Xiezhi: Toward Succinct Proofs of Solvency

Youwei Deng, Majid Khabbazian, and Jeremy Clark

[1] University of Alberta, Edmonton, Canada
{youwei3,mkhabbaz}@ualberta.ca
[2] Concordia University, Montreal, Canada
j.clark@concordia.ca

Abstract. A proof of solvency (or proof of reserves) is a zero-knowledge proof produced by a centralized cryptocurrency exchange to demonstrate that it holds enough cryptocurrency to cover all customer balances, while revealing nothing beyond solvency itself. A key challenge in building practical end-to-end proofs of solvency is the cryptographic mismatch between asset keys and modern succinct non-interactive arguments of knowledge (SNARKs): many deployed account-based cryptocurrency systems use `secp256k1`, whereas efficient SNARKs are built over pairing-friendly curves such as `bls12-381`. Bridging these two settings is therefore necessary for any private end-to-end construction. We introduce Xiezhi, an end-to-end proof of solvency protocol that avoids directly translating keys across curves (which is expensive) and instead transfers asset-ownership claims into the SNARK-friendly setting with much lower proving overhead. A Rust prototype over `bls12-381` shows that the reusable key-ownership phase remains practical for real-world exchanges, while the recurring asset proof has constant-size proofs and constant-time verification. For liabilities, the column-wise range proof scales linearly in proving time but keeps proof size and auditor verification independent of the number of users for fixed precision; with 32-bit balances, proofs are below 10 KB and verify in under 8 ms. These results indicate that Xiezhi offers practical end-to-end private proofs of solvency with substantially lower proving overhead than prior circuit-based end-to-end approaches.

Keywords: Proof of solvency · zero knowledge · polynomial commitments · KZG · SNARKs

1 Introduction

When the Bitcoin exchange Mt. Gox was declared bankrupt in 2014, a curious fact was reported in the *New York Times*—the missing 744K BTC "had gone unnoticed for years." Regular proofs of solvency would not eliminate every failure mode of a centralized exchange, but they would make such losses much harder to conceal and much more likely to be detected early. This motivates continued work on practical, privacy-preserving proofs that an exchange is solvent; that is, that it holds enough cryptocurrency to settle each of its users' balances.

A proof of solvency is a zero-knowledge proof conducted by a centralized cryptocurrency exchange (or, more generally, any custodian of cryptocurrencies) to offer evidence that the exchange owns enough cryptocurrency to settle each of its users' balances. The zero-knowledge component protects the exchange's proprietary information, such as the number of users, individual balances, total liabilities, which cryptocurrency addresses belong to the exchange, and the total amount of cryptocurrency owned by the exchange.

Academics showed that proofs of solvency can be carried out in strict zero-knowledge [1], and this initiated a line of work that continues to improve efficiency [2, 3, 4, 5, 6, 7] and examine the correctness of deployed proofs [8]. Despite this progress, there is still significant room to make such proofs more practical, especially by reducing proving time while retaining small proofs and fast verification.

As shown in Figure 1, an end-to-end proof of solvency naturally decomposes into several sub-components. First, (π_{keys}) proves that the exchange knows the private signing keys corresponding to a hidden subset of a public anonymity set of cryptocurrency addresses. Second, (π_{assets}) proves that the balances associated with this subset are accumulated correctly into the total assets, without revealing the subset or the total. Third, (π_{user}) is an individualized proof given to each user, allowing the user to verify that their balance is correctly included in the overall proof. Fourth, ($\pi_{\mathsf{liabilities}}$) proves that these individual user balances are accumulated correctly into the total liabilities. Finally, (π_{solvency}) proves that the exchange's total assets exceed or equal its total liabilities.

Xiezhi[3] encompasses all of these components in one end-to-end protocol. Its main technical novelty lies in (π_{keys}), which addresses the central obstacle in making end-to-end proofs of solvency practical. The

[3] A folklore creature revered in ancient Chinese culture for its ability to distinguish truth from deceit.

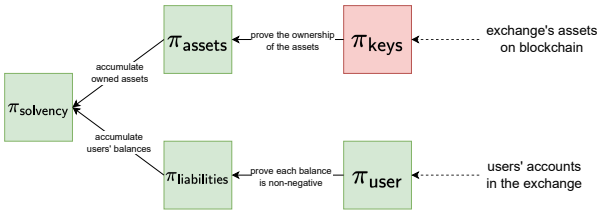


Fig. 1: A high-level overview of Xiezhi and its five sub-components which are coloured green if verifier time and proof size are constant in the number of assets/accounts and red if it is linear. Note π_{keys} is not succinct but the proof of ownership can be reused in different cycles of proving.

difficulty is that the anonymity set consists of public keys on `secp256k1`, whereas the hidden selector that drives the succinct asset proof is committed in `bls12-381`. Xiezhi bridges these two settings using composed Σ-protocols rather than by translating keys across curves. For each address, it proves that the corresponding selector bit either opens to 0, or opens to 1 together with knowledge of the matching `secp256k1` private key. The crucial point is that conjunction and disjunction of Σ-protocols are coordinated through a shared t-bit challenge, which both groups can encode. In this way, Xiezhi transfers the *fact of key ownership*, rather than the key itself, into the SNARK-friendly setting, allowing the remainder of the asset-side proof to proceed succinctly and with much lower proving cost.

Beyond this cross-curve bridge, Xiezhi also makes a practical contribution on the liability side by adapting the protocol to the concrete arithmetic limits of `bls12-381`. A natural approach to proving that all user balances are non-negative is to bit-decompose every balance and encode the resulting bits into a single polynomial over a root-of-unity domain. However, the scalar field of `bls12-381` supports an efficient domain of size at most 2^{32}, and this bound is exceeded quickly when the bits of many user balances are aggregated directly. For even moderately sized exchanges, a naive encoding leaves too few bits per account to represent realistic balances. Xiezhi addresses this by reorganizing the range proof column-wise: it groups the same bit position across all accounts into one polynomial, proves the range constraints across these columns, and then reconstructs balances and total liabilities through polynomial relations. As a result, the proof remains succinct in the number of users, with proof size and verifier work depending only on the chosen bit precision.

Our contributions can be summarized as follows:

- Xiezhi, a new end-to-end proof of solvency protocol[4] that covers all of the sub-components in Figure 1. The protocol is *mostly succinct*: except for π_{keys}, which can be reused across proof cycles until the exchange changes its asset keys, the proof size and verifier time of the remaining components are independent of the number of assets or accounts.
- A novel cross-curve bridge for π_{keys} that connects ownership of common blockchain keys, such as those used by Bitcoin and Ethereum on `secp256k1`, to the pairing-friendly setting of `bls12-381` used by succinct arguments. Rather than translating keys across curves, Xiezhi uses composed Σ-protocols to transfer the *fact of key ownership* into a hidden selector polynomial that can be used by the succinct asset proof.
- Practical adaptations for the concrete arithmetic limits of `bls12-381`, in particular its maximum root-of-unity domain. These adaptations make liability-side range proofs feasible for realistically sized exchanges by organizing the proof column-wise, so that proof size and verifier work depend on the chosen bit precision rather than the number of users.
- A proof-of-concept implementation of Xiezhi together with performance evaluation, showing that the protocol can achieve minute-scale proving for the dominant asset-side step while retaining proof sizes in the KB range and verifier times in seconds.

2 Preliminaries

2.1 Notation

Our protocols work on a finite field of prime order and are between a prover $\mathcal{P}$ and verifier $\mathcal{V}$. Let $g_{\texttt{secp}}$ and $h_{\texttt{secp}}$ denote two independent generators in the `secp256k1` group $\mathbb{G}_{\texttt{secp}}$; and we use $g_{\texttt{bls}}$ and $h_{\texttt{bls}}$

[4] We use the term "proof" loosely to refer to both proofs and arguments; Xiezhi contains both types of components.

for `bls12-381`. We denote by $e([x]_1, [y]_2)$ a non-degenerate bilinear pairing, with input groups specified $[x]_i := g_i^x \in \mathbb{G}_i$ for $i = 1, 2$ (when omitted, values are assumed to be in the first group). For vectors, we use an overline to denote a vector and brackets to denote the elements in this vector, e.g., $\overline{v} = \langle v_1, v_2, \ldots \rangle$. Let $\mathbf{C}(x)$ denote a Pedersen commitment to a value x with a hiding factor. Particularly, we use $\mathbf{C}_{\texttt{secp}}$ and $\mathbf{C}_{\texttt{bls}}$ to denote a commitment in `secp256k1` and `bls12-381`, respectively. To help distinguish polynomials and constants, we denote by $\mathcal{C}_f$ a polynomial commitment to f in `bls12-381`. We use ω to denote roots of unity of a field H.

2.2 Terminology

A balance sheet consists of liabilities (value owed to others) and assets (value owned). When the total asset value at least as large as total liabilities, the firm is called solvent. The amount by which the assets exceed the liabilities is called capital or equity (depending on context). Some literature prefers the term 'proof of reserves' to 'proof of solvency.'

2.3 zk-SNARK Customization

Modern SNARK toolkits let protocol designers choose how "close to the metal" they want to work. At the highest level, they can write ordinary software and prove its execution inside a zkVM. A step lower, they can write a dedicated circuit, or use a circuit DSL, thereby skipping the zkVM compilation layer for better performance. Going deeper still, they can design the arithmetization directly. We call this *application-specific arithmetization*: specifying a set of polynomial relations, typically expressed as vanishing constraints, that capture the protocol semantics. This arithmetization is then instantiated as a succinct argument via a polynomial interactive oracle proof.

Xiezhi operates at this lower layer. For the recurring SNARK-friendly components, we directly engineer the polynomial relations used by the protocol rather than compiling a higher-level program or circuit. For the key-ownership component, we avoid an even more expensive route: expressing `secp256k1` secret-key knowledge inside a generic SNARK circuit. This design choice is motivated by the cost of circuit-based end-to-end approaches. IZPR, the closest circuit-based proof-of-reserves system, reports roughly 5×10^5 constraints per key and day-scale proving time for an anonymity set of 6000 keys [7]. In contrast, Xiezhi proves key ownership using composed Σ-protocols and binds the result to a hidden selector polynomial over `bls12-381`. For the same 6000-key regime, this reusable key-ownership phase completes on the order of seconds in our prototype, after which the recurring asset-accumulation argument has constant proof size and constant argument-verification time.

2.4 Cryptographic Building Blocks

We assume familiarity with the following standard cryptographic primitives: the discrete logarithm assumption, Pedersen commitments, zero-knowledge proofs and their related properties, Σ-protocols, including conjunction and disjunction, roots of unity, zero-knowledge range proof, KZG commitment scheme, and polynomial interactive oracle proof (PIOP).

Particularly, Xiezhi uses two standard KZG evaluation interfaces. The first, denoted KZG_{zk}, is the zero-knowledge opening interface used by KZG-based PIOPs/SNARKs when polynomial relations are checked at verifier-chosen points. It relies on committing to blinded polynomials, so that openings outside the evaluation domain do not reveal unintended information about the witness polynomials. The second, denoted KZG_{cm}, is an open-to-a-commitment interface: instead of revealing an evaluation $f(a)$ in the clear, the prover binds it to a Pedersen commitment $\mathbf{C}_{\texttt{bls}}(f(a))$ and proves that this committed value is the evaluation of the polynomial committed in $\mathcal{C}_f$ at the point a. We use this interface for hidden aggregate values such as total assets, total liabilities, and equity. For compactness, the protocol descriptions in the main text use the notation $\mathbf{C}_{\texttt{bls}}(f(a)) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f; f(a); a)$ and $\{1/0\} \leftarrow \mathsf{KZG}_{cm}.\mathsf{Verify}(\mathcal{C}_f; \mathbf{C}_{\texttt{bls}}(f(a)); a)$. for the proving and verifying algorithms of KZG_{cm}.

3 Proof of Assets (PoA)

We now describe the asset side of Xiezhi, using ETH as the running example. We focus on account-based assets, where each account has a public balance at a fixed audit block height. For clarity, the main

construction is stated for an anonymity set of public keys.[5] The PoA consists of two phases: π_{keys} and π_{assets}.

In π_{keys}, the exchange publishes an ordered anonymity set of public keys, containing the keys it owns hidden among other keys from the blockchain. [6] The exchange then commits, in `bls12-381`, to a hidden binary selector vector. The selector records 0 at positions whose `secp256k1` secret key the exchange does not claim to know, and 1 at positions whose secret key it proves knowledge of. In π_{assets}, the public balances corresponding to the same ordered key set are encoded as a balance polynomial. The exchange proves that the balances selected by the hidden selector are accumulated correctly into a committed total asset value, without revealing which keys are selected or the total amount of assets.

3.1 The π_{keys} Proof

The purpose of π_{keys} is to certify a hidden selector polynomial that can be used by the succinct asset-accumulation argument. Let

$$\mathcal{A}' = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_\kappa\}$$

be the public anonymity set of blockchain public keys, ordered by index, and let

$$H_{\mathsf{keys}} = \{\omega^0, \omega^1, \ldots, \omega^{\kappa-1}\}$$

be the corresponding evaluation domain. The exchange commits to a binary selector vector

$$\overline{s} = (s_1, \ldots, s_\kappa) \in \{0,1\}^\kappa,$$

where $s_i = 1$ means that the exchange claims ownership of pk_i, and $s_i = 0$ means that it does not claim this key. The selector vector is interpolated as a polynomial $f_{\mathsf{sel}}(X)$ over the `bls12-381` scalar field, with

$$f_{\mathsf{sel}}(\omega^{i-1}) = s_i \qquad \text{for all } i \in [\kappa],$$

and the exchange publishes a KZG commitment $\mathcal{C}_{f_{\mathsf{sel}}}$. This committed selector is the only information about key ownership that the subsequent π_{assets} argument needs.

The challenge is that the ownership statement itself lives in a different group. For Ethereum-style keys, ownership means knowledge of sk_i such that

$$\mathsf{pk}_i = g_{\mathsf{secp}}^{\mathsf{sk}_i}$$

in the `secp256k1` group, whereas the selector commitment $\mathcal{C}_{f_{\mathsf{sel}}}$ and the later polynomial argument live over `bls12-381`. A direct approach would try to translate the secret key, or a commitment to it, across the two groups. This is unnecessary: π_{assets} never uses a `bls12-381` representation of the secret key. It only needs a hidden bit s_i that is guaranteed to be 1 only when the exchange knows the corresponding `secp256k1` secret key. Thus π_{keys} does not transfer secret-key values across curves; it transfers the truth of an ownership predicate into a hidden selector polynomial.

More concretely, for every $i \in [\kappa]$, π_{keys} certifies the extractable invariant

$$s_i = f_{\mathsf{sel}}(\omega^{i-1}) \in \{0,1\},$$

and, for every selected index,

$$s_i = 1 \Rightarrow \mathsf{PoK}_{\mathsf{secp}}\left\{\mathsf{sk}_i : \mathsf{pk}_i = g_{\mathsf{secp}}^{\mathsf{sk}_i}\right\}.$$

Equivalently, from an accepting prover, an extractor can recover sk_i for every index i with $s_i = 1$. For indices with $s_i = 0$, the protocol makes no statement about whether a discrete logarithm for pk_i exists mathematically, or whether some other party knows it; it only states that the exchange does not claim that key in this proof.

Cross-curve ownership-to-selector bridge. For each index i, π_{keys} first opens the committed selector polynomial at ω^{i-1} to a Pedersen commitment rather than to the plaintext bit:

$$\mathbf{C}_{\mathtt{bls}}(s_i) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f_{\mathsf{sel}}; s_i; \omega^{i-1}).$$

[5] In deployments where the public anonymity set consists of hashed or address-derived identifiers, the extension in Appendix 9 proves consistency with the claimed public key.

[6] The anonymity set should remain stable across proof cycles. The exchange may add keys to the set, but removing keys requires rerunning π_{keys} for the updated set.

For a fixed index i, $\mathcal{P}$ and $\mathcal{V}$ are both given pk_i and $\mathbf{C}_{\mathtt{bls}}(s_i)$. The prover knows $s_i \in \{0,1\}$ and an opening r_i such that $\mathbf{C}_{\mathtt{bls}}(s_i) = g_{\mathtt{bls}}^{s_i} h_{\mathtt{bls}}^{r_i}$. If $s_i = 1$, the prover also knows sk_i such that $\mathsf{pk}_i = g_{\mathtt{secp}}^{\mathsf{sk}_i}$.

1. Case 1: $s_i = 1$ ($\mathcal{P}$ claims ownership of pk_i).
   (a) $\mathcal{P}$ selects $e_1 \xleftarrow{\$} \{0,1\}^t$, $z_3, \beta \xleftarrow{\$} \mathbb{Z}_{\mathtt{bls}}$, and $\alpha \xleftarrow{\$} \mathbb{Z}_{\mathtt{secp}}$.
   (b) $\mathcal{P}$ publishes $t_1 = g_{\mathtt{secp}}^{\alpha}$.
   (c) $\mathcal{P}$ publishes $t_2 = h_{\mathtt{bls}}^{\beta}$.
   (d) $\mathcal{P}$ publishes $t_3 = g_{\mathtt{bls}}^{-e_1} h_{\mathtt{bls}}^{z_3 - r_i e_1}$.
   (e) $\mathcal{V}$ publishes a t-bit challenge $e \xleftarrow{\$} \{0,1\}^t$, or $\mathcal{P}$ derives it via Fiat–Shamir.
   (f) $\mathcal{P}$ computes $e_0 = e \oplus e_1$ and publishes e_0, e_1.
   (g) $\mathcal{P}$ publishes $z_1 = e_0 \mathsf{sk}_i + \alpha$.
   (h) $\mathcal{P}$ publishes $z_2 = e_0 r_i + \beta$.
   (i) $\mathcal{P}$ publishes z_3.
2. Case 2: $s_i = 0$ ($\mathcal{P}$ does not claim pk_i).
   (a) $\mathcal{P}$ selects $e_0 \xleftarrow{\$} \{0,1\}^t$, $z_1 \xleftarrow{\$} \mathbb{Z}_{\mathtt{secp}}$, and $z_2, \alpha \xleftarrow{\$} \mathbb{Z}_{\mathtt{bls}}$.
   (b) $\mathcal{P}$ publishes $t_1 = g_{\mathtt{secp}}^{z_1} / \mathsf{pk}_i^{e_0}$.
   (c) $\mathcal{P}$ publishes $t_2 = g_{\mathtt{bls}}^{e_0} h_{\mathtt{bls}}^{z_2 - r_i e_0}$.
   (d) $\mathcal{P}$ publishes $t_3 = h_{\mathtt{bls}}^{\alpha}$.
   (e) $\mathcal{V}$ publishes a t-bit challenge $e \xleftarrow{\$} \{0,1\}^t$, or $\mathcal{P}$ derives it via Fiat–Shamir.
   (f) $\mathcal{P}$ computes $e_1 = e \oplus e_0$ and publishes e_0, e_1.
   (g) $\mathcal{P}$ publishes z_1.
   (h) $\mathcal{P}$ publishes z_2.
   (i) $\mathcal{P}$ publishes $z_3 = e_1 r_i + \alpha$.
3. $\mathcal{V}$ outputs **acc** if and only if
   (a) $e = e_0 \oplus e_1$;
   (b) $g_{\mathtt{secp}}^{z_1} = \mathsf{pk}_i^{e_0} t_1$;
   (c) $g_{\mathtt{bls}}^{e_0} h_{\mathtt{bls}}^{z_2} = \mathbf{C}_{\mathtt{bls}}(s_i)^{e_0} t_2$;
   (d) $h_{\mathtt{bls}}^{z_3} = \mathbf{C}_{\mathtt{bls}}(s_i)^{e_1} t_3$.

Protocol 1: Per-index cross-curve ownership-to-selector proof. The proof shows that the committed selector either opens to 0, or opens to 1 together with knowledge of the corresponding `secp256k1` secret key.

The verifier checks that this committed value is indeed the evaluation of the committed selector polynomial:

$$\mathsf{KZG}_{cm}.\mathsf{Verify}(\mathcal{C}_{f_{\mathsf{sel}}}; \mathbf{C}_{\mathtt{bls}}(s_i); \omega^{i-1}) = 1.$$

The remaining task is to prove that $\mathbf{C}_{\mathtt{bls}}(s_i)$ is a commitment to a valid selector bit and that the bit is tied to key ownership. Writing $\mathbf{C}_{\mathtt{bls}}(s_i) = g_{\mathtt{bls}}^{s_i} h_{\mathtt{bls}}^{r_i}$, Xiezhi proves the following relation:

$$\begin{aligned} \mathrm{ZKPoK}\big\{(\mathsf{sk}_i, r_i) : [\mathsf{pk}_i = g_{\mathtt{secp}}^{\mathsf{sk}_i} \wedge \mathbf{C}_{\mathtt{bls}}(s_i) = g_{\mathtt{bls}} h_{\mathtt{bls}}^{r_i}] \\ \vee [\mathbf{C}_{\mathtt{bls}}(s_i) = h_{\mathtt{bls}}^{r_i}]\big\}. \end{aligned} \tag{1}$$

Equivalently, the prover shows that either $s_i = 0$, or $s_i = 1$ and it knows the `secp256k1` secret key for pk_i. The selector value remains hidden because $\mathbf{C}_{\mathtt{bls}}(s_i)$ is a Pedersen commitment and the proof is zero knowledge.

Although the relation in (1) spans two groups, it can be proved using standard Σ-protocol composition. The left branch combines a Schnorr proof of knowledge in `secp256k1` with a Pedersen-opening proof in `bls12-381`; the right branch is a Pedersen-opening proof in `bls12-381`. The disjunction is coordinated by a shared t-bit challenge $e = e_0 \oplus e_1$. Since both scalar fields can encode the same t-bit challenge, the usual conjunction and disjunction composition of Σ-protocols applies even though the algebraic checks are performed in different groups. Protocol 1 gives the concrete per-index proof.

In the full π_{keys} protocol, the combination of KZG_{cm} and Protocol 1 performs three roles. First, it binds the hidden Pedersen commitment $\mathbf{C}_{\mathtt{bls}}(s_i)$ to the i-th evaluation of the committed selector polynomial f_{sel}. Second, it enforces that the committed selector value is Boolean: under the binding property of Pedersen commitments, the right branch permits only a commitment to 0, while the left branch permits only a commitment to 1. Third, it ties the $s_i = 1$ case to knowledge of the corresponding `secp256k1` secret key. Thus an accepting proof certifies that every 1 in the hidden selector polynomial corresponds

1. $\mathcal{P}$ publishes an ordered anonymity set of public keys: $\langle \mathsf{pk}_1, \mathsf{pk}_2, \mathsf{pk}_3, \ldots, \mathsf{pk}_\kappa \rangle$.
2. $\mathcal{P}$ constructs a selector vector: $\overline{s} = \langle s_1, s_2, s_3, \ldots, s_\kappa \rangle$, where $s_i = 1$ for each key it claims and $s_i = 0$ otherwise.
3. $\mathcal{P}$ interpolates a polynomial $f_{\mathsf{sel}}(X)$ such that $f_{\mathsf{sel}}(\omega^{i-1}) = s_i$ for every $i \in [\kappa]$, and publishes the KZG commitment $\mathcal{C}_{f_{\mathsf{sel}}}$.
4. For each $i \in [\kappa]$, $\mathcal{P}$ computes $\mathbf{C}_{\mathtt{bls}}(s_i) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f_{\mathsf{sel}}; s_i; \omega^{i-1})$ and sends $\mathbf{C}_{\mathtt{bls}}(s_i)$ to $\mathcal{V}$.
5. For each $i \in [\kappa]$:
   (a) $\mathcal{P}$ and $\mathcal{V}$ run Protocol 1 on public inputs pk_i and $\mathbf{C}_{\mathtt{bls}}(s_i)$.
   (b) $\mathcal{V}$ aborts if $\mathsf{KZG}_{cm}.\mathsf{Verify}(\mathcal{C}_{f_{\mathsf{sel}}}; \mathbf{C}_{\mathtt{bls}}(s_i); \omega^{i-1})$ returns 0.

Protocol 2: The π_{keys} proof certifies that $f_{\mathsf{sel}}(X)$ encodes a hidden binary selector vector and that every selected public key is backed by knowledge of the corresponding `secp256k1` secret key.

to a public key owned by the exchange, while revealing neither which branch was used nor which keys are owned.

The proof is repeated once for each key in the anonymity set, so π_{keys} is linear in proof size and verifier work. This non-succinct step is deliberately isolated from the recurring asset proof. Once $\mathcal{C}_{f_{\mathsf{sel}}}$ has been certified, the exchange can reuse the same selector commitment across proof-of-solvency cycles until it changes the set of asset keys it claims. The subsequent π_{assets} argument then proceeds entirely in the `bls12-381` setting: it multiplies the hidden selector polynomial by the public balance polynomial and proves that the selected balances are accumulated correctly.

The full π_{keys} protocol is given in Protocol 2. It publishes the anonymity set, commits to the hidden selector polynomial, opens each selector evaluation to a Pedersen commitment through KZG_{cm}, and runs the cross-curve proof of Protocol 1 for each index.

3.2 The π_{assets} Argument

The π_{keys} protocol certifies that $\mathcal{C}_{f_{\mathsf{sel}}}$ is a commitment to a hidden binary selector polynomial $f_{\mathsf{sel}}(X)$ over the `bls12-381` scalar field. The selector is indexed consistently with the public anonymity set: for each $i \in [\kappa]$,

$$f_{\mathsf{sel}}(\omega^{i-1}) = s_i,$$

where $s_i = 1$ only for keys that the exchange claims and for which π_{keys} proves knowledge of the corresponding `secp256k1` secret key.

At a fixed audit block height, the balance of each public key in the anonymity set is public blockchain data. These balances are encoded in a polynomial $f_{\mathsf{bal}}(X)$ over the same domain $H_{\mathsf{assets}} = \{\omega^0, \omega^1, \ldots, \omega^{\kappa-1}\}$, so that

$$f_{\mathsf{bal}}(\omega^{i-1}) = \mathfrak{bal}_i \quad \text{for each } i \in [\kappa].$$

The verifier uses the balance commitment $\mathcal{C}_{f_{\mathsf{bal}}}$ for this audit block and the certified selector commitment $\mathcal{C}_{f_{\mathsf{sel}}}$ from π_{keys}.

The pointwise product $f_{\mathsf{sel}}(\omega^{i-1}) \cdot f_{\mathsf{bal}}(\omega^{i-1}) = s_i \mathfrak{bal}_i$ keeps the balance of every selected key and zeroes out every unclaimed key. Thus the total assets claimed by the exchange are $\sum_{i=1}^{\kappa} s_i \mathfrak{bal}_i$. To prove this sum without revealing the selected keys or the total asset value, the prover constructs an accumulator polynomial $f_{\mathsf{assets}}(X)$. Its evaluations are intended to satisfy

$$f_{\mathsf{assets}}(\omega^{i-1}) = \sum_{j=i}^{\kappa} s_j \mathfrak{bal}_j \quad \text{for each } i \in [\kappa].$$

Equivalently, $f_{\mathsf{assets}}(X)$ satisfies the recurrence

$$f_{\mathsf{assets}}(X) - f_{\mathsf{assets}}(X\omega) = f_{\mathsf{bal}}(X) \cdot f_{\mathsf{sel}}(X) \quad \text{for } X \neq \omega^{\kappa-1},$$

together with the boundary condition

$$f_{\mathsf{assets}}(\omega^{\kappa-1}) = f_{\mathsf{bal}}(\omega^{\kappa-1}) \cdot f_{\mathsf{sel}}(\omega^{\kappa-1}).$$

These two conditions imply that the first accumulator entry is the total selected balance:

$$f_{\mathsf{assets}}(\omega^0) = \sum_{i=1}^{\kappa} s_i \mathfrak{bal}_i.$$

– Inputs
  - $\mathtt{pk} := \{f_{\mathsf{bal}}(X), f_{\mathsf{sel}}(X)\}$, where f_{bal} is interpolated from the public balances $\langle \mathfrak{bal}_1, \mathfrak{bal}_2, \ldots, \mathfrak{bal}_\kappa \rangle$ of the public keys in the anonymity set at the audit block height, and f_{sel} is the selector polynomial certified by π_{keys}.
  - $\mathtt{vk} := \{\mathcal{C}_{f_{\mathsf{bal}}}, \mathcal{C}_{f_{\mathsf{sel}}}\}$, where $\mathcal{C}_{f_{\mathsf{bal}}}$ is the public balance commitment for the audit block and $\mathcal{C}_{f_{\mathsf{sel}}}$ is the certified selector commitment from π_{keys}.
– Protocol
  1. $\mathcal{P}$ takes as input the selector polynomial $f_{\mathsf{sel}}(X)$ certified by π_{keys} and the public balance polynomial $f_{\mathsf{bal}}(X)$ for the audit block.
  2. $\mathcal{P}$ constructs an accumulator polynomial $f_{\mathsf{assets}}(X)$ over $H_{\mathsf{assets}} = \{\omega^0, \omega^1, \ldots, \omega^{\kappa-1}\}$ such that

$$f_{\mathsf{assets}}(X) - f_{\mathsf{assets}}(X\omega) = f_{\mathsf{bal}}(X) \cdot f_{\mathsf{sel}}(X), \quad X \in H_{\mathsf{assets}} \setminus \{\omega^{\kappa-1}\} \tag{2}$$

$$f_{\mathsf{assets}}(\omega^{\kappa-1}) = f_{\mathsf{bal}}(\omega^{\kappa-1}) \cdot f_{\mathsf{sel}}(\omega^{\kappa-1}) \tag{3}$$

  3. $\mathcal{P}$ commits to f_{assets} and sends $\mathcal{C}_{f_{\mathsf{assets}}}$ to $\mathcal{V}$. The verifier uses the fixed commitments $\mathcal{C}_{f_{\mathsf{bal}}}$ and $\mathcal{C}_{f_{\mathsf{sel}}}$ from the inputs.
  4. $\mathcal{P}$ opens the committed evaluation of f_{assets} at ω^0, i.e.,

$$\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0)) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f_{\mathsf{assets}}; f_{\mathsf{assets}}(\omega^0); \omega^0).$$

  5. $\mathcal{P}$ and $\mathcal{V}$ run `PIOP` to prove the following relations hold over $H_{\mathsf{assets}} = \{\omega^0, \omega^1, \ldots, \omega^{\kappa-1}\}$:

$$\mathcal{R}_1 = \{f_{\mathsf{assets}}, f_{\mathsf{bal}}, f_{\mathsf{sel}} \mid [f_{\mathsf{assets}}(X) - f_{\mathsf{assets}}(X\omega) - f_{\mathsf{bal}}(X) \cdot f_{\mathsf{sel}}(X)] \cdot (X - \omega^{\kappa-1}) = 0\},$$

$$\mathcal{R}_2 = \{f_{\mathsf{assets}}, f_{\mathsf{bal}}, f_{\mathsf{sel}} \mid [f_{\mathsf{assets}}(X) - f_{\mathsf{bal}}(X) \cdot f_{\mathsf{sel}}(X)] \cdot \frac{X^\kappa - 1}{X - \omega^{\kappa-1}} = 0\}.$$

  6. $\mathcal{V}$ outputs **acc** if and only if
     (a) The `PIOP` proof for $\mathcal{R}_1$ and $\mathcal{R}_2$ accepts.
     (b) $\mathsf{KZG}_{cm}.\mathsf{Verify}(\mathcal{C}_{f_{\mathsf{assets}}}; \mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0)); \omega^0)$ returns 1.

Protocol 3: The π_{assets} proof demonstrates that the public balances selected by the hidden selector polynomial from π_{keys} are accumulated correctly into $f_{\mathsf{assets}}(\omega^0)$, while unselected balances are zeroed out and the total asset value is revealed only as a Pedersen commitment.

Finally, π_{assets} does not reveal $f_{\mathsf{assets}}(\omega^0)$ in the clear. Instead, the prover opens this evaluation through KZG_{cm} and publishes the Pedersen commitment

$$\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0)) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f_{\mathsf{assets}}; f_{\mathsf{assets}}(\omega^0); \omega^0).$$

This committed total is later used in the solvency argument. Therefore, after the reusable π_{keys} phase has certified the hidden selector, π_{assets} proceeds entirely in the `bls12-381` setting: it proves that the public balances selected by the hidden ownership vector are accumulated correctly, while revealing neither the selected keys nor the total asset value. The full details are provided in Protocol 3.

4 Proof of Liabilities

4.1 The $\pi_{\mathsf{liabilities}}$ Argument

The liability side must prove two facts: every reported user balance is a non-negative integer in an agreed range, and the committed total liabilities are the sum of those balances. The non-negativity requirement is essential because an insolvent exchange could otherwise reduce its total liabilities by adding fake accounts with negative balances. In Xiezhi, each balance is encoded as a k-bit integer in $[0, 2^k)$, where k is chosen large enough to represent the required asset precision while remaining safely below the `bls12-381` scalar-field modulus.

A direct encoding would concatenate the bit decompositions of all μ balances into one vector of length μk and prove the range constraints over a single root-of-unity domain. This is simple, but it is poorly matched to the concrete arithmetic of `bls12-381`: the largest efficient power-of-two domain has size 2^{32}. Thus, a single packed vector can exceed the available domain for large exchanges or high-precision balances. For example, 10^8 accounts with 64-bit balances already require 6.4×10^9 positions, which is larger than 2^{32}.

– Inputs
  - Private input to $\mathcal{P}$: liabilities $\langle \mathfrak{bal}_1, \mathfrak{bal}_2, \ldots, \mathfrak{bal}_\mu \rangle$.
  - Public parameters: the range bound 2^k and the domain $H_{\mathsf{liab}} = \{\omega^0, \omega^1, \ldots, \omega^{\mu-1}\}$.
– Protocol
  1. $\mathcal{P}$ and $\mathcal{V}$ run Protocol 7 on $\langle \mathfrak{bal}_1, \mathfrak{bal}_2, \ldots, \mathfrak{bal}_\mu \rangle$. This produces committed polynomials $p_1, p_2, \ldots, p_k$, where
  $$p_1(\omega^{i-1}) = \mathfrak{bal}_i \quad \text{for all } i \in [\mu].$$
  $\mathcal{V}$ aborts if the range proof rejects.
  2. $\mathcal{P}$ builds an additive accumulator $\overline{\nu} = \langle \nu_1, \nu_2, \ldots, \nu_\mu \rangle$ for p_1: $\nu_\mu = p_1(\omega^{\mu-1}) = \mathfrak{bal}_\mu$, and, for $i \in [1, \mu)$, $\nu_i = \nu_{i+1} + p_1(\omega^{i-1})$. Thus $\nu_1 = \sum_{i=1}^{\mu} \mathfrak{bal}_i$.
  3. $\mathcal{P}$ interpolates f_{liab} from $\overline{\nu}$ and publishes the commitment $\mathcal{C}_{f_{\mathsf{liab}}}$. The commitments to $p_1, p_2, \ldots, p_k$ are those produced in Protocol 7.
  4. $\mathcal{P}$ opens the committed evaluation of f_{liab} at ω^0, i.e., $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0)) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f_{\mathsf{liab}}; f_{\mathsf{liab}}(\omega^0); \omega^0)$.
  5. $\mathcal{P}$ and $\mathcal{V}$ run PIOP to prove the following relations over H_{liab}:
  $$\mathcal{R}_1 = \left\{ f_{\mathsf{liab}}, p_1 \;\middle|\; [f_{\mathsf{liab}}(X) - f_{\mathsf{liab}}(X\omega) - p_1(X)] \cdot (X - \omega^{\mu-1}) = 0 \right\},$$
  $$\mathcal{R}_2 = \left\{ f_{\mathsf{liab}}, p_1 \;\middle|\; [f_{\mathsf{liab}}(X) - p_1(X)] \cdot \frac{X^\mu - 1}{X - \omega^{\mu-1}} = 0 \right\}.$$
  6. $\mathcal{V}$ outputs **acc** if and only if
    (a) The PIOP proof for $\mathcal{R}_1$ and $\mathcal{R}_2$ accepts.
    (b) $\mathsf{KZG}_{cm}.\mathsf{Verify}(\mathcal{C}_{f_{\mathsf{liab}}}; \mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0)); \omega^0)$ returns 1.

Protocol 4: The $\pi_{\mathsf{liabilities}}$ proof demonstrates that every liability is in $[0, 2^k)$ and that all liabilities are accumulated correctly in $f_{\mathsf{liab}}(\omega^0)$, which is revealed only as a Pedersen commitment.

Xiezhi avoids this bottleneck by arranging the bit decomposition column-wise. Instead of placing all bits in one polynomial, the prover creates k polynomials

$$p_1(X), p_2(X), \ldots, p_k(X),$$

where p_j contains the j-th accumulator column for all users. For each user i, the evaluations

$$p_1(\omega^{i-1}), p_2(\omega^{i-1}), \ldots, p_k(\omega^{i-1})$$

encode the range proof for $\mathfrak{bal}_i$. The constraints enforce that $p_k(\omega^{i-1}) \in \{0, 1\}$ and, for $j \in [1, k)$,

$$p_j(\omega^{i-1}) - 2p_{j+1}(\omega^{i-1}) \in \{0, 1\}.$$

Consequently, $p_1(\omega^{i-1}) = \mathfrak{bal}_i$, so the first column contains the actual user balances.

The prover then constructs an accumulator polynomial $f_{\mathsf{liab}}(X)$ over the user domain. Its first evaluation is the total liabilities: $f_{\mathsf{liab}}(\omega^0) = \sum_{i=1}^{\mu} \mathfrak{bal}_i$. As in the asset argument, this total is not revealed in the clear. Instead, the prover opens it through KZG_{cm} and publishes the Pedersen commitment

$$\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0)) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f_{\mathsf{liab}}; f_{\mathsf{liab}}(\omega^0); \omega^0).$$

Thus $\pi_{\mathsf{liabilities}}$ proves that all liabilities are non-negative and accumulated correctly, while hiding both individual balances from the public auditor and the total liability value. For fixed precision k, proof size and auditor verification depend on k, not on the number of users; the prover's work remains linear in μ. The full protocol is given in Protocol 4.

4.2 The π_{users} Argument

The public $\pi_{\mathsf{liabilities}}$ argument proves that the committed liability total is computed from some list of non-negative balances. Each user must also be able to verify that their own balance is included in that list. The π_{users} argument provides this individualized check.

For each account, the exchange commits to a user identifier and a balance at the same index. The balance is already represented by p_1, since $p_1(\omega^{i-1}) = \mathfrak{bal}_i$. The exchange additionally commits to an identifier polynomial $f_{\mathsf{uid}}(X)$, where $f_{\mathsf{uid}}(\omega^{i-1}) = \mathfrak{uid}_i$. To check inclusion, user i receives openings of both

- Inputs
  - Public commitments $\mathcal{C}_{f_{\mathsf{uid}}}$ and $\mathcal{C}_{p_1}$.
  - For user i, the claimed identifier $\mathfrak{uid}_i$, claimed balance $\mathfrak{bal}_i$, and index i.
- Protocol
  1. $\mathcal{P}$ interpolates the identifier polynomial $f_{\mathsf{uid}}(X)$ such that $f_{\mathsf{uid}}(\omega^{i-1}) = \mathfrak{uid}_i \quad \text{for all } i \in [\mu]$, and publishes $\mathcal{C}_{f_{\mathsf{uid}}}$. The commitment $\mathcal{C}_{p_1}$ is produced by $\pi_{\mathsf{liabilities}}$.
  2. For user i, $\mathcal{P}$ opens $f_{\mathsf{uid}}(\omega^{i-1}) \quad \text{and} \quad p_1(\omega^{i-1})$ together with the corresponding KZG opening proofs.
  3. $\mathcal{V}$ outputs **acc** if and only if
     (a) the opening of f_{uid} at ω^{i-1} verifies and equals the user's identifier $\mathfrak{uid}_i$;
     (b) the opening of p_1 at ω^{i-1} verifies and equals the user's balance $\mathfrak{bal}_i$.

Protocol 5: The π_{users} proof lets each user verify that their balance is included at an index bound to their unique identifier, thereby mitigating clash attacks.

- Inputs
  - The committed asset total $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0))$ from π_{assets}.
  - The committed liability total $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0))$ from $\pi_{\mathsf{liabilities}}$.
- Protocol
  1. $\mathcal{P}$ computes $\mathsf{eq} = f_{\mathsf{assets}}(\omega^0) - f_{\mathsf{liab}}(\omega^0)$.
  2. $\mathcal{P}$ constructs a polynomial $f_{\mathsf{eq}}(X)$ such that $f_{\mathsf{eq}}(\omega^0) = \mathsf{eq}$, and publishes $\mathcal{C}_{f_{\mathsf{eq}}}$.
  3. $\mathcal{P}$ generates a range proof showing that $f_{\mathsf{eq}}(\omega^0) \in [0, 2^k)$.
  4. $\mathcal{P}$ opens the committed evaluation of f_{eq} at ω^0, i.e., $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{eq}}(\omega^0)) \leftarrow \mathsf{KZG}_{cm}.\mathsf{Prove}(f_{\mathsf{eq}}; f_{\mathsf{eq}}(\omega^0); \omega^0)$.
  5. $\mathcal{V}$ outputs **acc** if and only if
     (a) $\mathsf{KZG}_{cm}.\mathsf{Verify}(\mathcal{C}_{f_{\mathsf{eq}}}; \mathbf{C}_{\mathtt{bls}}(f_{\mathsf{eq}}(\omega^0)); \omega^0)$ returns 1.
     (b) The range proof for $f_{\mathsf{eq}}(\omega^0)$ accepts.
     (c) $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0)) = \mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0)) \cdot \mathbf{C}_{\mathtt{bls}}(f_{\mathsf{eq}}(\omega^0))$.

Protocol 6: The π_{solvency} proof demonstrates that the total assets exceed the total liabilities by a non-negative integer, called the equity.

$f_{\mathsf{uid}}(\omega^{i-1})$ and $p_1(\omega^{i-1})$. The user accepts only if the opened identifier matches their own identifier and the opened balance matches their account balance.

The identifier binding prevents clash attacks: if two users have the same balance, the exchange cannot safely show both users the same balance opening unless the identifier opening also matches both users. As in prior proofs of solvency, the identifier may be derived from account-specific data, such as a hash of the user's account name or email address [1, 8]. The full protocol is given in Protocol 5.

4.3 The π_{solvency} Argument

The final argument proves solvency by linking the committed asset total from π_{assets} to the committed liability total from $\pi_{\mathsf{liabilities}}$. At the end of π_{assets}, the total assets are committed as

$$\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0)).$$

At the end of $\pi_{\mathsf{liabilities}}$, the total liabilities are committed as

$$\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0)).$$

The prover computes the equity

$$\mathsf{eq} = f_{\mathsf{assets}}(\omega^0) - f_{\mathsf{liab}}(\omega^0)$$

and commits to it as the first evaluation of a polynomial $f_{\mathsf{eq}}(X)$. The verifier checks two conditions: first, that $f_{\mathsf{eq}}(\omega^0)$ is opened correctly through KZG_{cm}; and second, using the homomorphism of Pedersen commitments, that

$$\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0)) = \mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0)) \cdot \mathbf{C}_{\mathtt{bls}}(f_{\mathsf{eq}}(\omega^0)).$$

A range proof on $f_{\mathsf{eq}}(\omega^0)$ then establishes that $\mathsf{eq} \geq 0$. Together, these checks prove that total assets are at least total liabilities without revealing either total.

– Inputs
  - Private input to $\mathcal{P}$: values $\langle x_1, x_2, \ldots, x_\mu \rangle$ to be proved in $[0, 2^k)$.
  - Public parameters: the bit precision k, the number of values μ, and the domain $H_{\mathsf{rp}} = \{\omega^0, \omega^1, \ldots, \omega^{\mu-1}\}$.
– Protocol
  1. For each $i \in [\mu]$, $\mathcal{P}$ computes the binary decomposition of x_i:

$$x_i = \sum_{j=1}^{k} 2^{j-1} \cdot z_j^{(x_i)}, \qquad z_j^{(x_i)} \in \{0, 1\}.$$

  2. $\mathcal{P}$ puts these bits into accumulator form. For each $i \in [\mu]$, define

$$\chi_k^{(x_i)} = z_k^{(x_i)} \quad \text{and} \quad \chi_j^{(x_i)} = 2\chi_{j+1}^{(x_i)} + z_j^{(x_i)} \quad \text{for } j \in [1, k).$$

  Thus $\chi_1^{(x_i)} = x_i$.

$$\begin{bmatrix} \chi_1^{(x_1)} & \chi_2^{(x_1)} & \chi_3^{(x_1)} & \cdots & \chi_k^{(x_1)} \\ \chi_1^{(x_2)} & \chi_2^{(x_2)} & \chi_3^{(x_2)} & \cdots & \chi_k^{(x_2)} \\ \chi_1^{(x_3)} & \chi_2^{(x_3)} & \chi_3^{(x_3)} & \cdots & \chi_k^{(x_3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \chi_1^{(x_\mu)} & \chi_2^{(x_\mu)} & \chi_3^{(x_\mu)} & \cdots & \chi_k^{(x_\mu)} \end{bmatrix}$$

  3. For each column $j \in [k]$, let

$$\overline{p}_j = \left\langle \chi_j^{(x_1)}, \chi_j^{(x_2)}, \ldots, \chi_j^{(x_\mu)} \right\rangle.$$

  The intended constraints are, for every $i \in [\mu]$,

$$\overline{p}_1[i] = x_i,$$

$$\overline{p}_j[i] - 2\overline{p}_{j+1}[i] \in \{0, 1\} \quad \text{for } j \in [1, k),$$

  and

$$\overline{p}_k[i] \in \{0, 1\}.$$

  4. $\mathcal{P}$ interpolates each column as a polynomial $p_j(X)$ over H_{rp}, so that

$$p_j(\omega^{i-1}) = \overline{p}_j[i] \quad \text{for all } i \in [\mu],\ j \in [k],$$

  and publishes commitments $\mathcal{C}_{p_1}, \mathcal{C}_{p_2}, \ldots, \mathcal{C}_{p_k}$.
  5. $\mathcal{P}$ and $\mathcal{V}$ run `PIOP` to prove the following relations over H_{rp}:

$$\mathcal{R}_j = \{p_j, p_{j+1} \mid [p_j(X) - 2p_{j+1}(X)] \cdot [1 - (p_j(X) - 2p_{j+1}(X))] = 0\}, \quad j \in [1, k),$$
$$\mathcal{R}_k = \{p_k \mid p_k(X) \cdot [1 - p_k(X)] = 0\}.$$

  6. $\mathcal{V}$ outputs **acc** if and only if the above `PIOP` proof accepts.

Protocol 7: The multi-value range proof demonstrates that each committed value is a non-negative integer less than 2^k. The first column polynomial p_1 contains the proved values.

5 Security Analysis

We analyze Xiezhi under the standard assumptions used by its components: discrete-log hardness in `secp256k1` and `bls12-381`, the hiding and binding properties of Pedersen commitments, the binding and evaluation soundness of KZG commitments, and the security of the KZG evaluation interfaces KZG_{zk} and KZG_{cm} from Section 2.4. For non-interactive instantiations, we apply Fiat–Shamir in the random-oracle model and require strong transcript binding and domain separation [9, 10].

5.1 Security Definition

We adapt the proof-of-solvency definition of Provisions [1]. Let $\mathcal{A}'$ be the public anonymity set of blockchain keys, modeled as a map

$$(y = g^x) \mapsto \mathsf{bal}(y),$$

where y is a public key and $\mathsf{bal}(y)$ is the asset value controlled by that key on the blockchain. Let $\mathcal{A} \subseteq \mathcal{A}'$ be the subset of keys controlled by the exchange. Let $\mathcal{L}$ be a map $\mathfrak{uid} \mapsto \ell$, where ℓ is the liability owed to the user identified by $\mathfrak{uid}$. Balances are non-negative integers bounded by a public value MaxETH. The size of $\mathcal{A}'$ is public; the sizes of $\mathcal{A}$ and $\mathcal{L}$ may be hidden or only upper-bounded, depending on the deployment parameters.

Definition 1 (Valid Pair). *We say that $(\mathcal{A}, \mathcal{L})$ is a valid pair with respect to* MaxETH *if*

$$\sum_{y \in \mathcal{A}} \mathcal{A}[y] \geq \sum_{\mathfrak{uid} \in \mathcal{L}} \mathcal{L}[\mathfrak{uid}],$$

and, for every $\mathfrak{uid} \in \mathcal{L}$,

$$0 \leq \mathcal{L}[\mathfrak{uid}] \leq \mathsf{MaxETH}.$$

Consider a protocol $\mathsf{ProveSolvency}$ between an exchange $\mathcal{E}$ and a user $\mathcal{U}$. The exchange is given

$$(1^\lambda, \mathsf{MaxETH}, \mathcal{A}, \mathcal{L}, \mathcal{A}'),$$

and the user is given

$$(1^\lambda, \mathsf{MaxETH}, \mathcal{A}', \mathfrak{uid}, \ell).$$

The user's output is either ACCEPT or REJECT.

Definition 2 (Privacy-Preserving Proof of Solvency). *A probabilistic polynomial-time protocol* $\mathsf{ProveSolvency}$ *is a privacy-preserving proof of solvency if it satisfies the following properties.*

1. *Correctness. If $(\mathcal{A}, \mathcal{L})$ is a valid pair and $\mathcal{L}[\mathfrak{uid}] = \ell$, then the honest user accepts with probability* 1.
2. *Soundness. If $(\mathcal{A}, \mathcal{L})$ is not a valid pair, or if $\mathcal{L}[\mathfrak{uid}] \neq \ell$, then the user rejects except with negligible probability.*
3. *Ownership. If the protocol accepts, then the exchange must know the private keys for the asset keys it claims. More precisely, there exists an extractor that, given rewindable access to the exchange, extracts x for every selected public key $y = g^x \in \mathcal{A}$.*
4. *Privacy. A dishonest user interacting with an honest exchange learns nothing about $(\mathcal{A}, \mathcal{L})$ beyond solvency, its own liability $\mathcal{L}[\mathfrak{uid}]$, and any intentionally public parameters such as upper bounds on $|\mathcal{A}|$ or $|\mathcal{L}|$. Equivalently, the user cannot distinguish an execution using $(\mathcal{A}, \mathcal{L})$ from an execution using any other equally sized valid pair $(\widehat{\mathcal{A}}, \widehat{\mathcal{L}})$ such that*

$$\widehat{\mathcal{L}}[\mathfrak{uid}] = \mathcal{L}[\mathfrak{uid}].$$

5.2 Auxiliary Claims

For the per-key ownership proof, define the relation

$$\mathcal{R}_{\mathsf{key},i} = \mathcal{R}_{1,i} \vee \mathcal{R}_{0,i},$$

where

$$\begin{aligned} \mathcal{R}_{1,i} :&\quad \mathsf{pk}_i = g_{\mathtt{secp}}^{\mathsf{sk}_i} \ \wedge\ \mathbf{C}_{\mathtt{bls}}(s_i) = g_{\mathtt{bls}} h_{\mathtt{bls}}^{r_i}, \\ \mathcal{R}_{0,i} :&\quad \mathbf{C}_{\mathtt{bls}}(s_i) = h_{\mathtt{bls}}^{r_i}. \end{aligned}$$

The first branch states that the selector opens to 1 and the prover knows the corresponding `secp256k1` secret key. The second branch states that the selector opens to 0.

Theorem 1. *Protocol 1 is a Σ-protocol for $\mathcal{R}_{\mathsf{key},i}$. It is complete, special sound, and special honest-verifier zero knowledge.*

Proof. Completeness follows by direct inspection of Protocol 1. If $s_i = 1$, the prover knows both the `secp256k1` secret key and the opening of $\mathbf{C}_{\texttt{bls}}(s_i)$ as a commitment to 1. If $s_i = 0$, the prover knows the opening of $\mathbf{C}_{\texttt{bls}}(s_i)$ as a commitment to 0. In either case, the verification equations hold.

For special soundness, consider two accepting transcripts with the same first messages (t_1, t_2, t_3) and different challenges:

$$(e, e_0, e_1, z_1, z_2, z_3) \quad \text{and} \quad (e', e'_0, e'_1, z'_1, z'_2, z'_3),$$

where $e \neq e'$. Since $e = e_0 \oplus e_1$ and $e' = e'_0 \oplus e'_1$, at least one branch challenge differs. If $e_0 \neq e'_0$, the accepting equations yield

$$\mathsf{sk}_i = \frac{z_1 - z'_1}{e_0 - e'_0}, \qquad r_i = \frac{z_2 - z'_2}{e_0 - e'_0}.$$

Thus the extractor obtains a witness for $\mathcal{R}_{1,i}$. If $e_1 \neq e'_1$, the accepting equations yield

$$r_i = \frac{z_3 - z'_3}{e_1 - e'_1},$$

which is a witness for $\mathcal{R}_{0,i}$. Hence an extractor obtains a valid witness for one branch of the disjunction.

For special honest-verifier zero knowledge, given a challenge e, the simulator samples e_0, e_1 such that $e = e_0 \oplus e_1$, samples z_1, z_2, z_3 uniformly from the appropriate scalar fields, and defines

$$\begin{aligned} t_1 &= g_{\texttt{secp}}^{z_1} / \mathsf{pk}_i^{e_0}, \\ t_2 &= g_{\texttt{bls}}^{e_0} h_{\texttt{bls}}^{z_2} / \mathbf{C}_{\texttt{bls}}(s_i)^{e_0}, \\ t_3 &= h_{\texttt{bls}}^{z_3} / \mathbf{C}_{\texttt{bls}}(s_i)^{e_1}. \end{aligned}$$

The simulated transcript satisfies all verification equations and has the same distribution as an honest transcript.

Corollary 1. *Using Fiat–Shamir in the random-oracle model, Protocol 1 can be compiled into a non-interactive zero-knowledge proof of knowledge for $\mathcal{R}_{\mathsf{key},i}$.*

Proof. This follows from the standard Fiat–Shamir transform for Σ-protocols [11, 12, 13]. Strong Fiat–Shamir discipline is required so that all public inputs, commitments, branch statements, and prior messages are bound into the challenge [9, 10].

Theorem 2. *The polynomial protocol instantiated with KZG_{zk} is complete, knowledge sound in the algebraic group model, and honest-verifier zero knowledge.*

Proof. Completeness follows because an honest prover commits to the prescribed polynomials, opens them correctly, and constructs quotient polynomials for relations that hold on the evaluation domain.

For knowledge soundness, KZG binding and evaluation soundness bind the prover to committed polynomials. If a claimed polynomial relation does not hold, then the corresponding relation polynomial is nonzero. By the Schwartz–Zippel lemma, such a polynomial evaluates to zero at a verifier-chosen random point only with negligible probability. Thus an accepting proof implies that the committed polynomials satisfy the claimed relations, except with negligible probability.

Honest-verifier zero knowledge follows from the blinding in KZG_{zk}. The prover commits to randomized polynomials that agree with the witness polynomials on the evaluation domain but hide openings at verifier-chosen points outside the domain. Therefore, the verifier learns the validity of the relations but not the hidden polynomial evaluations beyond what is explicitly opened.

5.3 Security of the Components

Corollary 2. *π_{keys} is complete, sound, and honest-verifier zero knowledge.*

Proof. Completeness follows from Protocol 2. An honest prover commits to f_{sel}, opens each value $f_{\mathsf{sel}}(\omega^{i-1})$ to a Pedersen commitment through KZG_{cm}, and runs Protocol 1 for each index.

For soundness, KZG_{cm} binds each $\mathbf{C}_{\texttt{bls}}(s_i)$ to the corresponding evaluation of $\mathcal{C}_{f_{\mathsf{sel}}}$. Protocol 1 then proves that $\mathbf{C}_{\texttt{bls}}(s_i)$ either opens to 0, or opens to 1 together with knowledge of the `secp256k1` secret key for pk_i. Pedersen binding prevents a commitment from opening both ways, and KZG evaluation soundness prevents changing the selector polynomial after commitment. Hence every selected index is backed by extractable knowledge of the corresponding secret key.

Zero knowledge follows from the hiding of the Pedersen commitments, the zero-knowledge property of KZG_{cm}, and the special HVZK property of Protocol 1. The verifier learns that every selected key is owned, but not which keys are selected.

Corollary 3. *π_{assets} is complete, knowledge sound in the algebraic group model, and honest-verifier zero knowledge.*

Proof. Completeness follows from Protocol 3. Given the certified selector f_{sel} and the public balance polynomial f_{bal}, an honest prover constructs f_{assets} so that

$$f_{\mathsf{assets}}(\omega^0) = \sum_{i=1}^{\kappa} s_i \mathfrak{bal}_i.$$

Knowledge soundness follows from Theorem 2. The two relations in Protocol 3 force f_{assets} to be the correct accumulator for the pointwise product $f_{\mathsf{bal}} f_{\mathsf{sel}}$. The KZG_{cm} check binds $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{assets}}(\omega^0))$ to the committed evaluation. Hence an accepting proof commits to the correct total assets.

Zero knowledge follows from the zero-knowledge property of the polynomial protocol and from KZG_{cm}. The selected addresses and the total asset value remain hidden.

Corollary 4. *$\pi_{\mathsf{liabilities}}$ is complete, knowledge sound in the algebraic group model, and honest-verifier zero knowledge.*

Proof. Completeness follows from Protocol 4. An honest prover first runs Protocol 7, producing committed polynomials $p_1, \ldots, p_k$ such that

$$p_1(\omega^{i-1}) = \mathfrak{bal}_i \quad \text{and} \quad \mathfrak{bal}_i \in [0, 2^k).$$

The prover then constructs f_{liab} as an additive accumulator over p_1, so

$$f_{\mathsf{liab}}(\omega^0) = \sum_{i=1}^{\mu} \mathfrak{bal}_i.$$

Knowledge soundness follows in two steps. First, the range-proof relations enforce that each row of $p_1, \ldots, p_k$ is a valid k-bit accumulator. Second, the accumulator relations for f_{liab} force its first evaluation to be the sum of all entries of p_1. The KZG_{cm} check binds $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{liab}}(\omega^0))$ to this committed evaluation.

Zero knowledge follows from the zero-knowledge property of the polynomial protocol and from KZG_{cm}. The public auditor learns that all liabilities are non-negative and correctly accumulated, but not the individual balances or the total liability value.

Corollary 5. *π_{users} is complete, knowledge sound in the algebraic group model, and zero knowledge with respect to all non-opened entries.*

Proof. Completeness follows from Protocol 5. The honest exchange opens $f_{\mathsf{uid}}(\omega^{i-1})$ and $p_1(\omega^{i-1})$ for user i.

Knowledge soundness follows from KZG evaluation soundness. The user accepts only if the opened identifier equals $\mathfrak{uid}_i$ and the opened balance equals $\mathfrak{bal}_i$ at the same index. Since identifiers are unique, this prevents clash attacks in which the exchange attempts to show the same balance entry to multiple users with different identities.

Privacy follows from the hiding of the polynomial commitments and the zero-knowledge opening interface. User i learns only the two evaluations opened at its own index: its identifier and its balance. No other user's identifier or balance is revealed.

Corollary 6. *π_{solvency} is complete, sound, and honest-verifier zero knowledge.*

Proof. Completeness follows from Protocol 6. If the exchange is solvent, then there exists a non-negative equity value satisfying

$$f_{\mathsf{assets}}(\omega^0) = f_{\mathsf{liab}}(\omega^0) + f_{\mathsf{eq}}(\omega^0).$$

Thus the Pedersen commitment equation and the range proof both accept.

For soundness, Corollary 3 binds the asset commitment to the correct asset total, and Corollary 4 binds the liability commitment to the correct liability total. The KZG_{cm} check binds $\mathbf{C}_{\mathtt{bls}}(f_{\mathsf{eq}}(\omega^0))$ to the committed equity polynomial, and the range proof enforces

$$f_{\mathsf{eq}}(\omega^0) \geq 0.$$

By the binding of Pedersen commitments, the homomorphic check in Protocol 6 implies that total assets equal total liabilities plus a non-negative equity value. Hence total assets are at least total liabilities.

Zero knowledge follows from the hiding of the Pedersen commitments, the zero-knowledge property of the range proof, and KZG_{cm}. The verifier learns only that the committed totals satisfy the solvency equation with non-negative equity.

5.4 Main Theorem

Theorem 3. *Xiezhi is a privacy-preserving proof of solvency with respect to Definition 2.*

Proof. We verify the four properties of Definition 2.

Correctness. Assume $(\mathcal{A}, \mathcal{L})$ is valid and the user's claimed balance is correct. By Corollary 2, the honest exchange can certify the selector for its owned keys. By Corollary 3, it can prove that the selected public blockchain balances accumulate to the committed asset total. By Corollary 4, it can prove that all user balances are non-negative and accumulate to the committed liability total. By Corollary 5, each user can verify that its own balance is included at the correct identifier. Finally, since the pair is valid, the equity is non-negative, so Corollary 6 implies that the solvency proof accepts.

Soundness. Suppose the protocol accepts for an invalid pair. If the exchange claims assets without knowing the corresponding private keys, this contradicts Corollary 2. If the asset total is not the sum of selected public balances, this contradicts Corollary 3. If any liability is negative or outside the permitted range, or if the liability total is not the sum of the committed balances, this contradicts Corollary 4. If a user is shown an incorrect balance, this contradicts Corollary 5. Finally, if total assets are less than total liabilities but the final solvency check accepts, this contradicts Corollary 6. Therefore an invalid proof can be accepted only with negligible probability. The overall soundness error is bounded by the sum of the underlying KZG, polynomial-check, range-proof, and Σ-protocol errors; in particular,

$$\varepsilon_{\mathsf{snd}} \leq \varepsilon_{\mathsf{KZG}} + \varepsilon_{\mathsf{SZ}} + \varepsilon_{\mathsf{RP}} + 2^{-t}.$$

Ownership. If the protocol accepts, then every selected key has a selector value $s_i = 1$ certified by π_{keys}. By the special soundness of Protocol 1, an extractor can recover sk_i for each selected public key

$$\mathsf{pk}_i = g_{\mathsf{secp}}^{\mathsf{sk}_i}.$$

Thus the exchange has extractable knowledge of the private keys for the assets it claims.

Privacy. The selector values in π_{keys} are hidden by Pedersen commitments and the zero-knowledge cross-curve proof. The asset proof hides both the selected addresses and the asset total. The liability proof hides individual balances and the liability total from the public auditor, while π_{users} reveals to each user only its own identifier and balance. The final solvency proof reveals only that the committed asset total equals the committed liability total plus a non-negative committed equity value. Hence a dishonest user learns no more than solvency, its own liability, and any public size or parameter information allowed by Definition 2.

Therefore, Xiezhi is a privacy-preserving proof of solvency.

6 Performance Evaluation

6.1 Implementation and Experimental Setup

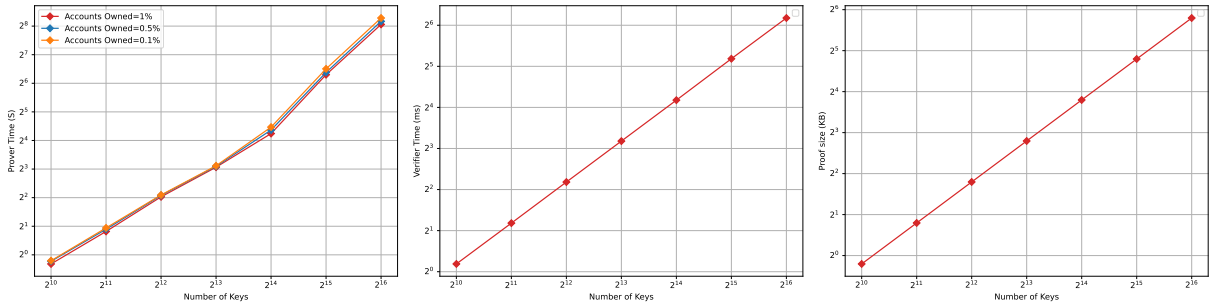
We implemented a proof-of-concept prototype of Xiezhi in Rust using the arkworks library[7]. The implementation is publicly available on GitHub[8]. We use `bls12-381` for KZG commitments and `secp256k1` for blockchain-style signing keys. The experiments were run on a commodity workstation with an Intel i9-13900KF CPU and 32 GB of memory.

All experimental inputs were generated locally. For the asset-side experiments, we generated synthetic `secp256k1` key pairs and sampled balances uniformly from $[1, 2^{64})$. For the liability-side experiments, we sampled balances from $[1, 2^{8})$, $[1, 2^{16})$, $[1, 2^{32})$, and $[1, 2^{64})$, corresponding to different balance precisions. We evaluated asset anonymity sets of size $2^{10}, 2^{11}, \ldots, 2^{16}$ and liability sets of size $2^{10}, 2^{11}, \ldots, 2^{20}$. We use a larger range of user counts for $\pi_{\mathsf{liabilities}}$ because π_{keys} is the dominant cost on the asset side. For each parameter setting, we ran ten trials and report the trimmed mean after discarding the maximum and minimum measurements.

6.2 Empirical Results

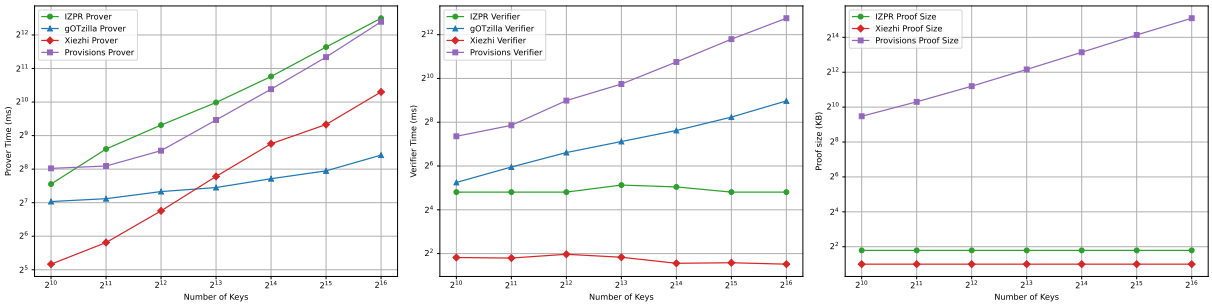
Figure 2 reports the cost of π_{keys}, the reusable key-ownership phase. This phase is not succinct: proof size and verifier work scale with the number of keys in the anonymity set. Its main benefit is that it can be reused across proof-of-solvency cycles until the exchange changes the asset keys it claims. Moreover, the per-key proofs are independent, so π_{keys} is naturally parallelizable across machines.

[7] `https://github.com/arkworks-rs`
[8] `https://github.com/Shvier/proof-of-solvency`



(a) Prover time (b) Verifier time (c) Proof size

Fig. 2: Performance of π_{keys}. The proof is linear in verifier work and proof size, and the per-key checks are independent and parallelizable. This phase is reusable across proof cycles unless the exchange changes the set of asset keys it claims.



(a) Prover time (b) Verifier time (c) Proof size

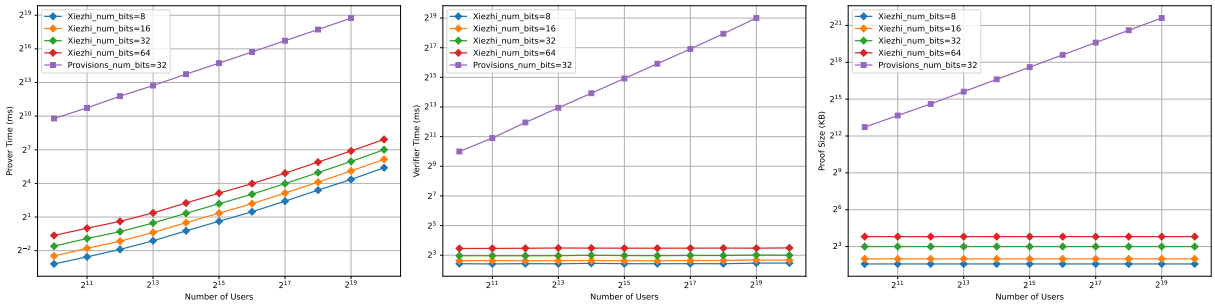
Fig. 3: Performance of the recurring π_{assets} argument compared with prior PoA protocols. The comparison shows that Xiezhi retains constant proof size and constant argument-verification time while avoiding circuit-level `secp256k1` arithmetic. For gOTzilla, the implementation does not expose proof size directly because the protocol is interactive; the authors report logarithmic proof size.

Scheme	**Proving time**	**Verifying time**		**Proof size**		**Full PoS**
		π_{users}	$\pi_{\mathsf{liabilities}}$	π_{users}	$\pi_{\mathsf{liabilities}}$	
Provisions [1]	$O(\mu)$	$O(1)$	$O(\mu)$	$O(1)$	$O(\mu)$	✓
Bulletproofs [2]	$O(\mu)$	$O(1)$	$O(\mu)$	$O(1)$	$O(\log \mu)$	✓
CompositeNIZK [14]	$O(\mu)$	$O(1)$	$O(\mu)$	$O(1)$	$O(\mu)$	✓
DAPOL+ [4]	$O(\mu \log \mu)$	$O(\log \mu)$	$O(1)$	$O(\log \mu)$	$O(1)$	✗
SSVT-based [5]	$O(\log_\lambda \mu)$	$O(\log_\lambda \mu)$	$O(1)$	$O(\log_\lambda \mu)$	$O(1)$	✗
Notus [6]	$O(\mu \log \mu)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	✗
Xiezhi (**Ours**)	$O(k \cdot \mu)$	$O(1)$	$O(k)$	$O(1)$	$O(k)$	✓

Table 1: Comparison with prior PoL schemes. Gray rows are liability-only schemes and therefore do not directly compete as full proof-of-solvency systems. Here μ is the number of users and k is the bit precision of each balance, which is explicit in the table but fixed in a deployment. For SSVT-based proofs [5], λ is the arity of the Verkle tree.

Figure 3 reports the cost of the recurring π_{assets} argument and compares it with prior PoA protocols. Once the certified selector commitment and the public balance commitment are fixed, Xiezhi has constant proof size and constant argument-verification time. If the auditor reconstructs the balance commitment directly from blockchain state, that public preprocessing is linear in the anonymity-set size.

Figure 4 reports the cost of $\pi_{\mathsf{liabilities}}$ for different balance precisions. For fixed precision k, prover time grows linearly with the number of users, while auditor verification time and proof size are independent of the number of users. As expected, verification time and proof size scale with k, the number of column polynomials used in the range proof.



(a) Prover time (b) Verifier time (c) Proof size

Fig. 4: Performance of $\pi_{\mathsf{liabilities}}$ for different balance precisions, compared with Provisions. For fixed k, prover time is linear in the number of users, while auditor verification time and proof size are independent of the number of users.

6.3 Theoretical Performance

Table 1 summarizes the liability-side comparison; the dominant asset-side costs are as follows. With the prototype's naive per-point KZG openings, reusable π_{keys} proving costs $O(\kappa^2)$, while its verifier and proof are $O(\kappa)$. The recurring π_{assets} prover is $O(\kappa)$, with constant proof size and argument verification once the public balance and certified selector commitments are fixed; reconstructing the balance commitment from the chain is $O(\kappa)$ public preprocessing. For k-bit balances, $\pi_{\mathsf{liabilities}}$ proving costs $O(k\mu)$, while auditor verification and proof size are $O(k)$; each user's inclusion proof remains constant size and constant time.

7 Related Work

Provisions gave the first complete zero-knowledge construction [1]; later work improved range proofs and liability aggregation or inclusion proofs [2, 3, 4, 5, 6]. Those liability-focused systems avoid the asset-side mismatch between deployed `secp256k1` keys and succinct systems over pairing-friendly curves. Publishing addresses sacrifices privacy, whereas circuit-level `secp256k1` verification, as in IZPR, raises prover cost [7]. COPZ studies cross-group equality of discrete logarithms [15], and gOTzilla gives an MPC-in-the-head asset proof without a complete solvency protocol [16]. Xiezhi instead transfers only the *fact of ownership* into a hidden `bls12-381` selector, retaining address privacy and an end-to-end statement without generic-circuit `secp256k1` arithmetic.

8 Conclusion and Future Research

Xiezhi provides an end-to-end private proof of key ownership, assets, user inclusion, liabilities, and solvency. Its cross-curve bridge certifies a hidden `bls12-381` selector without translating `secp256k1` keys, enabling a constant-size recurring asset argument. Its column-wise range proof respects `bls12-381` domain limits and, for fixed precision, keeps liability proof size and auditor work independent of the user count. The prototype confirms that the reusable ownership phase is practical and recurring proofs remain small and fast to verify.

Future research. The main opportunities are to batch the committed point checks in π_{keys} (e.g., with aggregated KZG openings [17]), use lookup or folding techniques to compress polynomial relations, reduce reliance on a universal setup [18], and support hashed identifiers, multisignature, threshold, and contract-controlled assets. Deployment work should also strengthen user-side checking and investigate on-chain verification and enforcement where pairing costs permit it.

9 Extension for Hashed or Address-Derived Keys

The main π_{keys} protocol is stated for an anonymity set of public keys, which is the cleanest setting for presenting the cross-curve ownership-to-selector bridge. In practice, an exchange or auditor may instead work with hashed or address-derived identifiers, such as $\mathsf{addr}_i = \mathcal{H}(\mathsf{pk}_i)$, rather than raw public keys.

The selector-polynomial part of π_{keys} is unchanged. The exchange still commits to a hidden selector polynomial f_{sel}, opens each evaluation $f_{\mathsf{sel}}(\omega^{i-1})$ to a Pedersen commitment $\mathbf{C}_{\mathtt{bls}}(s_i)$ through KZG_{cm},

and proves that every selected entry is owned. Only the per-index ownership proof changes: instead of proving knowledge of the secret key for a given public key, the prover shows knowledge of a secret key whose public key hashes to the advertised identifier.

For each index i, the desired relation is

$$\begin{aligned} \mathcal{R}^{\mathsf{addr}}_{\mathsf{key},i} &= \mathcal{R}^{\mathsf{addr}}_{1,i} \vee \mathcal{R}_{0,i}, \\ \mathcal{R}^{\mathsf{addr}}_{1,i}\colon \quad \mathsf{pk}_i &= g^{\mathsf{sk}_i}_{\mathtt{secp}}, \quad \mathsf{addr}_i = \mathcal{H}(\mathsf{pk}_i), \\ \mathbf{C}_{\mathtt{bls}}(s_i) &= g_{\mathtt{bls}} h^{r_i}_{\mathtt{bls}}, \qquad \mathcal{R}_{0,i}\colon\ \mathbf{C}_{\mathtt{bls}}(s_i) = h^{r_i}_{\mathtt{bls}}. \end{aligned}$$

Thus a selected entry proves knowledge of a `secp256k1` secret key whose public key hashes to the advertised identifier and that its selector commitment opens to one. An unselected entry proves only that its selector commitment opens to zero. Importantly, this branch requires neither the exchange nor the auditor to know a public-key preimage for an unselected identifier.

The selected branch is a composite statement: it combines the algebraic relation $\mathsf{pk}_i = g^{\mathsf{sk}_i}_{\mathtt{secp}}$ with the non-algebraic relation $\mathsf{addr}_i = \mathcal{H}(\mathsf{pk}_i)$. We instantiate it using the composite-statement techniques of Agrawal *et al.* [14], which provide Σ-protocol-compatible tools for combining algebraic and non-algebraic statements. The resulting proof is then OR-composed with the Pedersen proof for $\mathbf{C}_{\mathtt{bls}}(s_i) = \mathbf{C}_{\mathtt{bls}}(0)$, exactly as in the main π_{keys} protocol.

Consequently, the ownership-to-selector bridge and the subsequent π_{assets} argument require no changes: the extension only replaces the selected branch of each ownership proof. It is not required for the core construction or used in our performance evaluation, but shows how the same bridge supports anonymity sets represented by hashed or address-derived identifiers.

References

1. G. G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh, "Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 720–731. [Online]. Available: https://doi.org/10.1145/2810103.2813674
2. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 315–334.
3. J. Doerner, A. Shelat, and D. Evans, "Zeroledge: Proving solvency with privacy," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:211105655
4. Y. Ji and K. Chalkias, "Generalized proof of liabilities," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3465–3486. [Online]. Available: https://doi.org/10.1145/3460120.3484802
5. F. Falzon, K. Elkhiyaoui, Y. Manevich, and A. De Caro, "Short privacy-preserving proofs of liabilities," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1805–1819. [Online]. Available: https://doi.org/10.1145/3576915.3616645
6. J. Xin, A. Haghighi, X. Tian, and D. Papadopoulos, "Notus: Dynamic proofs of liabilities from zero-knowledge RSA accumulators," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/xin
7. T. Conley, N. Diaz, D. Espada, A. Kuruvilla, S. Mayne, and X. Fu, "Izpr: Instant zero knowledge proof of reserve," in *Financial Cryptography and Data Security. FC 2024 International Workshops*, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:266345810
8. K. Chalkias, P. Chatzigiannis, and Y. Ji, "Broken proofs of solvency in blockchain custodial wallets and exchanges," in *Financial Cryptography and Data Security. FC 2022 International Workshops*, S. Matsuo, L. Gudgeon, A. Klages-Mundt, D. Perez Hernandez, S. Werner, T. Haines, A. Essex, A. Bracciali, and M. Sala, Eds. Cham: Springer International Publishing, 2023, pp. 106–117.
9. D. Bernhard, O. Pereira, and B. Warinschi, "How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios," in *Advances in Cryptology – ASIACRYPT 2012*, X. Wang and K. Sako, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 626–643.
10. Q. Dao, J. Miller, O. Wright, and P. Grubbs, "Weak fiat-shamir attacks on modern proof systems," in *2023 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 199–216. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179408
11. A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.
12. I. Damgard, "On σ-protocols," 2010. [Online]. Available: https://www.cs.au.dk/~ivan/Sigma.pdf

13. U. Maurer, "Unifying zero-knowledge proofs of knowledge," in *International Conference on Cryptology in Africa*. Springer, 2009, pp. 272–286.
14. S. Agrawal, C. Ganesh, and P. Mohassel, "Non-interactive zero-knowledge proofs for composite statements," in *Advances in Cryptology – CRYPTO 2018*, H. Shacham and A. Boldyreva, Eds. Cham: Springer International Publishing, 2018, pp. 643–673.
15. M. Chase, M. Orrù, T. Perrin, and G. Zaverucha, "Proofs of discrete logarithm equality across groups," Cryptology ePrint Archive, Paper 2022/1593, 2022, https://eprint.iacr.org/2022/1593. [Online]. Available: https://eprint.iacr.org/2022/1593
16. F. Baldimtsi, P. Chatzigiannis, S. D. Gordon, P. H. Le, and D. McVicker, "gOTzilla: Efficient disjunctive zero-knowledge proofs from MPC in the head, with application to proofs of assets in cryptocurrencies," Cryptology ePrint Archive, Paper 2022/170, 2022. [Online]. Available: https://eprint.iacr.org/2022/170
17. D. Boneh, J. Drake, B. Fisch, and A. Gabizon, "Efficient polynomial commitment schemes for multiple points and polynomials," Cryptology ePrint Archive, Paper 2020/081, 2020, https://eprint.iacr.org/2020/081. [Online]. Available: https://eprint.iacr.org/2020/081
18. V. Nikolaenko, S. Ragsdale, J. Bonneau, and D. Boneh, "Powers-of-tau to the people: Decentralizing setup ceremonies," in *Applied Cryptography and Network Security*, C. Pöpper and L. Batina, Eds. Cham: Springer Nature Switzerland, 2024, pp. 105–134.