{F}^n$ is written as $(\mathbf{T}_1, \ldots, \mathbf{T}_n)$. Given two vectors of the same size, $\mathbf{u} + \mathbf{v}$ is $(\mathbf{u}_1 + \mathbf{v}_1, \ldots, \mathbf{u}_n + \mathbf{v}_n)$, and $\alpha\mathbf{u}$ is $(\alpha\mathbf{u}_1, \ldots, \alpha\mathbf{u}_n)$. Let $\mathbf{t} \in \mathbb{F}^n$ and $\mathbf{T} \in \mathbb{F}^N$, a lookup argument for $\mathbf{t} \mathrel{\hat{\subset}} \mathbf{T}$ asserts that for each $i \in [n]$: $\mathbf{t}_i \in \mathbf{T}$. Note that both $\mathbf{t}$ and $\mathbf{T}$ may contain duplicates, and they are not mathematical sets. When the context clear, ν and ω are n'th and N'th root of unity, i.e., $\nu^n = 1$ and $\omega^N = 1$. We define $\mathbb{V} = \{\omega^1, \ldots, \omega^N\}$ and $\mathbb{H} = \{\nu^1, \ldots, \nu^n\}$. Let $\{L_i(X)\}_{i=1}^N$ be the Lagrange base polynomials s.t. for each $i \in [n] : L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0$ for $j \neq i$. Similarly, the Lagrange polynomials for n is denoted as $l_i(X)$.

Given $\mathbf{T}$, we use the same symbol (with regular font) to denote the correspondingly polynomial $T(X) = \sum_{i=1}^N \mathbf{T}_i L_i(X)$. Similarly, given $\mathbf{t}$, $t(X) = \sum_{i=1}^n \mathbf{t}_i \cdot l_i(X)$. Given a KZG commitment [KZG10] key $\left([s]^i\right)_{i=1}^N$, the KZG commitment to $T(X)$ is $[T(s)]_1$.

Given a multi-set S, its vanishing polynomial $Z_S(X)$ is defined as $Z_S(X) = \prod_{s \in S}(X - s)$. It is known that $Z_{\mathbb{V}}(X) = X^N - 1$ and $Z_{\mathbb{H}}(X) = X^n - 1$.

2.1 Lookup Argument

We need a homomorphic and zero knowledge look-up argument, and a slight zk-enhancement of $\mathfrak{cq}$ [EFG22] satisfies our needs. We denote it as $\pi_{\texttt{ZK\_LOOKUP}}$.

1. $\mathsf{srs} \leftarrow \mathsf{Setup}\ (N, \lambda)$: a trusted set-up given security parameter λ and container table size limit N, samples bilinear groups and generates common reference string srs. We here do not distinguish between prover and verifier keys, however, in general verifier key is much smaller.

2. $(\mathsf{aux}_{\mathbf{T}}, \mathbf{C}_{\mathbf{T}}) \leftarrow \texttt{Preprocess}(\mathbf{T})$: Given the container table $\mathbf{T}$, it generates $\mathsf{aux}_{\mathbf{T}}$ (the preprocessed information) and $\mathbf{C}_{\mathbf{T}}$ (a vector commitment to $\mathbf{T}$).

3. $\pi \leftarrow \texttt{Prove}(\mathsf{srs}, \mathsf{aux}_{\mathbf{T}}, \mathbf{T}, \mathbf{t})$: produces a zero knowledge proof π for $\mathbf{t} \mathrel{\hat{\subset}} \mathbf{T}$.

4. $1/0 \leftarrow \texttt{Verify}(\mathsf{srs}, \mathbf{C}_{\mathbf{T}}, \mathbf{C}_{\mathbf{t}}, \pi)$: checks that the zk-aux-lookup argument is valid.

5. $\pi \leftarrow \texttt{FoldProve}(\mathsf{srs}, \mathsf{aux}_{\mathbf{U}}, \mathbf{U}, \mathsf{aux}_{\mathbf{V}}, \mathbf{V}, \mathbf{u}, \mathbf{v}, \alpha)$: produces a zero knowledge proof π for $\mathbf{u} + \alpha\mathbf{v} \mathrel{\hat{\subset}} \mathbf{U} + \alpha\mathbf{V}$.

The definition for completeness, knowledge soundness and zero knowledge is standard and will be presented in a full-version of this manuscript. The difference from a zero knowledge subset proof is that $\mathbf{t}$ is a multi-set (allowing repetitions). Also the `FoldProve` operation allows to easily prove multi-column tables exploiting the homomorphic property of `cq`. There are two options to provide zk. One way is bounded leaky-zk [CHM+20, CGG+23] where the degree of a polynomial is increased by k if it is known that it will be evaluated at most $k-1$ times. Another alternative is to blind each KZG commitment with a random factor, which increases the proof with $O(1)$ size extra Schnorr style proof. In the rest of the paper, for convenience of presentation, all protocols can be enhanced with zk, but we omit the details.

2.2 Batched Range Proof

We use $\pi_{\texttt{RANGE}}(\mathbf{C_a}, B)$ to denote a batched zk-range proof which asserts that each $\mathbf{a}_i$ in a vector commitment $\mathbf{a}$ is in range $[0, 2^B)$.

One possible $O(1)$ proof size construction is to build it over lookup argument. For example, given an array $\mathbf{a}$, if one wishes to assert that each $\mathbf{a}_i$ is a valid BTC address (a 160-bit number), a pre-processed table of all 32-bit numbers can be constructed, and $\mathbf{a}_i$ can be split into 5 chunks with each chunk to be asserted a valid 32-bit number. [2]

3 Positive-Negative Lookup

The goal is to show that a smaller table $\mathbf{t}$ does not have any common elements with a preprocessed lookup table $\mathbf{T}$. Here the elements of $\mathbf{t}$ are "unexpected" (e.g., new BTC addresses in a transaction cycle), thus aggregation of existing non-membership in [SKBP22] does not apply. The argument will be based on lookup and have the same asymptotic complexity.

We will in fact provide a more expressive construction called Zero Knowledge Positive-Negative Lookup argument ("zk-PN-Lookup" for short and denoted as $\pi_{\texttt{PN\_LOOKUP}}(\mathbf{C_T}, \mathbf{C_t}, \mathbf{C_o})$. It asserts that the Pedersen vector commitment $\mathbf{C_o}$ encodes a Boolean array that for each $i \in [|\mathbf{t}|]$: $o_i = \mathbf{t}_i \in \mathbf{T}$.

The basic idea is straight-forward. Let $\mathbf{T}$ be a *sorted* vector of distinct elements in ascending order, and $N = |\mathbf{T}|$. Let $\mathbf{T}' = \{\mathbf{T}_2, \ldots, \mathbf{T}_N\}$. There are two cases to consider: (1) $\mathbf{t}_i \in \mathbf{T}$. This is the standard lookup argument, and (2) $\mathbf{t}_i \notin \mathbf{T}$. Then the prover provides $\mathbf{T}_j$ and $\mathbf{T}_{j+1}$ s.t. $\mathbf{T}_j < \mathbf{t}_i < \mathbf{T}_{j+1}$. Given that $\mathbf{T}_{j+1} = \mathbf{T}'_j$, a folded lookup argument can accomplish the job.

Formally, we assume that all elements in lookup tables are in range $[0, 2^B)$, e.g., for BTC, B is 160. Given a private and sorted table $\mathbf{S} = \{\mathbf{S}_1 < ... < \mathbf{S}_{N-1}\}$ with all elements less than 2^B. Define $\mathbf{T} = \{0, \mathbf{S}_1, \ldots, \mathbf{S}_{N-1}\}$, and $\mathbf{T}' = \{\mathbf{S}_1, \ldots, \mathbf{S}_{N-1}, 2^B\}$. We call $(\mathbf{T}, \mathbf{T}')$ the *sorted vector pair* for $\mathbf{S}$ with

[2] Some adaptation of the polynomial and pairing based range proof is also a possibility. We will explore this option later.

bound 2^B. Their relation can be proved with a separate zk-proof. Let $\mathbf{t}$ be the lookup query of size n. The $\pi_{\texttt{PN\_LOOKUP}}$ is defined as below.

```
$\pi_{\texttt{PN\_LOOKUP}}(\mathbf{C_T}, \mathbf{C_{T'}}, \mathbf{C_t}, \mathbf{C_o}, B)\{$
    $(\mathbf{T}, \mathbf{T'}, \mathbf{t}, \mathbf{o}) : \forall i \in [n] :$
      $\mathbf{C_t} = [t(s)]_1 \ \wedge\ \mathbf{C_o} = [o(s)]_1 \ \wedge$
      $(o_i = 1 \ \wedge\ \exists j \ s.t.\ \mathbf{T}_j = \mathbf{t}_i) \ \vee\ (o_i = 0 \ \wedge\ \exists j \ s.t.\ \mathbf{T}_j < \mathbf{t}_i < \mathbf{T'}_j)$
$\}$
```

It is clear a $\pi_{\texttt{PN\_LOOKUP}}$ can be generated if and only if all elements of $\mathbf{t}$ are in range $[0, 2^B)$. Let KZG keys be $\left([x^i]_1\right)_{i=0}^{N}$. The protocol is presented below. We did not present the addition of zk for simplicity of notations, but it can be achieved using the techniques mentioned earlier.

1. $\mathbf{P}$ computes location vector $\mathbf{s}$ of size n for $\mathbf{t}$:

$$\mathbf{s}_i = \begin{cases} \text{j} & \text{if } \mathbf{T}_j = \mathbf{t}_i \\ \text{k} & \text{if } \mathbf{T}_k < \mathbf{t}_i < \mathbf{T'}_k \end{cases}$$

   and computes $\mathbf{o}$ s.t.

$$\mathbf{o}_i = \begin{cases} 1 & \text{if } \mathbf{T}_j = \mathbf{t}_i \\ 0 & \text{if } \mathbf{T}_k < \mathbf{t}_i < \mathbf{T'}_k \end{cases}$$

   Then $\mathbf{P}$ computes table $\mathbf{u} = (\mathbf{T}_{\mathbf{s}_i})_{i=1}^{n}$ and $\mathbf{v} = (\mathbf{T'}_{\mathbf{s}_i})_{i=1}^{n}$.

   $\mathbf{P} \to \mathbf{V} : ([u(s)]_1, [v(s)]_1)$.

2. $\mathbf{V} \to \mathbf{P} : \alpha \in \mathbb{F}$.

3. $\mathbf{P}$ computes $\pi \leftarrow \texttt{FoldProve}(\mathsf{srs}, \mathsf{aux}_{\mathbf{T}}, \mathbf{T}, \mathsf{aux}_{\mathbf{T'}}, \mathbf{T'}, \mathbf{u}, \mathbf{v}, \alpha)$. $\mathbf{P} \to \mathbf{V} : \pi$.

4. $\mathbf{V}$ aborts if $\texttt{Verify}(\mathsf{srs}, \mathbf{C_T} + \alpha\mathbf{C_{T'}}, \mathbf{C_u} + \alpha\mathbf{C_v}, \pi)$ returns 0.

5. $\mathbf{P}$ and $\mathbf{V}$ run $\pi_{\texttt{RANGE}}(\mathbf{C_t}/\mathbf{C_u}, B)$, and $\pi_{\texttt{RANGE}}(\mathbf{C_v}/\mathbf{C_t}, B)$.

6. $\mathbf{P}$ shows $\mathbf{o}$ is a Boolean array by proving that there exists a $q_o(X)$ s.t.

$$o(X)(o(X) - 1) = q_o(X)z_{\mathbb{H}}(X)$$

7. $\mathbf{P}$ proves that $o(X)$ is a correct output, i.e., when $\mathbf{t}_i > \mathbf{u}_i$, the value of $\mathbf{o}_i$ is set to 1, otherwise 0.

   This is accomplished by the following: $\mathbf{P}$ computes $\mathbf{d} = \mathbf{t} - \mathbf{u}$. Then $\mathbf{P}$ shows that there exist $q(X)$ and $v(X)$ (encoding the inverse of of each $\mathbf{d}_i$ if exists) s.t.

$$(1 - o(X))d(X) + o(X)(v(X)d(X) - 1) = q(X)z_{\mathbb{H}}(X)$$

$\pi_{\texttt{PN\_LOOKUP}}$ immediately leads to aggregated non-membership proof. Given a preprocessed $\mathbf{C_T}$, to prove that $\mathbf{t} \cap \mathbf{T} = \emptyset$ involves running $\pi_{\texttt{PN\_LOOKUP}}$ first and then showing that $\mathbf{C_o}$ is a Pedersen vector commitment to $(0)_{i=1}^{n}$.

Efficiency: The asymptotic complexity is the same of the construction of $\pi_{\texttt{ZK\_LOOKUP}}$. Using cq as the underlying homomorphic lookup argument, the prover complexity is $O(n\mathsf{log}(n))$ and verifier complexity is $O(1)$. Concretely, the main protocol needs 8n $\mathbb{G}_1$ operations. The batch range proofs need about 80n $\mathbb{G}_1$ operations (2 range proofs and each costing 5 lookups). Altogether it needs about $100n$ $\mathbb{G}_1$ operations. Verification can be batched to reduce pairings. The proof size is $O(1)$. Concretely it costs about about 100 $\mathbb{G}_1$ elements (this is about $4kb$). [3]

4 Non-Intrusive Protocol $\pi_{\texttt{ASSET1}}$

For markets that are "regulated", e.g., banks that require explicit identification of accounts, the asset proof in zero knowledge is simple and low cost, because the bank can directly run a database query and post a Pedersen commitment to the total asset of each client. We now consider the asset proof for typically cryptocurrencies where customer identity (instead of their accounts) and customer's ownership of accounts are typically hidden.

We first introduce the $\pi_{\texttt{ASSET1}}$ protocol. It is *non-intrusive* in the sense that there is no changes needed on the cryptocurrency blockchain, or any participants who have no need for asserting assets. The protocol assumes that the blockchain has information of transaction details such as sender/receiver accounts (note: not client identity) and transaction amount. It is applicable to platforms such as BTC and ETH.

We assume that each organization possess a fixed set $\mathbf{S}$ of accounts (e.g., BTC wallet addresses), including which are being used and those to be used in the future. This set can be expanded periodically. Let $\mathbf{T}$ and $\mathbf{T}'$ be the sorted vector pair defined for $\mathbf{S}$ of bound B (e.g., $B = 160$ for BTC). A Pedersen vector commitment over Lagrange bases is generated for $\mathbf{T}$. More formally, let $T(X) = \sum_{i=1}^{N} \mathbf{T}_i L_i(X)$, and $\mathbf{C_T} = \sum_{i=1}^{N} \mathbf{T}_i [L_i(s)]_1$.[4] At bootstrap, the organization needs to provide an ownership proof, asserting the knowledge of the secret key to each of the accounts in $\mathbf{T}$. For instance, for BTC, combinations of Σ-protocols and zkSNARK systems can be used as shown in [AGM18]. It is expensive, however, the proof is only needed once. Then $\mathbf{T}$ and $\mathbf{T}'$ will be preprocessed and used as the lookup table for instant zk-asset proofs.

For each transaction period, the blockchain makes two vectors public: $\mathbf{a}$: the list of accounts, and $\boldsymbol{\Delta}$, the corresponding balance change on each account. We use $[0, 2^B)$ as positive values and $[|\mathbb{F}| - 2^B, |\mathbb{F}|)$ as negative values. Let $n = |\mathbf{a}|$,

[3] The batched range proof, although $O(1)$ proof size, accounts for the majority of prover/verifier cost. For the design goal of 2000 TPS, this results in $400k$ $\mathbb{G}_1$ exponentiations (which is almost 50 seconds of CPU time for prover). We still need to improve range proof efficiency. A potential direction is polynomial and pairing based range proof.

[4] In practice, the hiding is achieved by appending a blinding factor $r[h]_1$. We omit all zk-related details in the presentation.

and $N = |\mathbf{T}|$. Typically, the value of n is not large and determined by the TPS of the platform. For instance, BTC, ETH, Visa Network and NYSE operate at 7, 30, 2000, and 24000 TPS, respectively. In this work, we aim at accomplishing 2000 TPS on a regular desktop computer. We set $n = 2048$ and $N = 8$ million.

Define $\mathbf{C_a} = [a(s)]_1$ (without blinding factor) and similarly is $\mathbf{C}_{\boldsymbol{\Delta}}$ defined. They are used only as succinct representation of $\mathbf{a}$ and $\boldsymbol{\Delta}$ and need to be publicly computable (thus no hiding property is needed). On the other hand, $\mathbf{C_T}$ and $\mathbf{C_{T'}}$ are published as the permanent commitment to the fixed set of accounts of the prover of $\pi_{\texttt{ASSET1}}$. They need to be hiding.

Intuitively, $\pi_{\texttt{ASSET1}}$ states that $\mathbf{C}_v$ commits to a value v that is the sum of balance changes for all accounts that appear in the intersection of $\mathbf{T}$ and $\mathbf{a}$ as sets. It is formally defined below.

$$\begin{aligned}
&\pi_{\texttt{ASSET1}}(B, \mathbf{C_T}, \mathbf{C_{T'}}, \mathbf{C_a}, \mathbf{C}_{\boldsymbol{\Delta}}, \mathbf{C}_v)\{(\mathbf{T}, \mathbf{T}', \mathbf{a}, \boldsymbol{\Delta}, v, r) : \\
&\quad \mathbf{C}_v = [v] + r[h]_1 \ \wedge \ v = \sum_{\mathbf{a}_j \in \mathbf{T} \cap \mathbf{a}} \Delta_j \ \wedge \\
&\quad \mathbf{C_a}, \mathbf{C}_{\boldsymbol{\Delta}} \text{ are commitments to } \mathbf{a}, \boldsymbol{\Delta} \\
&\}
\end{aligned}$$

The protocol is surprisingly simple with $\pi_{\texttt{PN\_LOOKUP}}$. Let $\mathbf{C_o}$ be the Pedersen commitment to the output Boolean array $\mathbf{o}$ of $\pi_{\texttt{PN\_LOOKUP}}$, which indicates if $\mathbf{a}_i$ appears in $\mathbf{T}$. We simply declare an accumulator polynomial $u(X)$ s.t. $u_{i+1} = u_i + \Delta_i$ if $\mathbf{o}_i = 1$.

1. $\mathbf{P}$ and $\mathbf{V}$ run $\pi_{\texttt{PN\_LOOKUP}}(\mathbf{C_T}, \mathbf{C_{T'}}, \mathbf{C_a}, \mathbf{C_o}, B)$.

2. Define polynomial $u(X) = u_i \cdot l_i(X)$ s.t.

$$\begin{aligned}
&u(\nu^0) = 1 \\
&u(\nu^{n-1}) = v \\
&u(X\nu) - u(X) - o(X)\Delta(x) = q(X)z_{\mathbb{H}}(X)/((X - \nu^{n-1})(X - 1))
\end{aligned}$$

$\mathbf{P}$ sends $[u(s)]_1$ and proves the above relation where $\mathbf{C}_v$ commits to v.

The protocol can be enhanced for zk, and the details are omitted here.

Efficiency: The prover complexity is $O(n\mathsf{log}(n))$. Proof size and verifier complexity are both $O(1)$. The concrete efficiency will be slightly greater than $\pi_{\texttt{PN\_LOOKUP}}$.

5 Intrusive Protocol $\pi_{\texttt{ASSET2}}$

We then introduce the "intrusive" protocol $\pi_{\texttt{ASSET2}}$, which works for cryptocurrencies such as Monero [YSL$^+$] and ZeroCoin [MGGR13], where the blockchain itself has no (or partial) knowledge of the sender/receiver account and the transaction amount. As the blockchain is unable to publish the aggregated and succinct transaction information, the change of blockchain system and participants is unavoidable, to support proof of assets. In this sense, the protocol is "intrusive". Note that π_{ASSET2} also works with open platforms such as BTC and ETC, if platform change is allowed. By sacrificing non-intrusiveness, the protocol is more efficient than π_{ASSET1} asymptotically and concretely. By leveraging preprocessing of lookup arguments (more exactly vector commitment open proof), the protocol's prover and verifier complexity are both $O(1)$.[5]

In the past, there are platform specific zk-asset proofs such as [DV19, DBV21] for Monero [YSL$^+$], and [DV21] for MimbleWimble. Compared with the aforementioned work, our protocol is orthogonal to the cryptocurrency platform, i.e., its core framework does not depend on platform's transaction protocols. Instead, we need each platform to add a zk-proof component to be compatible with our framework. In addition, our protocol is much more efficient concretely and asymptotically, e.g., in contrast to the linear proof size and verifier time in earlier work [DBB$^+$15, DV19, DBV21, DV21]. In addition, our "anonymity set" is the entire cryptocurrency platform's existing accounts.

For a platform to comply with π_{ASSET2}, we require that each platform user should have a unique and secret key s_i that generates its organization ID $a_i = [s_i]_1 + r[h]_1$, which is a Pedersen commitment to s_i and is made public. Each account (e.g., a wallet address in BTC, and a coin in ZeroCash) is generated by a one-way function $f(s_i, \mathsf{aux})$ where aux are other pertinent information, and s_i is the private key. For instance, s_i can be simply added as a leaf node of the Merkle tree that generates a ZeroCash coin.

We require that a unique random nonce k be available for each transaction cycle. When a user submits a transaction request to transfer balance from an account of organization a_i to an account of another organization a_j, let R be the original transaction request (usually including a zero knowledge proof for its validity), we require the following record be submitted.

$$R, \mathsf{hash}(s_i, k), \mathbf{C}_{\Delta_i}, \pi_i, \mathsf{hash}(s_j, k), \mathbf{C}_{\Delta_j}, \pi_j$$

Here we simplify the model and assume that a_i is the sender and a_j is the receiver. $\mathsf{hash}()$ is a collision resistant, and pre-image resistant hash function, and $\mathsf{hash}(s_i, k)$ is uniquely determined by s_i and k. π_i asserts that the change of balance for the account of a_i is consistent with the information hidden in the transaction request R. Similarly is π_j defined. We require that $\mathbf{C}_{\Delta_i}$ is a Pedersen commitment (thus *homomorphic*) to balance change of i so that balance change for the same organization can be accumulated for a given transaction cycle. We require that $\mathbf{C}_{\Delta_i}$ is defined over a nested curve for a zkSNARK system so that its $\mathbb{F}_p$ is the $\mathbb{F}_r$ of the zkSNARK system (e.g., the customized curve 25519 for BN-254 in [KCM$^+$15]). We let G and H be two Pedersen keys defined over the nested curve, thus $\mathbf{C}_{\Delta_i} = G^{\Delta_i} H^r$ for some random factor r.

[5]It requires $O(n\mathsf{log}(n))$ lookup argument preprocessing at the server side of cryptocurrency. This can be handled by a trusted smart contract service that employs parallel computing.

Once all transaction requests are collected. The blockchain publishes the following three vectors by merging the records for the same organization. This is possible because they are hashed using the same random nonce k.

$$(\mathbf{h}, \mathbf{C_\Delta.x}, \mathbf{C_\Delta.y})$$

Here $\mathbf{h}_i$ is $\mathsf{hash}(s_i, k)$ for some account a_i and $\mathbf{C}_{\mathbf{\Delta}i}$ is the corresponding accumulated balance changes (a Pedersen commitment). Note that the platform has no knowledge behind the commitments, and only when an organization is the owner of a_i, she knows the opening of the corresponding $\mathbf{C}_{\Delta i}$. As each element in $\mathbf{C_\Delta}$ is a curve point. We use $\mathbf{C_\Delta.x}$ and $\mathbf{C_\Delta.y}$ to indicate the vectors of x and y coordinates. Let $\mathbf{C_h}$, $\mathbf{C}_{\mathbf{C_\Delta.x}}$, and $\mathbf{C}_{\mathbf{C_\Delta.y}}$ be the Pedersen vector commitment to the three vectors with blinding factor set to 0, i.e., they are just used as succinct representation.

We note that the scheme leaks some structural information of a transaction cycle, e.g., how many distinct organizations and the multiplicity of their transactions. Notice that these organizations are anonymous and untraceable (linkable) between transaction cycles. We regard this small leak does not hurt the overall security. Under this restriction, the scheme has $O(1)$ prover/verifier complexity. A complete zk variation is available at increasing the prover complexity to $O(n\log(n))$ where n is the size of transaction set in a cycle.

Our scheme essentially relies on position-hiding vector commitment such as Caulk [ZBK+22]. However, for the convenience in implementation, we re-use the lookup argument in $\pi_{\texttt{ASSET1}}$, as a size-1-query zk-lookup argument is essentially a vector commitment open proof, with the same asymptotic complexity. Replacing it with a vector commitment scheme results in gain in concrete performance, and remains as a future direction of the work.

Once the transaction set is determined, the blockchain preprocesses $\mathbf{h}$, $\mathbf{C_\Delta.x}$, and $\mathbf{C_\Delta.y}$. Blockchain publishes the corresponding aux information. The prover (the organization) constructs $\pi_{\texttt{ASSET2}}(a_i, \mathbf{C}_v, \mathbf{C_h}, \mathbf{C}_{\mathbf{C_\Delta.x}}, \mathbf{C}_{\mathbf{C_\Delta.y}})$, which asserts that given the three succinct representation, for a_i: $\mathbf{C}_v$ hides its total balance change.

1. $\mathbf{P}$ computes $\mathbf{C}_{h_i} = [\mathsf{hash}(s_i, r)]_1 + r_i[h]_1$, and $\mathbf{C}_x = [\mathbf{C}_{\Delta_i}.x] + r_2[h]_1$, $\mathbf{C}_y = [\mathbf{C}_{\Delta_i}.y] + r_3[h]_1$, Then $\mathbf{P}$ and $\mathbf{V}$ run a folded 1-query lookup argument which asserts that the above three commitments hide an element (with the same index) in $\mathbf{h}$, $\mathbf{C_\Delta.x}$ and $\mathbf{C_\Delta.y}$.

2. $\mathbf{P}$ use a commit-and-prove zk-SNARK to prove the relation between $\mathbf{C}_{h_i}$, $\mathbf{C}_x$, $\mathbf{C}_y$ with the statement. Formally, it is defined as below:

$$\begin{aligned}
&\pi_{\texttt{SNARK}}(\mathbf{C}_{h_i}, \mathbf{C}_x, \mathbf{C}_y, a_i, \mathbf{C}_v, k) \\
&\quad (s_i, r_i, r_1, r_2, r_3, x, y) : \\
&\qquad h_i = \mathsf{hash}(s_i, k) \;\wedge \\
&\qquad (x, y) = G^v H^r \\
&\}
\end{aligned}$$

The system mainly reasons about how h_i is generated by hash and how (x, y) is generated by a Pedersen commitment over G and H (embedded curve). Note that the random nonces r_1, r_2, r_3 are served as witness wires in the above proof, and h_i, x, y are still private witness values. Their commitment $\mathbf{C}_{h_i}, \mathbf{C}_x, \mathbf{C}_y$ are extracted using QA-NIZK (the technique used by CP-link in LegoSnark [CFQ19]) from the zkSNARK proof (e.g., [Gro16]).

Efficiency: The platform cost is $O(n\mathsf{log}(n))$ for preprocessing, and the prover complexity and verifier complexity are both $O(1)$. Concretely, the SNARK proof needs to encode one hash operation and one curve point exponentiation. When hash like Poseidon [GKK+21] is used, the total number of R1CS constraints will be less than $1k$. The cost of the 1-query lookup argument is negligible. We estimate that one computing thread can generate the proof in 0.1 second. The only restriction is the preprocessing for the three tables at the server side. Parallel/distributed computing can be leveraged. For instance, for $24k$ TPS (e.g., that of NYSE), the preprocessing step at the platform will need to run 2^{20} $\mathbb{G}_1$ exponentiations. We estimate that with 256 CPUs, the computing time can be controlled within 1 second.

Acknowledgment: We would like to thank Dr. Ariel Gabizon for the discussion over lookup arguments.

References

[AGM18] S. Agrawal, C. Ganesh, and P. Mohassel. Non-interactive zero-knowledge proofs for composite statements. In *CRYPTO*, pages 643–673, 2018.

[CFQ19] M. Campanelli, D. Fiore, and A. Querol. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In *CCS*, pages 2075–2092, 2019.

[CGG+23] A. Choudhuri, S. Garg, A. Goel, S. Sekar, and R. Sinha. Sublonk: Sublinear prover plonk. `https://eprint.iacr.org/2023/902`, 2023.

[CHM+20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. P. Ward. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *EUROCRYPT*, pages 738–768, 2020.

[CLMN20] K. Chalkias, K. Lewi, P. Mohassel, and V. Nikolaenko. Distributed auditing proofs of liabilities. IACR Cryptol. ePrint Arch. `https://eprint.iacr.org/2020/468`, 2020.

[DBB+15] G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh. Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges. In *CCS*, pages 720–731, 2015.

[DBV21] A. Dutta, S. Bagad, and S. Vijayakumaran. MProve+: privacy enhancing proof of reserves protocol for monero. *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*, 16:3900–3915, 2021.

[DV19] A. Dutta and S. Vijayakumaran. MProve: a proof of reserves protocol for monero exchanges. In *Euros&p Workshops*, pages 330–339, 2019.

[DV21] A. Dutta and S. Vijayakumaran. Revelio: a MimbleWimble proof of reserves protocol. In *CVCBT*, pages 7–11, 2021.

[EFG22] L. Eagen, D. Fiore, and A. Gabizon. cq: Cached quotients for fast lookups. IACR Cryptol. ePrint Arch., 2022.

[GK22] A. Gabizon and D. Khovratovich. Flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size. IACR Cryptol. ePrint Arch., 2022.

[GKK$^+$21] L. Grassi, D. Kales, D. Khovratovich, A. Roy, C. Rechberger, and M. Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *USENIX Security*, 2021.

[Gro16] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[GW20] A. Gabizon and Z. J. Williamson. Plookup: A simplified polynomial protocol for lookup tables. IACR Cryptol. ePrint Arch., 2020.

[Hab22] U. Habock. Multivariate lookups based on logarithmic derivatives. IACR Cryptol. ePrint Arch., 2022.

[JC21] Y. Ji and K. Chalkias. Generalized proof of liabilities. In *CCS*, pages 3465–3486, 2021.

[KCM$^+$15] A. Kosba, Z. Chao, A. Miller, Y. Qian, T. H. Chan, C. Papamanthou, R. Pass, and a. shelat. *cøø*: A framework for building compososable zero-knowledge proofs. *Cryptology ePrint Archive. 2015/1093*, 2015.

[KZG10] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, pages 177–194, 2010.

[MGGR13] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *SSP*, pages 397–411, 2013.

[PK22] J. Posen and A. Kattis. CaulkPlus: Table-independent lookup arguments. IACR Cryptol. ePrint Arch., 2022.

[SKBP22] S. Srinivasan, I. Karantaidou, F. Baldimtsi, and C. Papamanthou. Batching, Aggregation, and Zero-Knowledge Proofs in Bilinear Accumulators. In *CCS*, pages 2719–2733, 2022.

[YSL$^+$] T. H. Yuen, S.-F. Sun, J. K. Liu, M. H. Au, M. F. Esgin, Q. Zhang, and D. Gu. Ringct 3.0 for blockchain confidential transaction: Shorter size and stronger security. *available at* `https://eprint.iacr.org/2019/508.pdf`.

[ZBK$^+$22] A. Zapico, V. Buterin, D. Khovratovich, M. Maller, A. Nitulescu, and M. Simkin. Caulk: Lookup Arguments in Sublinear Time. In *CCS*, pages 3121–3134, 2022.

[ZGK$^+$22] A. Zapico, A. Gabizon, D. Khovratovich, M. Maller, and C. Ràfols. Baloo: Nearly Optimal Lookup Arguments. IACR Cryptol. ePrint Arch., 2022.