Aegis: Scalable Privacy-preserving CBDC Framework with Dynamic Proof of Liabilities

Gweonho Jeong
Hanyang University
Seoul, Republic of Korea
kwonhojeong@hanyang.ac.kr

Jaewoong Lee
Kookmin University
Seoul, Republic of Korea
leejw1496@gmail.com

Minhae Kim
Hanyang University
Seoul, Republic of Korea
minhaekim44@hanyang.ac.kr

Byeongkyu Han
Kookmin University
Seoul, Republic of Korea
byeongkyuhan@kookmin.ac.kr

Jihye Kim
Kookmin University
Zkrypto Inc
Seoul, Republic of Korea
jihyek@kookmin.ac.kr

Hyunok Oh
Hanyang Unviersity
Zkrypto Inc
Seoul, Republic of Korea
hoh@hanyang.ac.kr

ABSTRACT

Blockchain advancements, currency digitalization, and declining cash usage have fueled global interest in Central Bank Digital Currencies (CBDCs). The BIS states that the hybrid model, where central banks authorize intermediaries to manage distribution, is more suitable than the direct model. However, designing a CBDC for practical implementation requires careful consideration. First, the public blockchain raises privacy concerns due to transparency. While zk-SNARKs can be a solution, they can introduce significant proof generation overhead for large-scale transactions. Second, intermediaries that provide user-facing services on behalf of the central bank commonly performs Proof of Liabilities on customers' static liabilities. However, in real-world scenarios where user liabilities can arbitrarily increase or decrease, the static nature poses such as window attacks.

In this paper, we propose a new smart contract-based privacy-preserving CBDC framework based on zk-SNARKs, called **Aegis**. our framework introduces a transaction batching technique to enhance scalability and defines a new dynamic PoL which is near-real time. We formally define the security models for our system and provide rigorous security proofs to demonstrate its robustness. To evaluate the system's performance, we instantiate our proposed framework and measure its efficiency. The result indicates that, the end-to-end process, including proof generation for 512 transactions, takes approximately 2.8 seconds, with a gas consumption of 74,726 per user.

1 INTRODUCTION

As physical cash usage declines and the digital economy accelerates, interest in Central Bank Digital Currency (CBDC) has grown significantly. Unlike volatile digital currencies such as Bitcoin, CBDC is a direct liability of the central bank, ensuring a 1:1 exchange rate with legal tender. Furthermore, compared to cash and reserves, CBDC has digital currency characteristics, allowing it to be issued in various forms to align with policy objectives. Many countries are conducting research and pilot programs to better understand CBDC design and implementation [7, 8, 31, 36]. The Bank for International Settlements (BIS) reports that central banks worldwide are showing strong interest in CBDC development, with over 80% actively engaged in research and prototype testing [6].

To implement CBDC in practice, various perspectives have been raised regarding its specific design and operational model [3, 4, 27, 33]. The BIS(2020) [3] proposed various operational models (e.g.,direct, indirect, and hybrid) for implementing CBDCs. The **direct** model is that the central bank issues, redeems, and distributes CBDC directly, while also managing customer services. This model is considered attractive for its simplicity, as it removes reliance on intermediaries such as private sector(i.e., retail banks). But in terms of scale, speed and efficiency, the BIS(2021) [4] stated that the customer-facing side of retail payments, which involves large operational tasks (e.g., payment accounts, authorization, clearing, settlement and dispute resolution), is better handled by retail banks. Even if a central bank were to build the necessary technological capability, they argued it could still be less attractive to consumers than today's retail payment systems, because of the challenges in handling connectivity outages or offline payments. For this reason, many countries have adopted the **hybrid** model, where the private sector is delegated user-facing responsibilities by the central bank, with the goal of providing consumers with convenient, efficient and scalable payment services. For example, there are the Sand Dollar, JAM-DEX, eNaira, and others [1, 42].

As authorized intermediaries distributing CBDC to customers, private sector entities such as retail banks strive to provide customers with confidence by safely and accurately operating the database that records the assets and transaction information. If retail banks fail to provide sufficient evidence of their honest operational behavior, it could amplify consumer anxiety and lead to various depositor confidence issues. The most representative example is a bank run, which occurs when a large number of depositors hastily withdraw their funds due to concerns about bank failure: notable instances include Northern Rock [39] in 2007, Silicon Valley Bank [17] in 2023, and First Republic Bank [19] in 2023.

Therefore, various works [14, 24, 29, 34, 36] have adopted blockchain technology in retail banking system to enhance customer confidence. They proposed publishing information on a public blockchain that customers can verify. For example, it could be customer transaction information that proves the validity of their transactions, or publishing users' liabilities and IDs that demonstrate a retail bank's solvency. In this process, the immutability of the blockchain ensures that stored information cannot be altered or deleted by

anyone, while its transparency allows everyone to confirm whether transactions have been conducted accurately.

However, this transparency raises privacy concerns as it exposes sensitive information such as user identity and transfer amounts. Simply addressing privacy issues by publishing encrypted data (e.g., commitments) on a ledger is not enough, as it does not guarantee the data's authenticity. To solve this problem, zk-SNARKs are widely adopted in this context [20, 23, 37, 38]. zk-SNARKs, which offer the advantages of succinct proof sizes and fast verification times, can prove statements about the properties of owned information without revealing the underlying sensitive data. Thus, personal information is published on the ledger in committed form, and its authenticity is verified using zk-SNARKs. As a result, a privacy-preserving banking system on a public blockchain can be effectively designed using cryptographic tools such as commitments and zk-SNARKs.

Despite the efforts of the above works to construct a privacy-preserving banking system based on blockchain, there are still two unresolved issues. First, depending on the zk-SNARK scheme used, system performance can vary due to factors such as proof generation time, proof size, and verification time. In real-world scenarios involving large-scale transactions, failing to use an appropriate zk-SNARK can introduce system overhead due to significant consumption of server resources, particularly proof generation time bottlenecks. Therefore, deciding which zk-SNARK to use requires careful and meticulous consideration. Second, a type of information published on blockchain to build confidence with customers regarding the retail bank's database, most works [7, 14, 21, 29] use customers' *static* liabilities. Providing monthly independent snapshots to prove the retail bank's solvency is considered a simple scheme [44], which is why it is commonly adopted in practice [9, 32]. However, this static approach does not treat the customer's liability as an update based on previous liabilities, but rather as an entirely new one. As a result, users must verify the correctness of the updated liabilities in every epoch to ensure that transactions are accurately reflected as intended. This can lead to a type of attack known as "window-of-opportunity attacks" [44], which exploit situations when a user forgets to verify in an arbitrary epoch. On the other hand, a *dynamic* PoL, which proves the connection between the user's liabilities before and after the transaction, eliminates the need for mandatory checks in every epoch and the risk of window-of-opportunity attacks. However, it requires a more complex scheme compared to static PoL, making it difficult to apply into financial system designs.

1.1 Our results

In this paper, we propose **Aegis**, a new smart contract-based privacy-preserving CBDC framework that supports dynamic Proof-of-Liabilities (PoL).

Our framework includes authorized intermediaries called delegators that function similarly to a retail bank, handling user-facing services and distributing CBDC on behalf of the central bank. This structure makes our model compatible with the hybrid model.

In real-world scenarios requiring the efficient management of large-scale transactions, we propose a batching mechanism to enhance scalability. Specifically, we adopt approaches from a branch of zk-SNARKs, commit-and-prove SNARKs (CP-SNARKs) [2, 13, 22], which utilize commitments to offload large computations outside the circuit, resulting in fast proof generation. Using this methodology, our framework generates a single proof for multiple transactions by a delegator rather than computing separate proofs for each transaction. This design guarantees fast verification times and a constant proof size, thereby enhancing the overall performance of the framework.

Additionally, we provide dynamic PoL. In the process of verifying the proof generated using the batching mechanism described earlier, we utilize the current user balance commitment stored in the smart contract at the point before the transaction is reflected. The smart contract adds the transfer amount to this balance and calculates the updated commitment. It then includes this updated value as the required commitment for verifying the CP-SNARK proof to check whether it passes. Furthermore, since the smart contract publicly "verifies" the proof for PoL before reflecting the transaction results, in an economy with a high volume of retail transactions, which can quickly fill the batch size, near real-time dynamic PoL can be performed.

Our contributions are summarized as follows:

- We propose a novel smart-contract based privacy-preserving CBDC framework that supports dynamic PoL. Making it suitable for practical use, we introduce a new transaction batching technique derived from CP-SNARKs in our model, which allows the generation of a single proof for large-scale transactions.
- Our framework supports dynamic PoL by proving that balances are appropriately updated based on the values stored in the smart contract. Additionally, we define an extended version of dynamic PoL that allows flexible, near-real-time dynamic PoL without requiring a fixed epoch.
- To evaluate the practicality and performance of our proposed framework, we conducted experiments to measure proof generation time, verification time and gas consumption. The proof generation time refers the time taken by the server to generate a proof for a batch of transactions, while the verification time and gas consumption represent the cost of verifying the proof and updating user balances within the smart contract. Evaluation results show the efficiency of our framework in high-throughput scenarios. Specifically, for a batch of 512 transactions, the proof generation process required only 2.2 seconds. Verifying the proof and updating user balances within the smart contract took 0.6 seconds, with a total gas cost of 38.26 million. This translates to approximately 74,726 gas per user, a significantly low cost that demonstrates the scalability and efficiency of our framework in real-world CBDC deployments.

1.2 Related Work

CBDC. Before the emergence of CBDC, the initial concept for an anonymous digital currency was proposed by Chaum [16]. This system ensures the sender's anonymity through blind signatures while the recipient's identity and the transaction amount are disclosed during withdrawals. Camenisch et al. [12] proposed an e-cash solution that enforces regulations by limiting the amount a user can anonymously spend with each merchant. Baldimtsi et al. [5] introduced a transferable e-cash system that allows coins to be

transferred to different users without interaction with the bank. This approach eliminates the leakage of transaction amounts to the bank by merchants, but the size of the coins changes depending on their usage frequency, affecting linkability.

Since the advancement of blockchain technology, several proposals for anonymous cryptocurrencies have been suggested [11, 20, 23, 38, 40]. While earlier systems proposed models for enabling privacy-preserving transactions between individuals, models supporting custodial transactions have also been proposed to enhance user convenience. Solidus [14] is a distributed ledger system that hides the transaction values and the transaction graph between bank users, maintaining the public verifiability. It uses Publicly-Verifiable Oblivious RAM to achieve strong confidentiality of transaction graph. Since Solidus can only support auditing by revealing all the keys used in the system to an auditor and revealing the transaction details, zkLedger [29] proposed columnar ledger. In this structure, every bank corresponding to a column is affected by all transactions. When the bank responds to the auditor, it must total all the commitments in its column, including commitments for transactions in which it was not involved.

Finally, with the emergence of CBDC, various designs have been proposed, especially with a focus on regulation. Wüst et al. [43] introduced Platypus, a model that combines the transaction processing of e-cash with an account-based fund management mode. It ensures the anonymity of both the sender and receiver, and unlinkability between transactions. But Platypus relies completely on a single authority and lacks regulatory compliance features. Focusing on this, PEReDi [25] uses encrypted distributed ledgers to support regulatory compliance and meet requirements for AML/CFT.

Proof of Liabilities. CONIKS [28], a key verification system for end users that provides consistency and privacy for users' name-to-key bindings, uses a Merkle prefix tree to enable users to efficiently verify the absence proof for unauthorized name-to-key bindings in their namespace. However, CONIKS is difficult to monitor efficiently as each ID owner must check their data every epoch. Focusing on this, Merkle^2 [21] proposed a transparency log system that uses a data structure consisting of several top-level chronological trees, sorted by time, with each internal node in these trees having a prefix tree sorted by IDs. It allows data owners to check only the latest digest instead of checking every digest.

The applications mentioned above support only a restricted range of queries, such as insertion, removal, and look-up. Meanwhile, DAPOL+ [24] and Dagher et al. [18] enables users to demonstrate that the sums of certain user values are non-negative. Also, several database scheme has been proposed that enables users to verify the results of a wide range of SQL queries, such as sum, count, average, min, and max. IntegriDB [46] employs an ADS, allowing a data owner to delegate database storage to an untrusted server while enabling anyone to execute verifiable SQL queries on the database. They use an ADS to verify set operations with summation and support range proof, insertion, and deletion. vSQL [45] supports more general SQL queries compared to IntegriDB, while performing similarly. FalconDB [34] is a blockchain database platform with low hardware requirements for individual clients. Client nodes store only the block headers, which include digests generated from ADS, and use these digests to authenticate the results of various queries requested from the server.

The common feature of the above systems is that their system design is based on the static case. They provide epoch-based independent snapshots to the users and auditors, which are not linkable. FalconDB and IntegriDB use dynamic ADS, but they do not support efficient updates. To address the above issue, a dynamic PoL with the concept that the server shows the accurate update between consecutive epochs and their corresponding digests has been devised. TAP [35] combines a single Merkle chronological prefix tree with multiple sorted Merkle sum trees. Dynamic updates are possible by adding new sum trees to the Merkle prefix tree constructed in previous epochs. They provide zk-range proofs to verify aggregates over the data of many independent users without revealing user data. Notus [44] proposes ZK-MultiSwap, which removes the previous states of the transaction set from the zkRSA accumulator and inserts the updated states, leading to a new zkRSA accumulator. It supports dynamic PoL by showing a series of chained MultiSwaps where the updated accumulator is derived from the original accumulator in the subsequent epoch.

2 PRELIMINARIES

2.1 Notations

We use $\boldsymbol{a}$ or $\{a_i\}$ for the list of elements, which is equivalent to a vector. We denote by λ a security parameter and by $\epsilon(\cdot)$ as a negligible function. Let $\mathbb{G}$ denote a group. Given a security parameter 1^λ, a relation generator $\mathcal{RG}$ returns a decidable relation $\mathcal{R} \leftarrow \mathcal{RG}(1^\lambda)$. For $(\vec{\boldsymbol{x}}; \vec{\boldsymbol{w}}) \in \mathcal{R}$ we say $\vec{\boldsymbol{w}}$ is a witness to the statement $\vec{\boldsymbol{x}}$ being in the relation.

2.2 Commitment scheme

A commitment scheme is a tuple of algorithms (Setup, Com, VerCom): (1) $\mathsf{ck} \leftarrow \mathsf{Setup}(1^\lambda)$ generates a commitment key ck; (2) $\mathsf{cm} \leftarrow \mathsf{Com}(m; o)$ takes as input a message m and an opening o, and outputs a commitment cm. (3) $\mathsf{true/false} \leftarrow \mathsf{VerCom}(\mathsf{cm}, m; o)$ takes as input a commitment cm, a message m, and an opening o, and outputs true if $\mathsf{cm} = \mathsf{Com}(m; o)$, otherwise false. The scheme satisfies *hiding* and *binding*.

2.2.1 Pedersen vector commitment. Pedersen vector commitment for vector $\boldsymbol{w}$ of size n can be expressed succinctly with the following algorithms:

- $\mathsf{Ped.Setup}(1^\lambda)$: chooses $g \xleftarrow{\$} \mathbb{G}$ and $\boldsymbol{h} \xleftarrow{\$} \mathbb{G}^n$ from a domain $\mathcal{D}$. It outputs a commit key $\mathsf{ck} := (g, \boldsymbol{h})$.
- $\mathsf{Ped.Com}(\mathsf{ck}, \boldsymbol{w})$: chooses an opening $o \xleftarrow{\$} \mathbb{Z}_q^*$ and returns $(\mathsf{cm}, o) := ((o, \boldsymbol{w})^\top \cdot \mathsf{ck}, o)$.
- $\mathsf{Ped.VerCom}(\mathsf{ck}, \mathsf{cm}, \boldsymbol{w}, o)$: returns true if $\mathsf{cm} = (o, \boldsymbol{w})^\top \cdot \mathsf{ck}$. Otherwise, false.

The Pedersen vector commitment is perfectly hiding and computationally binding if the discrete logarithm assumption holds.

2.3 Strongly-unforgeable digital signature

A signature scheme Sig is a tuple of algorithms (Setup, KeyGen, Sign, Verify): (1) $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ takes λ as input and generates public parameters pp. (2) $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ outputs a key

pair (pk, sk). (3) $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ takes a secret key and a message as input, and returns a signature σ. (4) $\mathsf{true/false} \leftarrow \mathsf{Verify}(\mathsf{pk}, \sigma)$ returns true if the signature σ is valid; otherwise, false. It satisfies strong unforgeability against chosen-message attacks (SUF-CMA).

2.4 Succinct Non-interactive arguments of knowledge

A SNARK is a succinct non-interactive argument of knowledge for a relation $\mathcal{R}$, consisting of a tuple of algorithms $\mathbf{\Pi} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. $(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathsf{Setup}(\mathcal{R})$ takes a relation $\mathcal{R}$ as input, and generates an evaluation key ek and a verification key vk. $\pi \leftarrow \mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; \boldsymbol{w})$ takes the evaluation key ek, a statement $\boldsymbol{x}$, and a witness $\boldsymbol{w}$ as inputs, and outputs a proof π. $\mathsf{true/false} \leftarrow \mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \pi)$ takes the verification key vk, the statement $\boldsymbol{x}$ and the proof π as inputs, and outputs accept (true) or reject (false). It satisfies the following properties:

Completeness. Given a true statement, a prover $\mathcal{P}$ with a witness can convince the verifier $\mathcal{V}$. For all $\lambda \in \mathbb{N}$, for all $\mathcal{R}$ and for all $(\boldsymbol{x}; \boldsymbol{w}) \in \mathcal{R}$,

$$\Pr\left[\begin{array}{l}(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathbf{\Pi}.\mathsf{Setup}(\mathcal{R}) \\ \pi \leftarrow \mathbf{\Pi}.\mathsf{Prove}(\mathsf{ek}, \boldsymbol{x}; \boldsymbol{w})\end{array} \;:\; \mathbf{\Pi}.\mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \pi) = \mathsf{true}\right] = 1$$

Knowledge Soundness. Knowledge soundness states that a prover must know a witness and such knowledge can be efficiently extracted from π by a knowledge extractor $\mathcal{E}$. Formally, the following is *negligible* for any PPT adversary $\mathcal{A}$.

$$\Pr\left[\begin{array}{ll}(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathbf{\Pi}.\mathsf{Setup}(\mathcal{R}) & \mathbf{\Pi}.\mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}^*, \pi^*) = 1 \\ (\boldsymbol{x}^*, \pi^*) \leftarrow \mathcal{A}(\mathsf{ek}, \mathsf{vk}) \;:\; & \wedge \\ \boldsymbol{w} \leftarrow \mathcal{E}_{\mathcal{A}}(trans_{\mathcal{A}}) & (\boldsymbol{x}^*; \boldsymbol{w}) \notin \mathcal{R}\end{array}\right]$$

Succinctness. $\mathbf{\Pi}$ is succinct if the time of Verify is $(\lambda + |\boldsymbol{x}| + \log|\boldsymbol{w}|)poly(\lambda)$ and the proof is $(\lambda + \log|\boldsymbol{w}|)poly(\lambda)$.

Remark. A SNARK may also satisfy *zero-knowledge*. Shortly, zero-knowledge states that the system does not leak any information besides the truth of the statement. This is modeled by a simulator Sim that does not know the valid witness. In this case, it is called a zk-SNARK.

2.4.1 Commit-and-Prove SNARK (CP-SNARK). A commit-and-prove SNARK is a SNARK, which can efficiently prove properties of inputs committed through a commitment scheme. Given a commitment cm, a CP-SNARK for a relation $\mathcal{R}(\boldsymbol{x}; \boldsymbol{w})$ can prove knowledge of $\boldsymbol{w} := (\boldsymbol{u}, \boldsymbol{\omega}, o)$ s.t. $\mathsf{cm} = \mathsf{Com}(\boldsymbol{u}, o)$ and $\mathcal{R}(\boldsymbol{x}; \boldsymbol{\omega})$, where $\boldsymbol{u}$ (committed) and $\boldsymbol{\omega}$ (free). A CP-SNARK scheme is defined as a tuple of algorithms $\mathbf{\Pi}_{\mathsf{cp}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$.

$(\mathsf{ek}, \mathsf{vk}) \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Setup}(\mathcal{R}, \mathsf{ck})$ follows $\mathbf{\Pi}.\mathsf{Setup}$, but differs in that a commitment key ck is provided as an input. $\pi_{\mathsf{cp}} \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Prove}(\mathsf{ek}, \vec{\boldsymbol{x}}; \vec{\boldsymbol{w}}) = \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Prove}(\mathsf{ek}, \{\boldsymbol{x}, \mathsf{cm}\}; \{m, o\}, \vec{\boldsymbol{\omega}})$ is based on $\mathbf{\Pi}.\mathsf{Prove}$, but differs in that the commitments cm are included as inputs, and the witness includes both the message and the opening of the commitments. Here, $\boldsymbol{x}$ and $\vec{\boldsymbol{\omega}}$ represent the statement and the witness, respectively, which are independent of the commitments. $\mathsf{true/false} \leftarrow \mathbf{\Pi}_{\mathsf{cp}}.\mathsf{Verify}(\mathsf{vk}, \boldsymbol{x}, \mathsf{cm}, \pi)$ is the same as $\mathbf{\Pi}.\mathsf{Verify}$, except that cm is added as an input.

By leveraging the CP approach, some works [13, 22] enable efficient proving of large circuits by offloading the high-computation parts (e.g., group exponentiation) outside the circuit. Briefly, the prover commits a witness based on the circuit-dependent commitment key ck and the commitment cm can then be directly plugged into the verification equation. These schemes are referred to as commit-carrying SNARKs (cc-SNARKs) and consist of three algorithms: $\mathbf{\Pi}_{\mathsf{cc}} := (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. The algorithms are described as follows.

- $\mathsf{Setup}(\mathcal{R}) \rightarrow (\mathsf{ck}, \mathsf{ek}, \mathsf{vk})$: takes a relation $\mathcal{R}$ as input and, unlike CP-SNARKs, outputs a circuit-dependent commitment key ck, along with (ek, vk).
- $\mathsf{Prove}(\mathsf{ek}, \vec{\boldsymbol{x}}; \vec{\boldsymbol{w}}, \vec{o})$: splits the witness $\vec{\boldsymbol{w}}$ into two parts $(\boldsymbol{u}, \vec{\boldsymbol{\omega}})$: the committed witness $\boldsymbol{u}$ and the non-committed witness $\vec{\boldsymbol{\omega}}$. Outputs a proof π and a proof-dependent commitment $\widetilde{\mathsf{cm}}$, such that $\mathsf{VerCom}(\mathsf{ck}, \widetilde{\mathsf{cm}}, \boldsymbol{u}, \vec{o}) = \mathsf{true}$.
- $\mathsf{Verify}(\mathsf{vk}, \vec{\boldsymbol{x}}, \widetilde{\mathsf{cm}}, \pi) \rightarrow \mathsf{true/false}$: verifies the proof. Outputs $\mathsf{true}(accepts)$ if the proof is valid or false otherwise.

3 SYSTEM CONFIGURATION

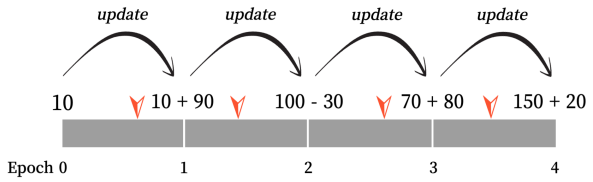
In this section, we describe the system configuration of our framework, including the trust model and our redefined concept of dynamic Proof of Liabilities (PoL). We also outline the underlying data structure used in the framework.

3.1 Trust Model

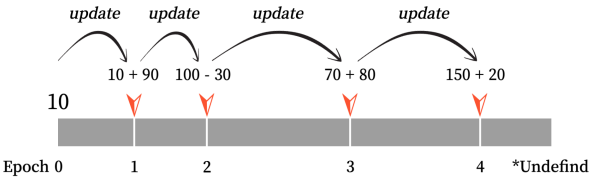
Our CBDC framework is designed with a focus on hybrid model structure. In the hybrid model, the central bank exclusively manages the issuance and redemption of CBDC, while delegating user-facing services and the distribution of CBDC to retail banks. Retail banks then provide a range of services for their customers, including account opening, asset management, deposits, withdrawals, and transfers. To provide these services, banks need access to the personal information. Due to the critical importance of securing user information, banks must be entities certified by the central bank, and they are responsible for fulfilling their obligations to prevent misuse or leakage to other banks. In our framework, delegators function analogously to retail banks in hybrid model.

As certified entities under the central bank certification, delegator manage key escrow with robust security measures and bear legal accountability for any misuse or leakage of users' keys. They process transactions solely based on user-initiated intentions, adhering to regulatory requirements such as AML/CFT to prevent illicit financial activities. This structure mirrors conventional banking procedures while ensuring compliance with modern regulatory standards.

Even though delegators are certified by government institutions, we assume delegators do not inherently trust each other and may behave dishonestly. For instance, a delegator could attempt to transfer less money than intended to a user managed by another delegator.



(a) Dynamic PoL of prior works



(b) Dynamic PoL of Aegis

Figure 1: Overview of dynamic PoL where the red arrow signifies that a transaction has processed.

3.2 Redefining Dynamic Proof of Liabilities

Proof of Liabilities (PoL) is a cryptographic mechanism designed to enable users to verify their liabilities and allow auditors to validate financial liabilities at regular intervals (known as epochs)[1].

Dynamic PoL, as referred by [44], builds on the concept of Proof of Liabilities by enabling the server to prove that updates between consecutive epochs are accurate and consistent with their corresponding proofs. In other words, dynamic PoL maintains continuity by linking updates across epochs. This ensures that validating the current epoch implicitly confirms the correctness of all preceding states, reducing the verification burden and addressing potential vulnerabilities such as window attacks.

We redefine the concept of dynamic PoL by introducing flexibility in the treatment of epochs while preserving the fundamental property of update continuity across epochs. Specifically, this flexibility allows the server, after publishing the proof for the current epoch, to generate the proof for the subsequent epoch either immediately or after a threshold of state changes has been reached. This refinement broadens the applicability of dynamic PoL to scenarios where a fixed epoch is not required.

3.3 Data structure

Entities. Our scheme identifies three key stakeholders: the issuer $\mathcal{I}$, representing the central bank responsible for manufacturing and issuing CBDC; the user $\mathcal{U}$, representing end-users who transact with the CBDC; and the delegator $\mathcal{D}$, representing retail banks that circulate CBDC between $\mathcal{I}$ and $\mathcal{U}$.

Appended-only Ledger. All users have access to the ledger, denoted as $\mathcal{L}$, which contains the data of all blocks. The ledger expands sequentially by appending new transactions to the previous blocks. Specifically, when transaction T' precedes transaction T, $\mathcal{L}_{\mathsf{T}}$ incorporates $\mathcal{L}_{\mathsf{T}'}$, ensuring continuity and order in the ledger.

Values and Commitments. We define three types of secret values, hidden through commitment, v_{cur}: the current balance of the entity's account; Δv: the transaction amount; and v_{new}: the updated balance of the entity's account after the transaction. The commitments corresponding to these values are: $\mathsf{cm}_{\mathsf{cur}}$ (for v_{cur}), $\Delta\mathsf{cm}$ (for Δv), and $\mathsf{cm}_{\mathsf{new}}$ (for v_{new}).

Accounts and Keys. Each entity maintains an account acct and an associated key pair. The account acct is defined as a tuple:

$$\mathsf{acct} := (\mathsf{cm}, \mathsf{addr}, v, o)$$

where cm is commitment representing the account's state, addr is the account address, v denotes the current balance held in the account, and o refers to the opening of the commitment. The key pair consists of a secret key sk and a corresponding public key pk, which are used by entities to generate signatures. The purpose of these signatures varies depending on the entity. For users, the key pair is used to generate a signature sent to the delegator to proving their identity.[2] For delegators, as will be detailed in subsequent sections, the key pair is additionally used to ensure the integrity and authenticity of transaction data.

4 MAIN IDEA

4.1 Batching Technique based on CP-SNARKs

We aim to prove a single batch transaction rather than proving each individual transaction. This section describes the structure of the CP-SNARK used in our approach.

In our design, user assets on the ledger are represented as commitments cm. User actions, such as deposits, withdrawals, and transfers, correspond to updates to these commitments. Thus, a technique for efficiently proving multiple commitments with a single proof is required. To achieve this, we leverage techniques proposed in several studies [2, 13, 22, 26]. We adopt the scheme proposed in [22] to design our system.

In the scheme, the circuit-dependent commitment key ck, generated by the setup algorithm, is divided into two parts, ck_1 and ck_2: ck_1 is used for committing to user information, while ck_2 acts as a bridge, binding the knowledge of these commitments within the SNARK. In other words, the commitments cm stored on the ledger are generated using ck_1. Then, we assume that both the prover and verifier have the same list of commitments denoted by $\mathsf{List}_{\mathsf{cm}}$, representing the multiple commitments to be proven. In the proving algorithm, the prover commits to the messages and their openings using the commitment key ck_2. Upon receiving a challenge ζ from the verifier, the prover computes a witness, agg_v and agg_o, and generates a proof π. agg_v and agg_o represent the aggregate of committed values and their openings, respectively. The verifier, without access to the messages or their openings, verifies the proof by computing the aggregated commitment based on the commitment list ($\mathsf{List}_{\mathsf{cm}}$) and the challenge (i.e., $\mathsf{cm}_{\mathsf{agg}} := \sum_{i=1}^{n} \zeta^i \cdot \mathsf{cm}_i$). We detail the protocol in Figure 2. Specifically, we define the structure of v_i in Figure 2 as $v_{\mathsf{cur}} + \Delta v$ and Δv, where v_{new} is represented by $v_{\mathsf{cur}} + \Delta v$. This structure also applies to o_i and cm_i, and includes

[1] The length of an epoch can vary, typically ranging from one month [9, 32].

[2] A delegator may act as a user in interactions with the central bank.

Protocol

$\mathcal{P}(\mathsf{ek}, \mathsf{List}_{\mathsf{cm}}; \{v_i, o_i\}_{i\in[n]})$ | | $\mathcal{V}(\mathsf{vk}, \vec{\boldsymbol{x}}, \mathsf{List}_{\mathsf{cm}})$

$(\widetilde{\mathsf{cm}}, o) \leftarrow \mathsf{Com}(\mathsf{ck}_2, \{v_i, o_i\}_{i\in[n]})$ $\xrightarrow{\widetilde{\mathsf{cm}}}$

$\xleftarrow{\zeta}$ $\zeta \xleftarrow{\$} \mathbb{Z}_p^*$

$\mathsf{agg}_v = \sum_{i=1}^{n} \zeta^i \cdot v_i$

$\mathsf{agg}_o = \sum_{i=1}^{n} \zeta^i \cdot o_i$

$u := \{\mathsf{agg}_v, \mathsf{agg}_o, \{v_i, o_i\}_{i\in[n]}\}$

$\pi \leftarrow \Pi_{\mathsf{cc}}.\mathsf{Prove}(\mathsf{ek}, (\boldsymbol{x}, \zeta); (u, \omega), o)$ $\xrightarrow{\pi_{\mathsf{cc}}}$ $\mathsf{cm}_{\mathsf{agg}} \leftarrow \sum_{i=1}^{n} \zeta^i \cdot \mathsf{cm}_i$

$\Pi_{\mathsf{cc}}.\mathsf{Verify}(\mathsf{vk}, (\boldsymbol{x}, \zeta), (\mathsf{cm}_{\mathsf{agg}} + \widetilde{\mathsf{cm}}), \pi_{\mathsf{cc}})$

Figure 2: Our used CP-SNARK for multiple Pedersen commitments where $\mathsf{List}_{\mathsf{cm}} := \{\mathsf{cm}_i\}_{i\in[n]}$

both the sender and the receiver. Note that the scheme can become a non-interactive scheme via the Fiat-Shamir transform and can also be expressed in the same format as the CP-SNARK defined in 2.4.1.

The delegator (i.e., the prover) must prove the following for N commitments: $\mathsf{cm}_{\mathsf{cur}}+\Delta\mathsf{cm} = \mathsf{cm}_{\mathsf{new}}$ and additional conditions structured as follows:

- *Non-negative balances*: Expenditures must not exceed available holdings, ensuring that balances remain non-negative after transfer. This is enforced by the following constraint:

$$0 \le v_{\mathsf{new}} = v_{\mathsf{cur}} + \Delta v \le \mathsf{Limit}$$

  where Limit denotes the maximum holding value. Limit is introduced to prevent overflows, as all elements are treated as finite field elements and cannot be negative within the circuit. The holding limit Limit is defined by regulations, ensuring that all balances remain below this limit, which represents an amount that would be impractical to hold in reality. For example, the holding limit may be set to 64 bits.
- *Transfer validity*: The summation of Δv across all trades must equal zero. This follows the equation

$$\sum_{i=1}^{N} \Delta v_i = 0$$

$$\mathsf{agg}_v = \sum_{i=1}^{N} \zeta^i \cdot (v_{\mathsf{cur},i} + \Delta v_i) + \zeta^{N+i} \cdot \Delta v_i$$

$$\mathsf{agg}_o = \sum_{i=1}^{N} \zeta^i \cdot (o_{\mathsf{cur},i} + \Delta o_i) + \zeta^{N+i} \cdot \Delta o_i$$

These conditions are handled within the circuit, while the integrity of the commitments is efficiently proven using the previously described method. The prover sets $\mathsf{List}_{\mathsf{cm}} = \mathsf{List}_{\mathsf{cm}_{\mathsf{new}}} || \mathsf{List}_{\Delta\mathsf{cm}}$, and generates a proof for $\mathsf{List}_{\mathsf{cm}}$. Then, the prover includes the proof and $\mathsf{List}_{\Delta\mathsf{cm}}$ as elements in the transaction. In the verification, since the current commitments $\mathsf{cm}_{\mathsf{cur},i}$ are already pre-published, it can compute $\mathsf{cm}_{\mathsf{new},i}$ from $\mathsf{cm}_{\mathsf{cur},i}$ and $\Delta\mathsf{cm}$ on the ledger. The verifier then uses a random, generated based on the Fiat-Shamir transform, to compute $\mathsf{cm}_{\mathsf{agg}}$ from $\mathsf{List}_{\mathsf{cm}}$, enabling the verifying for multiple commitments.

4.2 Observing on Dynamic PoL

Dynamic PoL enables a party to demonstrate that a user's balance at the current epoch accurately reflects all transactions-such as deposits, withdrawals, and transfers-that occurred between previous and current epochs [44]. As an example, a server stores user-associated transaction records and publishes proofs of this dataset on a ledger. To ensure transparency[3] and accountability, the server must prove the correctness of updates between successive epochs, verifying that all changes to user balances are accurately captured.

In our system, user balances are stored on a smart contract in the form of commitments. Given user requests (e.g., sending CBDC to others), the delegator should correctly update their balances in both the database and the smart contract based on the users' current balances. In other words, the delegator must carefully handle the transaction to maintain consistency between the two repositories.

In detail, Figure 3 illustrates the flow of dynamic PoL within our framework, achieved by applying the proposed batching technique and blockchain.

In this example, the delegator processes requests from two users (marked as yellow and green in the epoch bar). In this structure, a key feature is epoch's flexibility, allowing it to operate without fixed epochs. Transactions can be processed immediately when the batch size threshold is reached or when requests are deemed to have been pending for too long. Initially, the delegator's database stores user liabilities in plain form (i.e., v), and the smart contract stores them as commitments (i.e., cm). Based on the received requests, the delegator computes adjustment commitments (Δcm) for the sender and receiver and generates a batching CP-SNARK

[3]Transparency means ensuring the server operates correctly, not public visibility as in blockchain

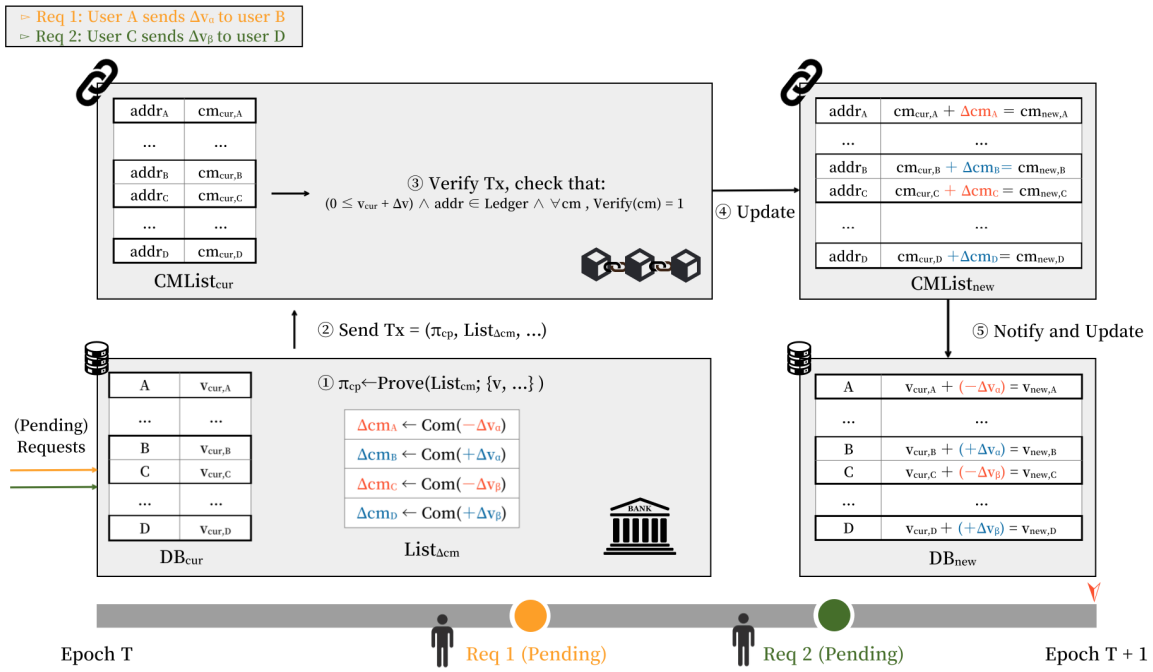


Figure 3: Overview of our dynamic PoL with the batching technique

proof π_{cp}. This proof, along with transaction data, is then submitted to the blockchain. If the proof is verified in the smart contract, it ensures that each user's transactions maintain non-negative balances and transfer validity by batching technique. Additionally, our framework guarantees membership by verifying the proof based on commitments stored in the smart contract. The smart contract then updates the current state ($\mathsf{CMList}_{\mathsf{cur}}$) of commitments to the new state ($\mathsf{CMList}_{\mathsf{new}}$) of commitments. Finally, the delegator's database is updated to synchronize with the confirmed transaction.

Note that proofs for each epoch are stored in an append-only ledger, ensuring public accessibility for users. This design allows the system to satisfy the core requirement of dynamic PoL: proving the correctness of the current state while guaranteeing the integrity of all previous epochs' liabilities. This shows that our system supports dynamic PoL.

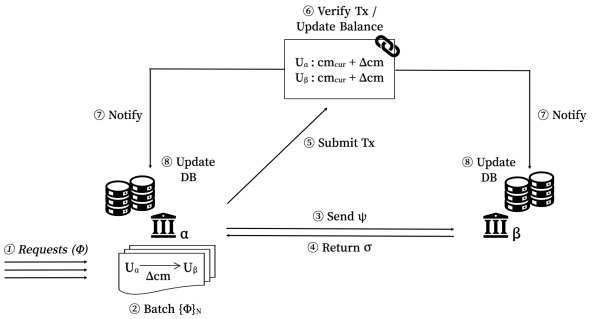


Figure 4: Overview of our framework

5 OUR CONSTRUCTION

Prior to describing our construction, we define additional notation for clarity. We denote a sender and a receiver as snd and rcv, respectively. A bijection p: $\mathcal{S} \to \mathcal{S}$ on the set $\mathcal{S}$ satisfies the properties $p(i) \neq i$ and $p(p(i)) = i$. In other words, p pairs the elements of $\mathcal{S}$ with distinct elements, forming a symmetric and reversible mapping. Since our model requires homomorphic commitments, we use Pedersen commitments.

5.1 System overview

Recall that each delegator can provide users with efficient and scalable batch transfers using the mentioned batching techniques. In this section, we provide an overview of the entire system, emphasizing the interactions between multiple delegators and the blockchain.

For anonymity, we adopt a k-anonymity approach like [11, 41], where k represents the size of the batch. In our model, k-anonymity means guaranteeing unlinkability in proportion to batch size k, thereby making it difficult to identify which sender corresponds to which receiver within a batch. Depicted in Figure 4, the actual account values are stored in the delegators' databases, while the corresponding commitments are recorded in the smart contract. The smart contract does not know the actual account values but instead verifies the validity of transactions for these commitments. The commitments stored in the smart contract represent the CBDC, which holds the actual monetary value.

$\mathsf{Setup}(1^\lambda, \mathcal{R}_\mathcal{D})$

$(\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow \Pi_{\mathsf{cp}}.\mathsf{Setup}(\mathcal{R})$

$\mathsf{pp} := (1^\lambda, \mathsf{ek}, \mathsf{vk}, \mathsf{ck}, \mathsf{ek}_{\mathsf{cp}}, \mathsf{vk}_{\mathsf{cp}})$

return pp

Figure 5: The issuer's algorithm

By utilizing homomorphic commitments, when a transfer occurs across different delegators, the smart contract updates the commitments homomorphically to reflect the changes: the sender's commitment decreases, and the receiver's commitment increases by the same amount. While the smart contract manages these commitment updates, each delegator is responsible for adjusting the corresponding account balances in their local databases to ensure consistency with the smart contract. For the sender's delegator (α), updating the sender's account is straightforward because the transfer amount is known from the user request. However, the receiver's delegator (β) needs additional information to update the receiver's account. Delegator α provides delegator β with the necessary commitment details and corresponding amounts for the batch. Delegator β verifies that the values are non-negative and match the commitments, then signs the list of commitments and returns the signature to delegator α. Once delegator α has collected all required signatures from the involved delegators, the transaction, complete with signatures, is submitted to the smart contract. The transaction is accepted and added to the ledger only if all signatures and the transaction are verified. After approval[4], each delegator updates their databases to reflect the changes, ensuring consistency with the commitment values in the smart contract.

5.2 Issuer algorithms

The main role of the issuer is responsible for setting up the system. This algorithm is denoted as $\mathsf{Setup}(1^\lambda, \mathcal{R}) \rightarrow \mathsf{pp}$. It takes a security parameter and the relation $\mathcal{R}$ as inputs, and outputs public parameters pp.

5.3 Delegator algorithms

A delegator processes transfers on behalf of users based on the intentions received from users. Users freely send transfer intentions to the delegator, defined as follows:

$$\phi = (\sigma^{\mathsf{snd}}, \mathsf{addr}^{\mathsf{snd}}, \mathsf{addr}^{\mathsf{rcv}}, \mathsf{pk}^{\mathsf{rcv}}, \Delta v)$$

where $\mathsf{pk}^{\mathsf{rcv}}$ is denoted by the public key of the delegator to which the receiver $\mathsf{addr}^{\mathsf{rcv}}$ belongs. The term σ^{snd} refers to the signature generated by signing the intention data with the sender's secret key $\mathsf{sk}^{\mathsf{snd}}$. This signature is then used by the delegator to authenticate the identity of the user. We refer to this sub-routine as the delegated transfer request (DTR), which involves a user submitting an intention ϕ to their delegator. When a delegator receives multiple DTR requests, they aggregate them to create a single transaction, which we denote as the delegated batch transfer (DBT). Note that the many DTR received by the delegator are stored in a manner similar to a message queue, which we denote as Pending.

[4]Delegators can monitor *Events* emitted by the smart contract to check the confirmation of transactions.

$\mathsf{IssueKey}(\mathsf{pp})$

$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{pp})$

$\mathsf{addr} \leftarrow \mathsf{CRH}(\mathsf{pk}); \quad o \xleftarrow{\$} \mathbb{Z}_p^*$

$\mathsf{cm} \leftarrow \mathsf{Ped.Com}(\mathsf{ck}, 0; o)$

$\mathsf{acct} := (\mathsf{cm}, \mathsf{addr}, 0, o)$

return $(\mathsf{pk}, \mathsf{sk}, \mathsf{acct})$

$\mathsf{DBT}(\mathsf{pp}, \mathsf{pk}^{\mathsf{snd}}, p, N)$

$\left\{\sigma_{\mathsf{snd}}, \mathsf{addr}_i^{\mathsf{snd}}, \mathsf{addr}_i^{\mathsf{rcv}} \mathsf{pk}_i^{\mathsf{rcv}}, \Delta v_i\right\}_{i \in [N/2]} \leftarrow \mathsf{Pending}(\mathsf{DTR})$

$\left\{\{v_{\mathsf{cur},j}, o_{\mathsf{cur},j}\}_{j \xleftarrow{\$} [N]} \leftarrow \mathsf{DB.Query}(\mathsf{addr}_i^{\mathsf{snd}})\right\}_{i \in [N/2]}$

$\forall j, v_{\mathsf{cur},p(j)} = 0, o_{\mathsf{cur},p(j)} = 0$

$\forall j, \Delta o_j, \Delta o_{p(j)} \xleftarrow{\$} \mathbb{Z}_p^*$

$\forall (i,j), \Delta v_j = -\Delta v_i, \Delta v_{p(j)} = \Delta v_i$

$\forall (i,j), \mathsf{pk}_j = \mathsf{pk}^{\mathsf{snd}}, \mathsf{pk}_{p(j)} = \mathsf{pk}_i^{\mathsf{rcv}}$

$\forall (i,j), \mathsf{addr}_j = \mathsf{addr}^{\mathsf{snd}}, \mathsf{addr}_{p(j)} = \mathsf{addr}_i^{\mathsf{rcv}}$

$v_{\mathsf{new},j} = v_{\mathsf{cur},j} - \Delta v_j, o_{\mathsf{new},j} = o_{\mathsf{cur},j} + \Delta o_j$

$v_{\mathsf{new},p(j)} = v_{\mathsf{cur},p(j)} + \Delta v_{p(j)}, o_{\mathsf{new},p(j)} = o_{\mathsf{cur},p(j)} + \Delta o_{p(j)}$

$\{\Delta \mathsf{cm}_k := \mathsf{Ped.Com}(\mathsf{ck}, \Delta v_k; \Delta o_k)\}_{k \in [N]}$

$\left\{\mathsf{cm}_{\mathsf{new},k} := \mathsf{Ped.Com}(\mathsf{ck}, v_{\mathsf{new},k}; o_{\mathsf{new},k})\right\}_{k \in [N]}$

$\vec{\boldsymbol{x}} := \left\{\{\mathsf{cm}_{\mathsf{new},k}\}_{k \in [N]}, \{\Delta \mathsf{cm}_k\}_{k \in [N]}\right\}$

$\vec{\boldsymbol{w}} := \left\{\{(v_{\mathsf{cur},k}, o_{\mathsf{cur},k})\}_{k \in [N]}, \{(\Delta v_k, \Delta o_k)\}_{k \in [N]}\right\}$

for $k \in \{1, \ldots, N\}$ **do**

  $\mathsf{List}_{\mathsf{pk}}.\mathsf{append}(\mathsf{pk}_k)$

  $\mathsf{List}_{\mathsf{addr}}.\mathsf{append}(\mathsf{addr}_k)$

  $\mathsf{List}_{\Delta\mathsf{cm}}.\mathsf{append}(\Delta \mathsf{cm}_k)$

endfor

$\mathsf{List}_{\mathsf{data}} := (\mathsf{List}_{\mathsf{pk}}, \mathsf{List}_{\mathsf{addr}}, \mathsf{List}_{\Delta\mathsf{cm}})$

$\pi_{\mathsf{cp}} \leftarrow \Pi_{\mathsf{cp}}.\mathsf{Prove}(\mathsf{ek}, \vec{\boldsymbol{x}}; \vec{\boldsymbol{w}})$

$\psi_{\mathcal{D}_i} := (\pi_{\mathsf{cp}}, \mathsf{List}_{\mathsf{data}}, \{\Delta v, \Delta o\}_{\forall \mathsf{addr} \in [\mathcal{D}_i.\mathsf{addr}]})$

$\mathsf{List}_\sigma \leftarrow \left[\mathsf{GenSig}(\mathsf{pp}, \psi_{\mathcal{D}_i})\right]$

return $\mathsf{Tx}_{\mathsf{DBT}} := (\mathsf{List}_\sigma, \pi_{\mathsf{cp}}, \mathsf{List}_{\mathsf{data}})$

$\mathsf{GenSig}(\mathsf{pp}, \psi)$

parse $\psi := (\pi_{\mathsf{cp}}, \mathsf{List}_{\mathsf{data}}, \{\Delta v_j, \Delta o_j\}_{j \in \mathcal{D}_i})$

parse $\mathsf{List}_{\mathsf{data}} := (\mathsf{List}_{\mathsf{pk}}, \mathsf{List}_{\mathsf{addr}}, \mathsf{List}_{\Delta\mathsf{cm}})$

for $(j, \mathsf{addr}) \in \mathcal{D}_i$ **do**

  assert $\Delta v_j \geq 0$

  assert $\mathsf{Ped.VerCom}(\mathsf{ck}, \mathsf{List}_{\Delta\mathsf{cm}}[j], \Delta v_j, \Delta o_j) = \mathsf{true}$

endfor

$\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, \pi_{\mathsf{cp}}, \mathsf{List}_{\mathsf{data}})$

return σ

Figure 6: The delegator's algorithms

Another sub-algorithm, denoted as GenSig, is defined. Delegators cannot access information about users assigned to other delegators and do not inherently trust one another. However, when transferring CBDC to users under a different delegator, transaction integrity must still be guaranteed. This requires communication between delegators and mutual verification to ensure the correctness of the updated values. To achieve this, we use digital signature to validate the transactions. The sending delegator sends disjoint data ψ to each delegator, including partial transactions and user-specific information like Δv and the receiver's address. Each delegator verifies this data, signs it, and returns the signature σ.

Define the main algorithm, Delegated Batch Transfer, denoted as DBT. The DBT algorithm aggregates multiple intentions received by the delegator into a single transaction. For a batch size of N, the algorithm retrieves intentions of $N/2$ users from the Pending. Each intention includes the sender's address, $\mathsf{addr}^{\mathsf{snd}}$. The delegator then retrieves the current balance v_{cur}, and its corresponding opening, o_{cur}, which are associated with $\mathsf{addr}^{\mathsf{snd}}$ from the database. Each v_{cur} and o_{cur} is then assigned a unique random index j, ranging from 1 to N, in order to make it difficult to distinguish whether it is the sender or the receiver. Consequently, $N/2$ pairs of the current balance $v_{\mathsf{cur},j}$ for sender j and its corresponding opening $o_{\mathsf{cur},j}$ are generated.

After preparing $N/2$ pairs of $v_{\mathsf{cur},j}$ and $o_{\mathsf{cur},j}$ for the senders, the algorithm also prepares v_{cur} and o_{cur} for $N/2$ receivers in the same way. Since the sending delegator cannot access the v_{cur} and o_{cur} of users belonging to other delegators, the receiver's v_{cur} and o_{cur} are set to 0. Receivers, like senders, are also assigned an index j. At this point, a bijection p is used to map each sender j to a receiver $p(j)$.

Next, the algorithm computes the updated balance v_{new} for both the sender and the receiver using the transaction amount Δv. For the sender, the updated balance is computed as $v_{\mathsf{new}} = v_{\mathsf{cur}} - \Delta v$. Similarly, the opening for the updated balance, o_{new} is updated as $o_{\mathsf{new}} = o_{\mathsf{cur}} + \Delta o$. In contrast, for users belonging to other delegators, whose v_{cur} and o_{cur} are not accessible, the values are set as $v_{\mathsf{new}} = \Delta v$ and $o_{\mathsf{new}} = \Delta o$.

Then, the commitments $\mathsf{cm}_{\mathsf{new}}$ for the updated balance v_{new} and the commitments $\Delta\mathsf{cm}$ for the transaction amount Δv are computed. These commitments are included in the statement $\vec{x}$, while the corresponding $v_{\mathsf{new}}, o_{\mathsf{new}}, \Delta v$ and Δo are included in the witness $\vec{w}$. Using these, a proof $\mathbf{\Pi}_{\mathsf{cp}}$ is generated to satisfy the non-negative and transfer validity conditions described in Section 4.1.

However, using only $\mathbf{\Pi}_{\mathsf{cp}}$, the delegators participating in the trade cannot verify the transaction amount Δv. For example, a malicious sender could attempt to create a transaction where they gain $+\Delta v$ while the receiver incurs $-\Delta v$, thereby compromising the receiver's account. To prevent such attacks, it is crucial to mutually verify that the transaction information $\Delta\mathsf{cm}$ received by the receiving delegator has been correctly derived from the Δv sent by the sending delegator. To achieve this, the GenSig sub-algorithm is employed.

GenSig takes as input the delegator-specific data Δv and its corresponding Δo, as well as the transaction data, including the proof $\mathbf{\Pi}_{\mathsf{cp}}$ and $\mathsf{List}_{\mathsf{data}}$. It performs two key validations: first, it verifies that the transaction amount Δv is non-negative; second, it ensures that the commitment of Δv and Δo matches $\mathsf{List}_{\Delta\mathsf{cm}}$, which is included in $\mathsf{List}_{\mathsf{data}}$. Upon successful verification, GenSig signs the transaction data and returns the signature to the sending delegator. Finally, the DBT algorithm outputs, as the transaction, consisting of the list of signatures List_{σ} for the delegators participating in the trade, the proof $\mathbf{\Pi}_{\mathsf{cp}}$, and $\mathsf{List}_{\mathsf{data}}$.

In summary, the delegator's algorithm is described as follows:

- $\mathsf{IssueKey}(\mathsf{pp}) \rightarrow (\mathsf{pk}, \mathsf{sk}, \mathsf{acct})$: It opens a delegated account acct along with a key pair (pk, sk).
- $\mathsf{GenSig}(\mathsf{pp}, \psi) \rightarrow \sigma$: It takes the public parameter pp and ψ as inputs, and returns a signature σ.
- $\mathsf{DBT}(\mathsf{pp}, \mathsf{pk}^{\mathsf{snd}}, p, N) \rightarrow \mathsf{Tx}_{\mathsf{DBT}}$: It is used to process multiple DTR requests received from users by the delegator, and outputs a transaction $\mathsf{Tx}_{\mathsf{DBT}}$.

Remark. Our system can support interoperability with privacy-preserving transfer systems using self-managed accounts, as proposed in [20, 23, 37, 38]. These systems generally utilize commitments and consist of functionalities such as Send and Receive. To combine with such systems, our system can support interactions between self-managed accounts and delegated accounts, enabling value exchange between the two types of accounts.

For these exchanges to be valid, the following must hold:

- The transfer amount and the updated balance after the transfer must both be positive.
- The new commitment, reflecting the transfer amount added to the previous balance, must be valid.
- Both the self-managed account and the delegated account must have valid key pairs.

Since the proof is generated by the owner of the self-managed account, the delegator cannot know the transfer amount. Therefore, they must inform the delegator of the transfer amount through a secure channel to allow the delegator to update their database.

5.4 Smart Contract

In our system, various smart contract functions are defined. To indicate the usability for each entity, subscripts are used. These functions output either 1 or 0, representing success and failure, respectively.[5] Note that msg.sender means the address of the entity that called the function. Additionally, CMList represents the on-chain storage where users' balances are stored as commitments. It is structured as a mapping that associates each address (key) with its corresponding commitment. The detailed smart contracts are described in the Figure 7.

- $\mathsf{Deploy}_{\mathcal{I}}(\mathsf{pp}, \Delta v)$: This algorithm deploys a smart contract and stores the keys.[6] Additionally, it mints the initial supply of CBDC, issuing the CBDC into circulation.
- $\mathsf{GrantRole}_{\mathcal{I}}(\mathsf{pk})$: This algorithm grants the validator role to a specified delegator (with the public key pk). This role signifies that it has the authority to perform GenSig.
- $\mathsf{Mint}_{\mathcal{I}}(\Delta v)$: This algorithm enables $\mathcal{I}$ to issue additional CBDC. Thus, Δv is added to the existing commitment of $\mathcal{I}$'s account.

[5] If any inputs are of incorrect type, the function automatically fails.

[6] While the verification key vk in pp must be required, but the evaluation key ek is optional and does not need to be included in the smart contract; it can be shared through other channels.

Deploy(pp, v)

Store vk, ck
$\mathcal{I} \leftarrow$ msg.sender
Mint(v)/ issue the initial supply of CBDC
Initialize CMList

GrantRole(pk)

assert msg.sender = $\mathcal{I}$
isValid[pk] $\leftarrow$ true

Mint(Δv)

assert msg.sender = $\mathcal{I}$
$\mathsf{CMList}[\mathcal{I}] = \mathsf{CMList}[\mathcal{I}] + g^{\Delta v}$
/ g is the element of ck

DBT$(\mathsf{Tx}_{\mathsf{DBT}})$

assert isValid[msg.sender] = true
parse $\mathsf{Tx}_{\mathsf{DBT}} := (\mathsf{List}_{\sigma}, \pi_{\mathsf{cp}}, \mathsf{List}_{\mathsf{data}})$
for $i \in \{1, N\}$
  $\mathsf{List}_{\mathsf{cm}}[i] = \mathsf{CMList}[\mathsf{addr}_i] + \mathsf{List}_{\Delta\mathsf{cm}}[i]$
  $\mathsf{List}_{\mathsf{cm}}[i+N] = \mathsf{List}_{\Delta\mathsf{cm}}[i]$
endfor
assert $\Pi_{\mathsf{cp}}.\mathsf{Verify}(\mathsf{vk}, \mathsf{List}_{\mathsf{cm}}, \pi_{\mathsf{cp}}) = \mathsf{true}$
assert $\forall i \in \mathsf{List}_{\mathsf{pk}} : \mathsf{isValid}[\mathsf{pk}_i] = \mathsf{true} \wedge \mathsf{Sig.Verify}(\mathsf{pk}_i, \sigma_i) = \mathsf{true}$
for $i \in \{1, \ldots, N\}$
  $\mathsf{CMList}[\mathsf{addr}_i] = \mathsf{CMList}[\mathsf{addr}_i] + \mathsf{List}_{\mathsf{cm}}[i+N]$
endfor

Figure 7: Aegis's smart contract

- $\mathsf{DBT}_{\mathcal{D}}(\mathsf{Tx}_{\mathsf{DBT}})$: This algorithm takes $\mathsf{Tx}_{\mathsf{DBT}}$ as input and allows the delegator to update the balance commitments of each delegated account. The algorithm proceeds through the following steps: (1) check the validator's authority; (2) make a statement using the current user commitments CMList stored on-chain and the commitment of the changes $\mathsf{List}_{\Delta\mathsf{cm}}$ included in the transaction; (3) verify the proof π_{cp}; (4) verify the signatures of the delegators involved in the transaction; and (5) update the users' balances accordingly.

Remark. Delegators must check that their private storage is updated correctly only after confirming that DBT has been properly executed and reflected in the ledger (e.g., when the status equals 1 or an event is emitted). Incorrect updates may result in mismatches between the values stored in the private storage and those in the ledger, causing verification failures.

Additionally, the described approach assumes a single smart contract where the sender bank updates the receiver bank's data only for verified transactions. This can be extended by allowing each bank to deploy its own smart contract, which can internally call the smart contracts of other banks to update user balances. Furthermore, if the sender bank records a transaction on the ledger without the receiver bank's involvement, it may result in discrepancies with the receiver bank's data, potentially leading the receiver bank to file a formal complaint or dispute.

6 SECURITY

To ensure the robustness of our privacy-preserving CBDC system, we show that it satisfies the completeness and security requirements.

Completeness. This property guarantees that the system will function correctly and transactions will always succeed when all participating entities act honestly and generate transactions according to protocol rules.

Security requirements. Our system satisfies two critical security properties: update soundness and privacy. Update soundness ensures that the sum of all transaction changes equals zero while also guaranteeing that no user spends more money than they are legitimately entitled to. This prevents any unauthorized increase or decrease in transaction amounts at both the system-wide and individual levels. Privacy ensures that no additional information about the transfers of honest parties, beyond what is intended, is revealed to an adversary. This is formalized through three key aspects: hiding transferred amounts, protecting the identities of transaction participants, and preventing the linkability of different transactions. These security properties are designed to uphold the system's integrity and confidentiality while preserving its functionality.

6.1 Security proofs

We prioritize the discussion of security requirements and omit the detailed proof for completeness, as it is relatively straightforward.

6.1.1 Security requirements. Before describing security games, we define the common elements of security games. In the setup phase of the game, entities $(\mathcal{C}, \mathcal{A}, \mathcal{O}_{\mathsf{SC}})$ take the security parameter λ as input. The oracle, $\mathcal{O}_{\mathsf{SC}}$, adds transactions submitted by a challenger or an adversary as pending.[7] The challenger can query $\mathcal{O}_{\mathsf{SC}}$ at any time to obtain the current state or any prior state of SC. Meanwhile, an adversary $\mathcal{A}$ has full visibility of the oracle, including all transactions sent by the challenger, the state transitions of SC, and the complete transaction history, etc.

Adversary's behavior. The adversary $\mathcal{A}$ can influence the state of SC as follows.

- Indirect control via challenger ($\mathcal{C}$): $\mathcal{A}$ can instruct $\mathcal{C}$ to execute a delegator algorithm with specific inputs and send the resulting transaction to $\mathcal{O}_{\mathsf{SC}}$.
  (1) For IssueKey, $\mathcal{C}$ sends the public information related to the generated accounts to $\mathcal{A}$, excluding any secret information such as private keys and openings.
  (2) For DBT, $\mathcal{A}$ specifies the sender, recipient, and the transfer amount to $\mathcal{C}$.
  (3) When $\mathcal{C}$ receives an instruction, it utilizes the state of SC from the last block of the previous epoch to execute the requested operation.
  (4) If $\mathcal{C}$ has already initiated a transfer in the current epoch using a particular public key, any further instructions to generate the same transaction from that key within the same epoch will be rejected. Additionally, $\mathcal{C}$ will reject

[7] In fact, each delegator is authorized by the central bank to act as a validator and publish transactions after undergoing strict qualification test. However, we aim to demonstrate that the system remains secure even if this requirement is disregarded.

instructions to transfer involving overlapping users within the anonymity set.
- Direct submission by $\mathcal{A}$: $\mathcal{A}$ can directly publish arbitrary DBT transactions to O_{SC} (as previously mentioned, we omit the role verification for EOAs in the smart contract).

Let ADDR be the set of addresses generated by C at $\mathcal{A}$'s request. However, $\mathcal{A}$ does not possess the secret information corresponding to these addresses.

Update soundness. We formally define a game, denoted as $\mathsf{Game}_{\mathsf{US}}$, involving a challenger C, an adversary $\mathcal{A}$, and the oracle O_{SC}. First, let addr represent the addresses belonging to users in ADDR (addr $\in$ ADDR), and addr$'$ denote the addresses not in ADDR (i.e., those not generated through queries to C, addr$' \notin$ ADDR). We define the notations related to amounts in the game from the perspective of the adversary as follows:

- α: The total deposits associated with addr$'$.
- β: The total amount received by the adversary through transfers, which can be broken down into the following subcomponents:
  - β_a: The total deposits made by users in addr.
  - β_b: The total balance of the smart contract held by users in addr.
  - β_c: The total amount withdrawn by users in addr.

  Consequently, $\beta = \beta_a - \beta_b - \beta_c$.
- γ: The total withdrawals associated with addr$'$.

We define the system to satisfy update soundness if, for all PPT adversaries $\mathcal{A}$, the probability that $\alpha + \beta < \gamma$ is negligible.

Proof. For transfer (i.e., DBT), let $\mathsf{List}_{\mathsf{addr}} := (\mathsf{addr}_1, \mathsf{addr}_2, \ldots)$ represent the anonymity set, $\mathsf{List}_{\Delta\mathsf{cm}} := (\Delta\mathsf{cm}_1, \Delta\mathsf{cm}_2, \ldots, \Delta\mathsf{cm}_n)$ the transfer commitments, and π_{cp} the proof for the DBT transaction Tx. Also, let $(v_{\mathsf{cur},1}, v_{\mathsf{cur},2}, \ldots, v_{\mathsf{cur},n})$ denote the current holdings of users included in the DBT transaction, and let $(v_{\mathsf{new},1}, v_{\mathsf{new},2}, \ldots, v_{\mathsf{new},n})$ denote their updated holdings after the transaction. To define the transfer results concisely, we introduce a bijection $p : \mathcal{S} \to \mathcal{S}$ over the set $\mathcal{S} = \{1, \ldots, n\}$ such that $p(i) \neq i, p(p(i)) = i$, and $n/2$ disjoint pairs exists: $\{(i, j) | p(i) = j, p(j) = i, i \neq j\}$. For the honest transfer, there are exactly $n/2$ pairs of the form $(\Delta v_{p(i)}, -\Delta v_{p(j)})$. If there exists any pair (i, j) such that $\Delta v_{p(i)} \neq -\Delta v_{p(j)}$, it implies that the sender and receiver are not transferring equal but opposite values. This imbalance allows the adversary to win the $\mathsf{Game}_{\mathsf{US}}$ where $\alpha + \beta < \gamma$. However, such a scenario contradicts the knowledge soundness of the zk-SNARKs and binding property of the commitment scheme. Hence, the probability is at most negligible.

Privacy. We formally define a game, denoted as $\mathsf{Game}_{\mathsf{PV}}$, involving a challenger C, an adversary $\mathcal{A}$, and the oracle O_{SC}.[8] In the game, the adversary $\mathcal{A}$ provides two consistent instructions to the challenger C, rather than a single instruction. The challenger executes the $(b+1)$-th instruction based on a secret random bit b. At the end of the game, the adversary outputs a bit b' as its guess for b. Two instructions are considered consistent if they satisfy the following conditions for a transfer transaction:

(1) They originate from the same EOA.
(2) The anonymity set must be identical across both instructions (including the involved delegator)
(3) If the receiver is controlled by the adversary, both instructions must specify the same address and the same amount.
(4) The transferred amounts for all users must be valid.

The system is private if, for all PPT adversaries $\mathcal{A}$, the probability that $b = b'$ is at most $1/2 + \mathsf{negl}(\lambda)$.

Proof. For two consistent transfer transactions, they differ in List_σ, $\mathsf{List}_{\Delta\mathsf{cm}}$, and π_{cp}, Firstly, the adversary gains no advantage through List_σ. The transferred commitments either commit to zero or valid transferred amounts, which are indistinguishable to the adversary due to the hiding property of the commitments. Similarly, the proof is indistinguishable due to zero-knowledge property of the zk-SNARK. Consequently, the adversary's advantage in distinguishing between the two consistent transactions is negligible.

[8]The game adopts a left-right setting commonly utilized in indistinguishability-based definitions.

Table 1: Performance of our system

(a) zk-SNARK Performance by Batch Size

size (log)	Setup (s)	Prove (s)	Verify (ms)	Constraints (k)	ek (KB)	vk (B)
3	0.048	0.031	1.264	4.1	925	296
4	0.069	0.054	1.313	8.2	1,850	
5	0.128	0.099	1.336	16.4	3,698	
6	0.245	0.191	1.525	32.9	7,397	
7	0.472	0.316	1.821	65.9	14,793	
8	0.896	0.618	2.040	131.8	29,586	
9	1.781	1.150	2.900	263.6	59,171	
10	3.469	2.217	4.245	527.3	118,342	
11	7.046	4.557	6.646	1,054.7	236,684	
12	14.169	9.571	11.796	2,109.4	473,367	

(b) Smart Contract Performance by Batch Size

size(log)	Time(s)	TPS	Gas(M)
3	0.015	64.19	0.62
4	0.021	46.62	0.90
5	0.028	35.08	1.46
6	0.047	21.41	2.59
7	0.078	12.83	4.87
8	0.152	6.57	9.46
9	0.296	3.37	18.83
10	0.626	1.59	38.26
11	1.512	0.66	79.94
12	3.102	0.33	174.48

7 EXPERIMENT

Our implementation is based on Rust (Arkworks) and TypeScript, using the BN254 elliptic curve for cryptographic operations. Blockchain experiments were conducted on the Ethereum Hardhat testnet, and the smart contracts were developed in Solidity. To reduce gas costs, the SHA3 hash function was applied. All experiments were executed on a MacBook Pro 16 equipped with an Apple M1 CPU and 32GB of RAM.

7.1 Evaluations

In this section, we first clarify two key aspects of our batched transaction approach before delving into detailed performance results. First, since our construction includes both senders and receivers in each batch, the number of transactions is half the batch size. Thus, if a batch size is 1024, the system handles 512 distinct transactions. Second, we represent the batch size in log scale for convenience. For example, size = 10 corresponds to a batch size of $2^{10} = 1024$.

7.1.1 Performance of zk-SNARK. We evaluate our system's performance for batched proofs, as shown in Table 1a. The experiment was conducted with the number of threads fixed at 10. The setup time ranges from 0.048 seconds at size = 3 to 14.169 seconds at size = 12. The proving phase experiences an increase from 0.031 to 9.571 seconds over that same range, and verification time grows from 1.264 ms to 11.796ms. Notably, while the number of constraints and the proving key (ek) size expand with the batch size, the verification key (vk) remains constant at 296 bytes. This stability in vk size means that, regardless of how large the batch becomes, the overhead for storing and deploying the vk on the smart contract side does not scale. Despite this increase in resource demand, our framework effectively accommodates higher workloads, indicating that it is well-suited for real-world use cases where processing a large volume of transactions in parallel is essential.

7.1.2 Performance of smart contract. From Table 1b, we observe that as the batch size increases, both the verification time and gas consumption rise. The verification time grows from 0.015 seconds to 3.102 seconds, and gas consumption increases from 0.62 million to 174.48 million. Since executing $\mathbf{\Pi}_{cc}.\mathsf{Verify}(\cdot)$ involves calculating cm_{agg}, the computational overhead increases with larger batch sizes, as shown in Figure 2.

7.1.3 Comparisons. Table 2 compares our system (**Aegis**) with two existing solutions, zkLedger [30] and Solidus [15], on two dimensions: transaction scalability (Table 2a) and threading performance (Table 2b). In terms of total processing time for different numbers of transactions, our system consistently outperforms the other two schemes, demonstrating lower latency as the number of transactions increases from 4 to 2,048. Specifically, **Aegis** shows only a marginal increase in total processing time even when the transaction count scales exponentially, while both zkLedger and Solidus exhibit substantially higher growth rates.

Turning to the effects of multi-threading, Table 2b indicates that increasing the number of threads dramatically reduces processing times for all three schemes. Specifically, when the number of threads increases from 1 to 10, the execution time of **Aegis** decreases from 9.88 seconds to 2.83 seconds, while zkLedger reduces from 29.81 to 9.72, and Solidus experiences the most significant drop, from 802.92 to 109.22.

Lastly, we compare the performance of **Aegis** with other Proof of Liabilities (PoL) approaches, as shown in Table 2c. This experiment was conducted with 1024 users. The comparison includes SNARKedMT, Notus [44], TAP [35] and **Aegis**. The SNARKedMT is a simplified version of Binance's PoL approach [10], using the implementation from Notus and evaluated with a Merkle tree of depth 28. Experimentally, SNARKedMT exhibits a proving time of 35.47 seconds and a verification time of 0.001 seconds, making its proving time significantly longer than other approaches. Moreover, SNARKedMT only supports static PoL and does not support dynamic PoL, which makes it unsuitable for systems where PoL needs to be performed frequently. Next, we evaluate the performance of Notus, TAP and our proposed system, **Aegis**, all of which support dynamic PoL. Notus demonstrates a proving time of 6.399 seconds and a verification time of 0.001 seconds. TAP achieves the fastest proving time of 1.11 seconds and a verification time of 0.004 seconds, showing its efficiency in proof generation. Our proposed system, **Aegis**, shows a proving time of 4.56 seconds, the seconds fastest among the schemes, and a verification time of 0.006 seconds. While **Aegis** provides k-anonymity corresponding to the batch size instead of the full anonymity offered by other schemes, it offers a unique advantage by integrating PoL directly into transaction processing. This eliminates the need for a separate PoL algorithm, thereby simplifying the process and enhancing system efficiency.

Table 2: Comparison between our work and existing works

(a) Transaction Comparison(10 threads)

Tx \ Name	Total Time (s)		
	Aegis	zkLedger	Solidus
4	0.05	0.09	0.89
8	0.08	0.13	1.77
16	0.13	0.36	3.53
32	0.24	0.63	6.95
64	0.39	1.22	14.02
128	0.77	2.50	27.89
256	1.45	4.93	55.57
512	2.84	9.73	111.29
1024	6.07	19.42	221.31
2048	12.67	38.71	449.64

(b) Thread Comparison(512 tx)

Thread \ Name	Total Time (s)		
	Aegis	zkLedger	Solidus
1	9.88	29.81	802.92
2	6.03	17.07	403.01
4	4.12	10.85	212.23
6	3.25	10.11	148.44
8	3.12	9.79	119.14
10	2.83	9.72	109.22

(c) PoL Comparison(1024 users)

	Prove(s)	Verify(s)	Dynamic PoL
SNARKedMT	35.47	0.001	✗
Notus	6.399	0.001	✓
TAP	1.11	0.004	✓
Aegis	4.56	0.006	✓

REFERENCES

[1] 2020. The Riksbank's e-krona pilot. Technical Report Reg. no 2019-00291. Sveriges Riksbank. https://www.riksbank.se/globalassets/media/rapporter/e-krona/2019/the-riksbanks-e-krona-pilot.pdf

[2] Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. 2021. ECLIPSE: Enhanced Compiling method for Pedersen-committed zkSNARK Engines. Cryptology ePrint Archive, Paper 2021/934. https://eprint.iacr.org/2021/934 https://eprint.iacr.org/2021/934.

[3] Raphael Auer and Rainer Böhme. 2020. The technology of retail central bank digital currency. Technical Report March 2020. 1–16 pages. https://www.bis.org/publ/qtrpdf/r_qt2003j.pdf

[4] Raphael Auer and Rainer Böhme. 2021. Central Bank Digital Currency: The Quest for Minimally Invasive Technology. BIS Working Papers 948. Bank for International Settlements. https://www.bis.org/publ/work948.htm

[5] Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. 2015. Anonymous Transferable E-Cash. In Proceedings of the Public Key Cryptography Conference. 14, 15.

[6] Bank for International Settlements. 2020. Central bank digital currencies: foundational principles and core features. https://www.bis.org/publ/othp33.pdf.

[7] Bank of England. 2020. Central Bank Digital Currency: Opportunities, challenges and design. https://www.bankofengland.co.uk/-/media/boe/files/paper/2020/central-bank-digital-currency-opportunities-challenges-and-design.pdf.

[8] Bank of Japan. 2024. Central Bank Digital Currency Experiments: Progress on the Pilot Program (April 2024). https://www.boj.or.jp/en/paym/digital/index.htm Retrieved from the Bank of Japan website.

[9] Binance. [n. d.]. Binance proof of reserves. https://www.binance.com/en/proof-of-reserves/

[10] Binance. 2024. zkmerkle-proof-of-solvency. https://github.com/binance/zkmerkle-proof-of-solvency

[11] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh. 2019. Zether: Towards Privacy in a Smart Contract World. IACR Cryptology ePrint Archive (2019).

[12] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. 2006. Balancing Accountability and Privacy Using E-Cash. In Proceedings of the International Conference on Security and Cryptography for Networks. 2, 14.

[13] Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. Cryptology ePrint Archive, Report 2019/142. https://eprint.iacr.org/2019/142.

[14] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. 2017. Solidus: Confidential Distributed Ledger Transactions via PVORM. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (Dallas, Texas, USA) (CCS '17). Association for Computing Machinery, New York, NY, USA, 701–717. https://doi.org/10.1145/3133956.3134010

[15] Ethan Cecchetti, Fan Zhang, Yan Ji, Ahmed Kosba, Ari Juels, and Elaine Shi. 2017. Solidus: Confidential distributed ledger transactions via PVORM. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 701–717.

[16] David Chaum. 1983. Blind signatures for untraceable payments. In Advances in Cryptology: Proceedings of Crypto 82. Springer, 199–203.

[17] Dong Beom Choi, Paul Goldsmith-Pinkham, and Tanju Yorulmazer. 2023. Contagion Effects of the Silicon Valley Bank Run. Technical Report Working Paper 31772. National Bureau of Economic Research. http://www.nber.org/papers/w31772

[18] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. 2015. Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (Denver, Colorado, USA) (CCS '15). Association for Computing Machinery, New York, NY, USA, 720–731. https://doi.org/10.1145/2810103.2813674

[19] Terry L. Gibson. 2023. Material Loss Review of First Republic Bank. https://www.fdicoig.gov/reports-publications/bank-failures/material-loss-review-first-republic-bank Report No. EVAL-24-03.

[20] Zhangshuang Guan, Zhiguo Wan, Yang Yang, Yan Zhou, and Butian Huang. 2022. BlockMaze: An Efficient Privacy-Preserving Account-Model Blockchain Based on zk-SNARKs. IEEE Transactions on Dependable and Secure Computing 19, 3 (2022), 1446–1463. https://doi.org/10.1109/TDSC.2020.3025129

[21] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, and Raluca Ada Popa. 2021. Merkle^2: A Low-Latency Transparency Log System. In 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. IEEE, 285–303. https://doi.org/10.1109/SP40001.2021.00088

[22] Byeongjun Jang, Gweonho Jeong, Hyuktae Kwon, Hyunok Oh, and Jihye Kim. 2024. Lego-DLC: batching module for commit-carrying SNARK under Pedersen Engines. Cryptology ePrint Archive, Paper 2024/1405. https://eprint.iacr.org/2024/1405

[23] Gweonho Jeong, Nuri Lee, Jihye Kim, and Hyunok Oh. 2023. Azeroth: Auditable Zero-Knowledge Transactions in Smart Contracts. IEEE Access 11 (2023), 56463–56480. https://doi.org/10.1109/ACCESS.2023.3279408

[24] Yan Ji and Konstantinos Chalkias. 2021. Generalized Proof of Liabilities. In Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (Virtual Event, Republic of Korea) (CCS '21). Association for Computing Machinery, New York, NY, USA, 3465–3486. https://doi.org/10.1145/3460120.3484802

[25] Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. 2022. PEReDi: Privacy-Enhanced, Regulated and Distributed Central Bank Digital Currencies. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (Los Angeles, CA, USA) (CCS '22). Association for Computing Machinery, New York, NY, USA, 1739–1752. https://doi.org/10.1145/3548606.3560707

[26] Jiwon Lee, Jaekyoung Choi, Jihye Kim, and Hyunok Oh. 2019. SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization. IACR Cryptol. ePrint Arch. (2019), 1270. https://eprint.iacr.org/2019/1270

[27] Stefano Leucci, Massimo Attoresi, and Xabier Lareo. 2023. TechDispatch: Central Bank Digital Currency (CBDC). Technical Report. European Data Protection Supervisor (EDPS). https://www.edps.europa.eu/system/files/2023-03/23-03-29_techdispatch_cbdc_en.pdf

[28] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. 2015. Coniks: Bringing key transparency to end users. In Proceedings of the 24th USENIX Security Symposium (Proceedings of the 24th USENIX Security Symposium). USENIX Association, 383–398. Publisher Copyright: © 2015 Proceedings of the 24th USENIX Security Symposium. All rights reserved.; 24th USENIX Security Symposium ; Conference date: 12-08-2015 Through 14-08-2015.

[29] Neha Narula, Willy Vasquez, and Madars Virza. 2018. zkLedger: Privacy-Preserving Auditing for Distributed Ledgers. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18). USENIX Association, Renton, WA, 65–80. https://www.usenix.org/conference/nsdi18/presentation/narula

[30] Neha Narula, Willy Vasquez, and Madars Virza. 2018. {zkLedger}:{Privacy-Preserving} auditing for distributed ledgers. In 15th USENIX symposium on networked systems design and implementation (NSDI 18). 65–80.

[31] Office of Science and Technology Policy (OSTP). 2022. Technical Evaluation for a U.S. Central Bank Digital Currency System. Retrieved from http://www.whitehouse.gov/ostp.

[32] OKX. 2023. Proof of Reserves. https://www.okx.com/learn/proof-of-reserves

[33] BIS Consultative Group on Innovation and the Digital Economy (CGIDE). 2024. A proposal for a retail central bank digital currency (CBDC) architecture. Technical Report December 2024. Bank for International Settlements. 1–22 pages. https://www.bis.org/publ/othp89.htm

[34] Yanqing Peng, Min Du, Feifei Li, Raymond Cheng, and Dawn Xiaodong Song. 2020. FalconDB: Blockchain-based Collaborative Database. Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (2020). https://api.semanticscholar.org/CorpusID:218982030

[35] Daniël Reijsbergen, Aung Maw, Zheng Yang, Tien Tuan Anh Dinh, and Jianying Zhou. 2023. TAP: Transparent and Privacy-Preserving Data Services. In 32nd USENIX Security Symposium (USENIX Security 23). USENIX Association, Anaheim, CA, 6489–6506. https://www.usenix.org/conference/usenixsecurity23/presentation/reijsbergen

[36] Sveriges Riksbank. 2020. The Riksbank's e-krona pilot. https://www.riksbank.se/globalassets/media/rapporter/e-krona/2019/the-riksbanks-e-krona-pilot.pdf.

[37] A. Rondelet and M. Zajac. 2019. ZETH: On Integrating Zerocash on Ethereum. arXiv preprint arXiv:1904.00905 (2019).

[38] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In 2014 IEEE symposium on security and privacy. IEEE, 459–474.

[39] Hyun Song Shin. 2009. Reflections on Northern Rock: The Bank Run that Heralded the Global Financial Crisis. Journal of Economic Perspectives 23, 1 (winter 2009), 101–119.

[40] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. 2022. UTT: Decentralized E-Cash with Accountable Privacy. Cryptology ePrint Archive (2022). Cited on p. 15.

[41] Nicolas van Saberhagen. 2013. CryptoNote v 2.0. https://cryptonote.org/whitepaper.pdf. Accessed: [Insert Date Here].

[42] Working Group on E-CNY Research and Development of the People's Bank of China. 2021. Progress of Research & Development of E-CNY in China. http://www.pbc.gov.cn/en/3688110/3688172/4157443/4293696/index.html

[43] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. 2022. Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy-Preserving Regulation. In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (Los Angeles, CA, USA) (CCS '22). Association for Computing Machinery, New York, NY, USA, 2947–2960. https://doi.org/10.1145/3548606.3560617

[44] Jiajun Xin, Arman Haghighi, Xiangan Tian, and Dimitrios Papadopoulos. 2025. Notus: dynamic proofs of liabilities from zero-knowledge RSA accumulators. In Proceedings of the 33rd USENIX Conference on Security Symposium (Philadelphia, PA, USA) (SEC '24). USENIX Association, USA, Article 82, 18 pages.

[45] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. vSQL: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases. In 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. IEEE Computer Society, 863–880. https://doi.org/10.1109/SP.2017.43

[46] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2015. IntegriDB: Verifiable SQL for Outsourced Databases. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (Denver, Colorado, USA) (CCS '15). Association for Computing Machinery, New York, NY, USA, 1480–1491. https://doi.org/10.1145/2810103.2813711