PulpoPay for Auditable Privacy CBDCs: Introducing High Concurrency and Offline Confirmation

Chenke Wang*, Shuangcheng Liu*, Yu Long*§, Xian Xu†§ and Dawu Gu*
* Shanghai Jiao Tong University, Shanghai, China
† East China University of Science and Technology, Shanghai, China
{w_chenke, lsc2021, longyu, dwgu}@sjtu.edu.cn, xuxian@ecust.edu.cn
§ Corresponding authors

***Abstract*—Central Bank Digital Currencies (CBDCs) aim to replicate the utility of physical cash in the digital era by balancing privacy with regulatory auditability. Despite recent advancements, existing CBDC schemes suffer from limited concurrency and the lack of offline payment confirmation, forcing users to process concurrent payments one by one and requiring counterparts to remain online. These constraints severely limit CBDC's applicability in high-frequency, real-world payment scenarios.**

This paper proposes PulpoPay, the first CBDC scheme to provide high concurrency and offline confirmation, while maintaining auditable privacy. To achieve high concurrency and offline confirmation, PulpoPay, at its core, introduces a coin-based payment protocol that facilitates users to concurrently withdraw, transfer, and deposit coins without synchronization or online requirements for counterparts. For auditable privacy, we leverage signatures on randomizable ciphertexts to enable fine-grained auditing without compromising privacy. Furthermore, we design a coin identifier with offline-checkable uniqueness to resist double-spending attacks even in disconnected (offline) environments. We formally prove PulpoPay's security within the Universal Composability (UC) framework.

We implement a prototype of PulpoPay. Experimental results demonstrate that (1) a payment with an offline receiver can be finished within 0.3 s using a mobile phone; (2) PulpoPay outperforms the state-of-the-art PEReDi [CCS'22] in the scenarios of concurrent payments. For a typical setting of 200 concurrent payments, PulpoPay reduces processing time by at least 66.89% and communication overhead by at least 76.45%. This advantage becomes even more significant as the number of concurrent payments grows.

I. Introduction

Central bank digital currencies (CBDCs) are digital equivalents to traditional physical cash (e.g., banknotes) issued by central banks. CBDCs could guarantee public access to central bank money in the digital world, facilitate users' access to various digital financial services, and offer auditable privacy in digital payments [23]. Driven by these benefits, central banks in over 130 countries, accounting for 98% of global GDP, are actively exploring the development and implementation of CBDCs. For instance, the European Central Bank [27] has launched a legislative proposal to adopt the digital euro and the People's Bank of China [18] has already initiated trials of a digital yuan. CBDCs are regarded as a crucial upgrade of the monetary system.

When serving as the digital replacement of traditional cash, CBDCs are desired to fulfill two fundamental properties: privacy and auditability. On the one hand, *privacy* ensures that payer/payee identities and payment values remain hidden from commercial banks. The guarantee of privacy addresses users' concerns about banks leveraging transaction data to infer sensitive information, such as social relationships, spending habits, and income levels. On the other hand, unconditional privacy in CBDC systems could exacerbate illegal activities [34], thus *auditability* is required to ensure compliance with financial regulations. Such regulations can be summarized as transaction limits, privacy revocation and the ability to trace payments involving suspicious users [42], [29].

Extensive efforts have been made to develop payment systems offering *auditable privacy* [11], [42], [25], [41], [19], [21], [40], [29]. Based on their underlying ledger models, these works can be divided into two categories. Some works [25], [40] use the UTXO-based model, where the assets of a user are independent transaction outputs (TXOs). These works can achieve payer/payee identity and value privacy. However, to enforce regulatory policies (e.g., holding or spending limits), they require aggregating and proving properties over a dispersed set of TXOs belonging to the same user, which often leads to complex operations. In contrast, the mainstream designs [11], [42], [41], [19], [21], [29] utilize the account-based model, where each user has a persistent account state, to simplify regulatory enforcement. Earlier account-based CBDC designs [11], [41], [19], [21] achieve payment value privacy but necessitate ledger maintainers having explicit knowledge of payer/payee identity to perform account updates. Consequently, these schemes *fail* to achieve identity privacy.

To further achieve identity privacy alongside auditability in the account-based model, Platypus [42] and PEReDi [29]

Network and Distributed System Security (NDSS) Symposium 2027
22 - 26 March 2027, Seoul, Republic of Korea
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2027.231987
www.ndss-symposium.org

utilize a new payment procedure that leverages users' self-sovereignty over their accounts. The payment procedure is illustrated in the *upper part* of Figure 1 and contains three stages. First, the sender and receiver confirm that they are mutually available (step ①). Next, they synchronously provide the updated accounts in a private and auditable manner (step ②). Finally, the ledger maintainer verifies these updates to ensure balance conservation (i.e., the amount deducted matches the amount credited) without gaining additional information. Once verified, the maintainer signs the updated accounts (step ③). As established in [29], this design achieves identity and value privacy, while preserving auditability.

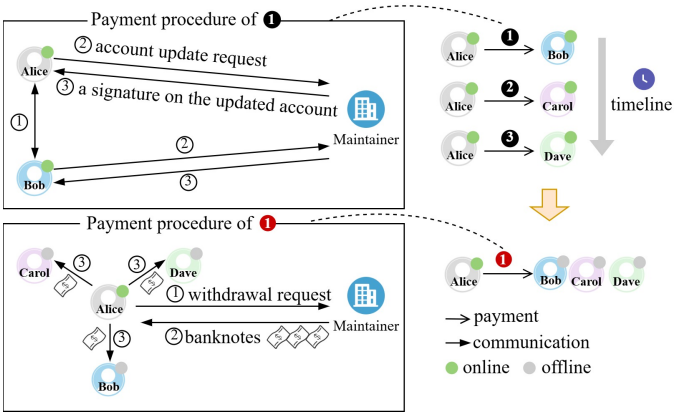


Fig. 1: The CBDC-specific procedure (above) vs. traditional cash procedure (below) when Alice wants to pay Bob, Carol and Dave.

However, such a payment procedure prevents these CBDCs from achieving two crucial properties intrinsic to traditional cash, namely, *high concurrency* and *receiver-side offline confirmation*. Such limitations hinder the practicality of CBDCs in real-world payment scenarios, and thus make them less comparable to traditional cash.

- **Limited concurrency issue.** For the payment procedure shown in the upper part of Figure 1, a user's account state is "locked" until the current transaction is finalized by the maintainer's signature (step ③). Consequently, if a user is involved in multiple concurrent payments, it has to process them *one-by-one* and bear counterparty delay for each payment. This severely throttles throughput in high-frequency payment scenarios like payroll or high-volume retail (cf. Section I-B). While one might consider aggregating multiple payments into a single batch, such a remedy is fragile: the abort or failure of any single counterparty would invalidate the entire batch, rendering the system unstable. In contrast, traditional cash can inherently support multiple concurrent payments. For example, as shown in the *lower part* of Figure 1, Alice can pay Bob, Carol, and Dave all at once. Therefore, to approximate this feature of traditional cash, CBDCs are desired to achieve high concurrency. It means that multiple concurrent payments involving the same user (as sender or receiver) can be conducted simultaneously without dependency on the state of the counterparty.
- **Inability to confirm payment offline.** Existing CBDCs with the above payment procedure force the receiver to go online and confirm receipt with the commercial bank. Therefore, these CBDCs are unsuitable for offline scenarios, such as public transit payments (cf. Section I-B), where the receiver may lack stable internet connections to the commercial bank. In contrast, the traditional cash system allows payments to be completed entirely offline: as depicted in the lower part of Figure 1, Bob, Carol, and Dave can instantly receive and *confirm* banknotes without internet access. Hence, to close the gap with the traditional cash system, CBDCs are expected to support *receiver-side offline confirmation* (hereafter referred to simply as "offline confirmation"). Specifically, (1) the receiver can locally confirm receipt of a payment without requiring an online connection to the bank; (2) Once online, the deposited amount is guaranteed to be credited to the receiver's account. Notably, the offline confirmation is different from offline payments with transferability, which will be discussed in Section I-C.

Revising the payment procedure to achieve both high concurrency and offline confirmation is far from straightforward. A direct attempt is to *decouple the sender's account update from the receiver's account update*. In this decoupled model, the sender updates their account state independently for multiple payments, while the receiver asynchronously updates their own account by referencing the sender's transactions recorded on the ledger. While this approach prevents payments from being disrupted by the counterparty, translating it into a secure protocol poses some challenges:

- The receiver cannot reveal the link between the two parties when proving that its account increment is derived from a specific, valid sender update in an auditable manner.
- It is necessary to prevent a receiver from updating their account repeatedly based on the same sender transaction.
- It is inherently difficult to convince an offline receiver, who lacks access to the continuously evolving ledger, that the sender's payment is indeed recorded and has not been double-spent.

To address these challenges, we propose PulpoPay, a novel CBDC that achieves high concurrency and offline confirmation, while maintaining auditable privacy.

A. Our Contributions

The contributions can be summarized as follows:

- **Cash-like payment procedure for CBDC.** Our procedure eliminates the synchronous interaction between transacting parties, allowing for asynchronous coin transfers. In this procedure, the sender can simultaneously withdraw multiple coins from its account, each tailored to a specific payment value and receiver. After receiving a coin, the receiver can confirm the receipt without connection to the bank and is guaranteed a successful coin deposit when online.
- **Novel techniques facilitating the payment procedure.** First, we introduce signatures on randomizable ciphertexts to CBDCs, enabling private payments and fine-grained auditing (i.e., disclosure of only the designated part, such

TABLE I: Comparison of existing payment systems with auditable privacy.

Scheme	Privacy		Auditability			High concurrency	Offline support	Distributed authority	Ledger model
	Identity	Value	Limits policy	Privacy revocation	Tracing				
E-cash [11]	◑	○	◑(S)	●	●	●	No	○	A
DAP [25]	●	●	◑(S)	●	●	●	No	●	U
PRCash [41]	◑	●	◑(S-R)	●	○	●	No	●	A
PGC [19]	○	●	●(S-R-H)	●	○	●	No	●	A
UTT [40]	●	●	◑(S)	○	○	●	No	●	U
Platypus [42]	●	●	●(S-R-H)	●	●	○	No	○	A
PEReDi [29]	●	●	●(S-R-H)	●	●	○	No	●	A
KAIME [21]	◑	●	○	●	○	●	No	●	A
PayOff [7]	◑	●	●(S-R-H)	○	○	●	P-BL*	○	A
[1]	●	○	○	●	○	●	P-SE*	●	A
DART [36]	●	●	○	●	○	●	No	●	A
[31]	◑†	◑†	●	●	●	●	P-SE*	○	A
PulpoPay	●	●	●(S-R-H)‡	●	●	●	Conf*	●	A

●: support; ○: no support; ◑: partially support; S: sending limits; R: receiving limits; H: holding limits; A: account-based; U: UTXO-based

∗ P-SE: Support offline payment, but is dependent on secure elements; P-BL: Support offline payment, but the system bears double-spending loss; Conf: Support offline confirmation. The comparison between offline payment and offline confirmation is detailed in Section I-C.

† The payment value and identities are disclosed to the commercial bank once the users are online.

‡ To enable high concurrency and offline confirmation, PulpoPay enforces holding and receiving limits at the point of online deposit rather than at offline receipt. This is acceptable because any coins that a user tries to avoid regulation by delaying the deposit would remain unspendable offline anyway.

as the balance, rather than the entire account/coin state). Second, we design a novel coin identifier (i.e., id) with offline-checkable uniqueness to resist double-spending attacks even in disconnected environments. With these techniques, PulpoPay achieves auditable privacy, high concurrency, and offline confirmation.

- **Formal security model and proof.** We define an ideal functionality $\mathcal{F}_{\mathsf{CBDC}}$ to formalize the security properties of CBDCs, including our proposed high concurrency and offline confirmation, in the Universal Composability (UC) framework. We show that PulpoPay UC-realizes $\mathcal{F}_{\mathsf{CBDC}}$. This ensures that PulpoPay can be securely integrated with other payment systems and into larger finance systems.
- **Implementation and comprehensive evaluation.** We present a proof-of-concept implementation of PulpoPay. First, our experiment demonstrates that, even if the receiver is offline, a payment can be finished within 0.3 s on a mobile phone (i.e., 0.229 s for online withdrawal and 0.015 s for offline confirmation). Thereafter, when the receiver is online, the deposit takes 0.201 s. To evaluate payment performance under high concurrency, we measure the processing time and communication overhead of a single user when processing multiple concurrent payments. Beyond the measurement in the optimistic case, we also consider the impact of counterparty-induced delays. The results demonstrate that PulpoPay increasingly outperforms the state-of-the-art PEReDi as the number of concurrent payments grows. While the two exhibit virtually identical performance for a single payment, as the scale expands to 200 payments, PulpoPay reduces processing time by at least 66.89% and communication overhead by at least 76.45%.

B. Application Scenarios of PulpoPay

Beyond widely adopted 1-to-1&online payments in CBDC research, PulpoPay can support more payment scenarios, which frequently require either highly concurrent processing or offline confirmation. Below, we discuss a few examples.

Salary payments (sender-intensive). Companies might execute hundreds of salary payments simultaneously to their employees during payroll. With PulpoPay, employers can withdraw hundreds of coins in various denominations in one batch, and distribute them to employees. PulpoPay guarantees that these salary payments will be completed successfully, even if some employees are offline or involved in other transactions.

Payments to e-commerce platforms (receiver-intensive). E-commerce platforms, such as Amazon, frequently process a large number of incoming transactions per second from their customers [15]. Instead of frequently collaborating with multiple users to update the account balance, PulpoPay makes receiving money from other users an easy game. Specifically, the e-commerce platform can efficiently confirm each incoming transaction by verifying the validity of the received coins. These coins can be deposited into the platform's account all at once, at any convenient time.

Public transit payments (offline transactions). In public transit payments, fare-collection devices on buses often operate without continuous or reliable internet connectivity. With PulpoPay, a passenger can tap an NFC-enabled phone against a fare-collection device, obtain the device's public key pk_{dev}, withdraw a coin bound to that device, and transfer it instantly to the device. Throughout the day, these devices remain offline while handling a large volume of transactions. During daily settlement, they come back online to deposit all received coins into their corresponding accounts. Crucially, this is achieved without relying on specialized Trusted Execution Environments (TEE) or secure hardware, distinguishing our approach from existing hardware-dependent offline payment schemes (cf. Section I-C).

C. Related Work

Many digital currencies with auditable privacy [11], [42], [25], [41], [19], [21], [40], [29] have been inspiring the design of CBDCs. A detailed comparison of these schemes is shown in Table I and introduced below.

Camenisch *et al.* [11] focus on the identity management of e-cash systems to allow compliance with current regulations. However, like existing e-cash systems [16], [10], [12], this work also leaks the receiver identities and payment values when coins are deposited. Garman *et al.* [25] add regulatory policies to Zerocash [37] by providing improved definitions. However, this approach requires the receiver to scan all transactions on the ledger, making it unsuitable for resource-constrained receivers. PRCash [41] leverages zero-knowledge proofs for commitment-based transactions to achieve value privacy and some degree of regulatory compliance. However, PRCash discloses payment identities to the validators (i.e., maintainers) and cannot allow regulators to trace suspicious users. PGC [19] constructs an auditable decentralized confidential payment (ADCP) system from the integrated signature and encryption scheme, and zero-knowledge proofs, but does not hide payment identities. UTT [40] hides payment identities and values via randomizable signatures on commitments [35] and zero-knowledge proofs. However, it only supports sending limits (i.e., users can anonymously send payments, but only up to a limited amount of currency per month).

Most recently, as the first CBDC scheme to provide privacy together with expressive regulation, Platypus [42] introduced a new payment procedure by combining the transaction processing model of e-cash with an account-based fund management model. Later, PEReDi [29] adopts the core techniques (i.e., users' self-sovereignty over their accounts) from Platypus. It further removes Platypus's reliance on a central authority for ledger maintenance and auditing, by utilizing threshold blind signatures [39] and ElGamal encryption. However, as explained in Section I, both Platypus and PEReDi incur limited concurrency. KAIME [21] focuses on modular privacy, i.e., value privacy can be added or removed as desired, but it cannot provide identity privacy. DART [36] is a decentralized tokenization system that provides privacy and supports privacy revocation. Compared with PEReDi, it supports non-interactive payments to improve concurrency, allowing an online sender to submit a spending transaction independently. However, it lacks experimental evaluation, and the payment procedure cannot support offline confirmation. Thus, DART cannot apply to scenarios where the receiver must decide whether to provide service immediately without network connectivity.

Offline confirmation vs. offline payments. A stronger notion, *offline payments*, requires that a payment can be conducted when *both the sender and receiver are offline*, and the coins received offline can be used in subsequent offline payments (i.e., *transferable*). This property remains essential for CBDCs [21], [6], [4], [32]. Several solutions enable offline payment depending on secure elements [1], [7], [31]. As pointed out by [7], such solutions offer no guarantees in case of hardware compromise, resulting in potentially significant payment fraud. Therefore, PayOff [7] focuses on an offline payment system that is robust to secure element failure. Nonetheless, PayOff cannot detect the double-spending until the coin is finally deposited online, which compromises the system-wide security.

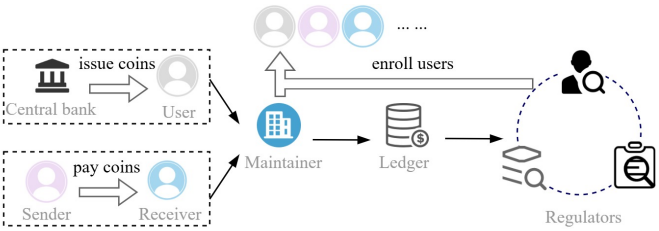


Fig. 2: System model of PulpoPay. The solid arrow means the transmission of transactions.

Consequently, achieving offline payments that are resilient to both hardware compromise and double-spending remains an open challenge. To address this, we propose *offline confirmation* as a strategic trade-off. Unlike full offline payments, offline confirmation prioritizes system-wide security by enforcing *non-transferability*. While it may require *the sender to be online in certain phases*, it ensures *the receiver remains offline* and can locally verify a payment's validity *without* relying on trusted hardware. This trade-off is highly practical for receiver-constrained scenarios, such as public transport, where immediate, cryptographically-secure receipt is more critical than subsequent offline transferability.

II. Technique Overview

A. CBDC Model

PulpoPay builds upon the retail CBDC model for public day-to-day payments, proposed in PEReDi [29]. This model is designed based on reports [23], [17] from real-world central banks, such as the Bank of Canada, the European Central Bank, and the Bank of Japan. We recap the model as below.

System model. Figure 2 shows the system model of PulpoPay, which is described as follows.

- Entities. PulpoPay comprises four types of entities: (1) The *central bank* issues currency and defines monetary policy. (2) The *Maintainer* (e.g., specific commercial bank) validates transactions and maintains the ledger. (3) Approved *regulators* audit the payment information within the ledger. (4) *Users* participate in the CBDC system as senders or receivers of payments. Notably, we consider a one-commercial-bank system only for notational simplicity, which can be naturally generalized to multi-banks system and support cross-bank payments (cf. Section VIII).
- Protocols. The entities run four protocols. (1) The *user registration protocol* enables legal users to join the CBDC system following the approval by the maintainer and the regulators. (2) The *currency issuance protocol* ensures that the central bank issues money to a user once the maintainer has verified the issuance request. (3) The *payment protocol,* which is the most fundamental, enables the sender to transfer funds to the receiver after the verification of the maintainer. (4) The *auditing protocol* permits a group of approved regulators, when necessary, to disclose payment information.

Threat model. We consider threats from different entities:

1) The central bank is trusted to determine the amount of currency being issued [17]. However, it may be curious about payment information.
2) The maintainer (e.g., a commercial bank) acts honestly in validating transactions but remains "honest-but-curious", meaning that they may seek to infer private information from transaction records stored in the ledger.
3) The regulators constitute an honest majority. However, a minority of them may misbehave and attempt to disclose private information without legitimate justification.
4) Users are regarded as untrusted. They may endanger other participants and the system through malicious activities. These activities can include financial fraud, transaction disruption, or unauthorized access to private data, etc.

This threat model aligns with established risk frameworks from global financial authorities. Specifically, the "honest-but-curious" maintainer and potential data misuse by central entities are formally recognized as critical privacy concerns in recent reports by international financial institutions [33], [3].

B. Security, Privacy, Availability Goals

We now informally present the design goals of a CBDC system. Building on the well-established security requirement of *integrity*, as well as the widely adopted privacy goals of *privacy* and *auditability* [29], [42], [41], [40], we additionally introduce *availability* (including high concurrency and offline confirmation), to bring digital payments closer in functionality to traditional cash. The formal definitions are in Section IV.

- *Integrity.* All transactions in a CBDC system should be operated in compliance with financial rules to prevent counterfeiting, theft, and double-spending.
- *Privacy.* This requirement consists of: (i) Transaction-specific details, including the payer/payee identities and payment values, are hidden from all parties except the involved sender and receiver. (ii) Given a transaction, no one except the user generating it can link any past transaction that provides funds used by the current transaction.
- *Auditability.* It imposes limits on account balances and transaction amounts. In addition, authorized regulators can (i) link all transactions related to a specific user, and (ii) reveal private information in transactions.
- *High concurrency.* Beyond payments concerning completely distinct sender-receiver pairs, multiple payments involving the same user (acting as sender or receiver) can be carried out simultaneously and independently.
- *Offline confirmation.* Without an internet connection to the bank, a receiver can receive digital coins via short-range communication technologies, such as NFC, and is guaranteed a successful coin deposit to its account when online.

C. Technical Challenges and Solutions

*1) **Challenge 1**. Enabling high concurrency without compromising integrity or privacy:* As explained in Section I, the fundamental reason for the limited concurrency of existing CBDC schemes is that the sender's and receiver's accounts are required to be updated simultaneously. To support high concurrency, our *high-level idea* is to replace this simultaneous update of both accounts with a simultaneous update of the sender's account and a newly generated coin (akin to a digitized banknote). The newly generated coin then serves as a surrogate for the receiver's account. By generating multiple coins as surrogates for different receivers, the sender can engage in multiple payment transactions concurrently.

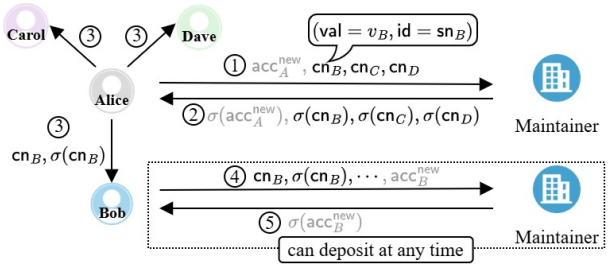


Fig. 3: A strawman payment procedure to provide high concurrency, where Alice wants to pay Bob, Carol, and Dave. Since the private account update is inherited from PEReDi [29], the detailed account-related operations are omitted and marked in gray.

Adopting the above idea, a strawman payment procedure where Alice wants to pay Bob, Carol, and Dave, is illustrated in Figure 3.

- Step ①. Alice updates its old account to a new account $\mathsf{acc}_A^{\mathsf{new}}$ and *generates coins* $\mathsf{cn}_B, \mathsf{cn}_C, \mathsf{cn}_D$ independently. Each coin is composed of the coin value v and a randomly chosen serial number sn as the unique coin identifier, i.e., $(\mathsf{cn.val} = v, \mathsf{cn.id} = \mathsf{sn})$.
- Step ②. The maintainer verifies the correctness of the new account $\mathsf{acc}_A^{\mathsf{new}}$ and coins $\mathsf{cn}_B, \mathsf{cn}_C, \mathsf{cn}_D$. Then, the maintainer signs on $\mathsf{acc}_A^{\mathsf{new}}$, $\mathsf{cn}_B, \mathsf{cn}_C, \mathsf{cn}_D$, and sends them back to Alice.
- Step ③. Alice sends these signed coins to multiple receivers, including Bob. Each receiver checks the coin signature validity for payment confirmation.
- Step ④. At any time, Bob can update its old account to a new account $\mathsf{acc}_B^{\mathsf{new}}$ by crediting the received coins, including cn_B, into its account.
- Step ⑤. The maintainer verifies: (1) the validity of coin signatures; (2) $\mathsf{cn.id}$ of these coins has never appeared in the ledger (i.e., to prevent double deposit); (3) the correctness of $\mathsf{acc}_B^{\mathsf{new}}$. Then, the maintainer signs on $\mathsf{acc}_B^{\mathsf{new}}$.

The above procedure decouples the sender's account update from the receiver's account update, and thus allows each user to process multiple payments concurrently. However, this scheme still faces the following issues that compromise integrity and privacy.

- *Issue 1-Integrity.* A malicious user can steal the coin sent from Alice to Bob (i.e., $\mathsf{cn}_B, \sigma(\mathsf{cn}_B)$) in step ③ and depositing the coin in front of Bob. This attack is possible because the coin can be deposited by any user.
- *Issue 2-Privacy.* As we introduce *coins*, the maintainer can easily know the payment details: (1) the payment amount, (2) sender-receiver correspondence. The latter is due to the

fact that the same coin and signature are used in step ② and step ④, enabling straightforward linkability.

Solutions to challenge 1. The solutions to address the above two issues are presented as follows.

For *issue 1*, to prevent a coin from being deposited by any non-target user, the sender embeds the receiver's identity into the coin, i.e., $(\mathsf{cn.val}=v, \underline{\mathsf{cn.rec}=\mathsf{pk}_r}, \mathsf{cn.id}=\mathsf{sn})$. After that, once the maintainer receives a deposit request from a user in step ④, it checks that the user's identity initiating the deposit matches the identity cn.rec in the coin. Therefore, once a coin is withdrawn, only the designated receiver can deposit it.

For *issue 2*-(1), to hide the payment information contained in coins without losing auditability, the sender *encrypts each coin* (e.g., encrypting cn as $\mathsf{cn}^{\mathcal{B}}$) with the regulator's public key. Accordingly, the sender generates an additional zero-knowledge proof π_A to convince the maintainer that these coins have the correct format and that the account is correctly updated with respect to these coins. Then, the sender transfers coin plaintext cn, the encrypted coin $\mathsf{cn}^{\mathcal{B}}$, the randomness r_{cn} for encryption, and the signature $\sigma(\mathsf{cn}^{\mathcal{B}})$ to the receiver. The receiver verifies the well-formedness of $\mathsf{cn}^{\mathcal{B}}$ via cn, r_{cn} and the validity of $\sigma(\mathsf{cn}^{\mathcal{B}})$. Finally, the receiver deposits the encrypted coins into its account, discloses cn.id to resist double deposit, and generates π_B as a proof of correctness. Therefore, nobody except the regulators can decrypt $\mathsf{cn}^{\mathcal{B}}$ and learn the private payment information contained in coins.

For *issue 2*-(2), to hide the link between the sender and receiver while maintaining auditability, we instantiate the coin-signature pair using Signatures on Randomizable Ciphertexts (SoRC). This primitive provides two key properties:

- *Unlinkability through randomization.* The sender *randomizes* the encrypted coin-signature pair before transmitting it to the receiver. This ensures that, from the maintainer's perspective, the randomized coin-signature pair will look like a random coin with a fresh signature.
- *Auditability via identity embedding.* To ensure the regulators to learn the link between the deposited coin with the sender, the sender's identity pk_s is embedded into the coin as an audit trapdoor, i.e., $(\mathsf{cn.val}=v, \underline{\mathsf{cn.snd}=\mathsf{pk}_s}, \mathsf{cn.rec}=\mathsf{pk}_r, \mathsf{cn.id}=\mathsf{sn})$. This allows regulators to reveal the payment's origin without compromising user privacy against the maintainer.

After solving issues 1 and 2, the revised payment procedure is shown in Figure 4.

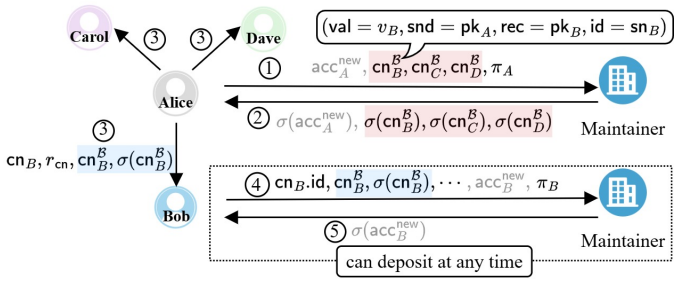


Fig. 4: A payment procedure enabling high concurrency. The randomization of the coin-signature pair is indicated by a colour change.

*2) **Challenge 2**. Achieving offline confirmation:* We recall that a successful PEReDi payment requires real-time connectivity with the maintainer for both the sender and receiver, which makes offline confirmation infeasible. This property of PEReDi persists in the new high-concurrency payment procedure: although an offline receiver can get signed coins, it may fail when depositing the coins later. Specifically, a malicious sender can send multiple coins with the same $\mathsf{cn.id}=\mathsf{sn}$ targeting different receivers. While offline, each receiver would verify only the coin's well-formedness and the validity of its signature. Unfortunately, only the first receiver who deposits the coin would succeed, since each sn can be recorded in the ledger only once. The key point is that an offline receiver cannot check the uniqueness of cn.id until it goes online to query the maintainer.

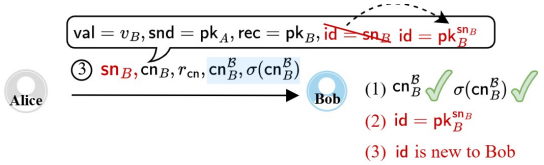


Fig. 5: Change the coin identifier to enable offline confirmation.

Solutions to challenge 2. Our idea of achieving offline confirmation lies in introducing an innovative coin id that enables an offline receiver to verify its uniqueness. Specifically, as shown in Figure 5, we design the coin id as $\mathsf{pk}_r^{\mathsf{sn}}$ in which pk_r is the public key of the coin receiver. To transfer a payment to Bob, Alice sends to Bob $(\mathsf{cn}_B, \sigma(\mathsf{cn}_B), \mathsf{sn})$, where B represents Bob. Once the *offline* Bob locally confirms that $\mathsf{cn}_B.\mathsf{id}$ is (1) equal to $\mathsf{pk}_B^{\mathsf{sn}}$, and (2) new to itself, Bob accepts the coin as unique. Subsequently, when depositing this coin, Bob discloses $\mathsf{pk}_B^{\mathsf{sn}}$ and proves that it knows the discrete logarithm of $\mathsf{pk}_B^{\mathsf{sn}}=h^{\mathsf{sk}_B\cdot\mathsf{sn}}$, i.e., $\mathsf{sk}_B\cdot\mathsf{sn}$, in a zero-knowledge way (here h is a generator of the corresponding group).

This coin id design guarantees, with overwhelming probability, that *the* id *of a coin is unique and well-formed for one receiver locally, then the* id *must be unique for the global ledger*. The technical essence of this design is that the coin id is cryptographically tied to the receiver's public key. Thus, a malicious sender is unable to generate two different pairs $(\mathsf{pk}_r, \mathsf{sn})$, $(\mathsf{pk}'_r, \mathsf{sn}')$ such that $\mathsf{id}=\mathsf{pk}_r^{\mathsf{sn}}=(\mathsf{pk}'_r)^{\mathsf{sn}'}$ (assuming that either pk_r or pk'_r is honest), due to the hardness assumption of the discrete-logarithm problem. This property is formalized as *offline-checkable uniqueness* and is proven in Theorem 1, Section VI. Notably, to support offline checks, we assume that each pk is tied to a unique device, avoiding multiple devices from synchronizing a shared id list.

3) PulpoPay: an efficient CBDC scheme: Till now, we have focused on the payment protocol and addressed the two mentioned challenges to achieve high concurrency and offline confirmation. As the next step, to construct a comprehensive construction, we should adapt the remaining protocols of PEReDi (user registration, currency issuance, and auditing protocols) to integrate the above solutions into the CBDC system and further improve efficiency. For example, we adapt

the currency issuance protocol in PEReDi [29] following the way we revised the payment protocol. This is because, in the currency issuance protocol, the central bank is equivalent to a sender with an unlimited balance. Detailed description and evaluation of all protocols are in Section V and VII.

III. Preliminaries

Notations. We assume the existence of a PPT algorithm $\mathcal{G}$ that, on input 1^λ, outputs a description of an (asymmetric) bilinear group $(p, \mathbb{G}, g, \hat{\mathbb{G}}, \hat{g}, \mathbb{G}_T, e)$. Here, groups $\mathbb{G}$ and $\hat{\mathbb{G}}$ are p-order groups generated by g and $\hat{g}$, respectively, and $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ is a bilinear map such that $e(g, \hat{g})$ generates $\mathbb{G}_T$, called pairing. For $n \in \mathbb{N}$, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. We say a probability is overwhelming if it is $1-\mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\lambda)$.

We briefly introduce core cryptographic primitives here to be self-contained in the main body. Moreover, we defer the detailed security properties and the auditing primitives, i.e., threshold secret sharing and threshold public key encryption, to Appendix A.

Signatures on randomizable ciphertexts. Signatures on randomizable ciphertexts [5] allow the signer to generate signatures on ciphertexts. Moreover, the ciphertext and signature pair can be randomized by anyone knowing neither the signing key nor the encrypted messages. Inheriting $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ from ElGamal encryption, the signatures on randomizable ciphertexts scheme (SoRC) has four additional algorithms. $\mathsf{SKeyGen}(pp)$ takes the public parameter pp as input and outputs a signing key $\hat{\mathsf{sk}}$ and a verification key $\hat{\mathsf{vk}}$. $\mathsf{Sign}(\hat{\mathsf{sk}}, \mathsf{ek}, \mathsf{ct})$ takes the signing key $\hat{\mathsf{sk}}$, the encryption key ek and a ciphertext ct as input, outputs a signature $\hat{\sigma}$. $\mathsf{Vf}(\hat{\mathsf{vk}}, \mathsf{ek}, \mathsf{ct}, \hat{\sigma})$ takes $\hat{\mathsf{vk}}, \mathsf{ek}$, a ciphertext ct, and a signature $\hat{\sigma}$ as input, outputs 1 if valid or 0 if invalid. $\mathsf{Rdm}(\mathsf{ek}, \mathsf{ct}, \hat{\sigma}, r^{\mathsf{rnd}})$ takes the encryption key ek, a ciphertext-signature pair $\mathsf{ct}, \hat{\sigma}$, and a randomness factor r^{rnd} as input, outputs a randomized ciphertext-signature pair $\mathsf{ct}^{\mathsf{rnd}}, \hat{\sigma}^{\mathsf{rnd}}$.

SoRC not only inherits the properties of the underlying encryption scheme, but also satisfies *class hiding*, *signature adaptability*, and *unforgeability*. Class hiding guarantees that a randomization of one ciphertext is indistinguishable from a random element of the ciphertext space. Signature adaptation requires that signatures on a randomization of the signed ciphertext are distributed like fresh signatures on the randomized ciphertext. Unforgeability prevents the adversary from producing a valid signature on a new encrypted messages.

Remark 1. *In this paper, we use the generalized version [5] of the threshold public encryption scheme and the* SoRC *scheme to support message vectors with length v. In our implementations, when $v=9$, account information is encrypted, whereas when $v = 4$, coin information is encrypted. Specifically, the signing key $\hat{\mathsf{sk}}$, the verification key $\hat{\mathsf{vk}}$, the ciphertext* ct *are extended to vectors. By contrast, the size of signatures $\hat{\sigma}$ remains independent of the message length v.*

Non-interactive zero-knowledge proof. Let $\mathsf{R} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be an NP relation with corresponding NP-language $L = \{x | \exists w \text{ s.t. } \mathsf{R}(x, w) = 1\}$. A non-interactive zero-knowledge proof (NIZK) [20] for the relation R consists of three algorithms. $\mathsf{Setup}(1^\lambda)$ takes the security parameter 1^λ as input and outputs a common reference string crs and a simulation trapdoor τ. $\mathsf{Prove}(\mathsf{crs}, x, w)$ takes a common reference string crs, a statement x, and a witness w as input, outputs a proof π to show the validity of x. $\mathsf{Vf}(\mathsf{crs}, x, \pi)$ can verify whether an proof π is valid or not.

A NIZK protocol is *complete* if the prover knowing a witness of $x \in L$ can always output a valid proof. *Knowledge soundness* means that for a valid proof π of an instance x, there exists a PPT extractor Ext who gets full access to the prover's state, can extract the witness w from π, where $(x, w) \in \mathsf{R}$. *Zero-knowledge* means that there exists a PPT algorithm Sim, taking the simulation trapdoor τ as an extra input, can simulate π for $x \in L$ without knowledge of w. This paper uses the ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ of NIZK formalized by Groth *et al.* [28].

IV. Formal Definition of PulpoPay

A. Security Model

We formalize PulpoPay in the universal composability (UC) framework [13]. We rely on the synchronous version of the global UC framework (GUC) [14].

Adversary model. We consider a *static* adversary $\mathcal{A}$, which corrupts parties at the beginning of execution. Once a party is corrupted, $\mathcal{A}$ gains access to its internal state and can fully control its behavior. A corrupted party may act maliciously by refusing to cooperate with honest parties or colluding with other parties.

Communication model. For securely transmitting information and avoiding privacy leakage at the network layer (e.g., the extraction of user identities by traffic analysis), PulpoPay employs different types of communication channels among participants. The authenticated channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ guarantees the legitimacy of the source, the target, and the integrity of each message. The full anonymous channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ delivers messages without leaking the sender identity, the receiver identity, and the message. The secure channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$ delivers messages without leaking the message. The sender-anonymous channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ delivers messages without leaking the sender's identity. The broadcast channel $\mathcal{F}_{\mathsf{BC}}$ delivers messages to multiple parties and does not guarantee the secrecy of messages.

The formal definitions of ideal functionalities $\mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{BC}}$ are given in Appendix F.

Security definition. $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}}, \mathcal{S}, \mathcal{Z}}(\lambda)$ denotes the the output of the environment $\mathcal{Z}$ when interacting with the ideal functionality $\mathcal{F}_{\mathsf{CBDC}}$ and the simulator $\mathcal{S}$. $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{NIZK}}}_{\Pi, \mathcal{A}, \mathcal{Z}}(\lambda)$ denotes the output of the environment $\mathcal{Z}$ when interacting with the real protocol Π and the adversary $\mathcal{A}$ in the $(\mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid model.

Definition 1. *A protocol Π running in the $(\mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{NIZK}})$-hybrid world UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{CBDC}}$, if for any* PPT *adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that*

$$\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}}, \mathcal{S}, \mathcal{Z}}(\lambda) \approx \mathsf{EXEC}^{\mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{NIZK}}}_{\Pi, \mathcal{A}, \mathcal{Z}}(\lambda),$$

where $\approx$ denotes that the indistinguishability.

B. Ideal Functionality of PulpoPay

In this section, we describe the ideal functionality of PulpoPay, i.e., $\mathcal{F}_{\mathsf{CBDC}}$. The definition of $\mathcal{F}_{\mathsf{CBDC}}$ is shown in Figure 6, which consists of four protocols. Notably, the *n-deposit-m-payment* protocol allows a user to deposit n coins and withdraw m coins for payments in a single transaction. The payment-only protocol corresponds to $n = 0$, and the deposit-only protocol corresponds to $m = 0$.

Concretely, the functionality $\mathcal{F}_{\mathsf{CBDC}}$ interacts with the central bank B, regulators $\{\mathsf{R}_j\}_{j\in[D]}$, users, and the simulator $\mathcal{S}$. It is parameterized by D, t, α, and Max, which are the number of regulators, the upper bound of the number of corrupted regulators, the threshold of audit, and the upper bound of the transaction value, respectively. Moreover, $D = 2t + 1$, $\alpha = t + 1$. The functionality maintains mappings as follows, which are initially empty.

1) $\mathsf{Tx}[\mathsf{pid}] = (\mathsf{U}_s, \mathsf{U}_r, v, 0/1)$. On input the payment identifier pid, $\mathsf{Tx}[\mathsf{pid}]$ outputs the sender U_s, receiver U_r, transaction value v, and status (0 is unfinished and 1 is finished).
2) $\mathsf{Acc}[\mathsf{U}] = (B, B_{\max}, S, S_{\max}, R, R_{\max})$. On input a user identity U, $\mathsf{Acc}[\mathsf{U}]$ outputs the account information, including the balance B, the sum of all sent values S, the sum of received values R, and their upper limits $B_{\max}, S_{\max}, R_{\max}$.
3) $\mathsf{Cn}[\mathsf{U}] = \{(\mathsf{U}_s, v)\}$. On input a user identity U, $\mathsf{Cn}[\mathsf{U}]$ outputs a *multiset* of undeposited coins of U, where each pair (U_s, v) represents sender identity and coin value.
4) $\mathsf{CnRvk}[\mathsf{cn}^{\mathcal{B}}] = (\mathsf{U}_s, \mathsf{U}_r, v)$. On input the encrypted coin $\mathsf{cn}^{\mathcal{B}}$, $\mathsf{CnRvk}[\mathsf{cn}^{\mathcal{B}}]$ outputs the identity U_s the coin comes from, the identity U_r the coin is sent to, and the coin value.
5) $\mathsf{AccRvk}[\mathsf{acc}^{\mathcal{B}}] = (B, B_{\max}, S, S_{\max}, R, R_{\max}, \mathsf{U})$. On input the encrypted account $\mathsf{acc}^{\mathcal{B}}$, $\mathsf{AccRvk}[\mathsf{acc}^{\mathcal{B}}]$ outputs $B, B_{\max}, S, S_{\max}, R, R_{\max}$, and the owner U.
6) $\mathsf{Trace}[\mathsf{U}] = \{(\mathsf{cn}^{\mathcal{B}}, \mathsf{snd}/\mathsf{rec})\}$. On input a user identity U, it outputs encrypted coins involved by U and the roles (i.e., sender or receiver).

User Registration.

1. Receiving $(\text{GenAcc}, sid, \mathsf{U}, B_{\max}, S_{\max}, R_{\max})$ from U, ignore if $\mathsf{Acc}[\mathsf{U}] \neq \perp$ or $B_{\max}, S_{\max}, R_{\max} \notin [0, \mathsf{Max}]$. Otherwise, send $(\text{GenAcc}, sid, \mathsf{U}, B_{\max}, S_{\max}, R_{\max})$ to $\mathcal{S}$.

2. Upon receiving $(\text{ok.GenAcc}, sid, \mathsf{U}, \mathsf{acc}^{\mathcal{B}})$ from $\mathcal{S}$:
- Set $\mathsf{Acc}[\mathsf{U}] = (0, B_{\max}, 0, S_{\max}, 0, R_{\max})$.
- Set $\mathsf{AccRvk}[\mathsf{acc}^{\mathcal{B}}] = (0, B_{\max}, 0, S_{\max}, 0, R_{\max})$.
- Output $(\text{AccGened}, sid, \mathsf{U})$ to U.

Currency Issuance.

1. Receiving $(\text{Issue}, sid, \{(\mathsf{U}_{r,j}, v_{r,j})\}_{j=1}^m)$ from B:
- Assert that $v_{r,j} \in [0, \mathsf{Max}]$.
- Randomly generate payment identifiers $\{\mathsf{pid}_j\}_{j=1}^m$.
- If $\mathsf{U}_{r,j}$ is corrupt, send $(\text{Issue}, sid, \mathsf{pid}_j, \mathsf{B}, \mathsf{U}_{r,j}, v_{r,j})$ to $\mathcal{S}$, otherwise, send $(\text{Issue}, sid, \mathsf{pid}_j, \mathsf{B})$ to $\mathcal{S}$.

2. Receiving $(\text{Issued}, sid, \{(\mathsf{pid}_j, \mathsf{cn}^{\mathcal{B}}_{r,j})\}_{j=1}^m)$ from $\mathcal{S}$:
- Set $\mathsf{Tx}[\mathsf{pid}_j] = (\mathsf{B}, \mathsf{U}_{r,j}, v_{r,j}, 0)$.
- Set $\mathsf{CnRvk}[\mathsf{cn}^{\mathcal{B}}_{r,j}] = (\mathsf{B}, \mathsf{U}_{r,j}, v_{r,j})$.
- Output (Issued, sid) to B.

3. Receiving $(\text{AcceptIss}, sid, \mathsf{B}, v_{r,j})$ from $\mathsf{U}_{r,j}$:
- Assert that $(\mathsf{B}, \mathsf{U}_{r,j}, v_{r,j}, 0)$ is recorded in the mapping Tx.
- Retrieve pid such that $\mathsf{Tx}[\mathsf{pid}] = (\mathsf{B}, \mathsf{U}_{r,j}, v_{r,j}, 0)$, randomize the payment identifier to be pid'.
- If B is corrupt, send $(\text{AcceptIss}, sid, \mathsf{pid}', \mathsf{B}, \mathsf{U}_{r,j}, v_{r,j})$ to $\mathcal{S}$, otherwise, send $(\text{Payee}, sid, \mathsf{pid}')$ to $\mathcal{S}$.

4. Receiving $(\text{ok}, sid, \mathsf{pid}')$ from $\mathcal{S}$:
- Update $\mathsf{Tx}[\mathsf{pid}] = (\mathsf{B}, \mathsf{U}_{r,j}, v_{r,j}, 1)$ and $\mathsf{Cn}[\mathsf{U}_{r,j}] \cup \{(\mathsf{U}, v_{r,j})\}$.
- Output $(\text{Accepted}, sid)$ to $\mathsf{U}_{r,j}$.

***n*-Deposit-*m*-Payment.**

1. Receiving $(\text{UpAcc}, sid, \{(\mathsf{U}_{s,i}, v_{s,i})\}_{i=1}^n, \{(\mathsf{U}_{r,j}, v_{r,j})\}_{j=1}^m)$ from U:
- Retrieve $\mathsf{Acc}[\mathsf{U}] = (B, B_{\max}, S, S_{\max}, R, R_{\max})$.
- Assert that
  - $\{(\mathsf{U}_{s,i}, v_{s,i})\}_{i=1}^n \subseteq \mathsf{Cn}[\mathsf{U}]$.
  - For $j \in [m]$, $v_{r,j} \in [0, \mathsf{Max}]$.
  - $B' \stackrel{\text{def}}{=} B + \sum_{i=1}^n v_{s,i} - \sum_{i=1}^m v_{r,j} \in [0, B_{\max}]$.
  - $S' \stackrel{\text{def}}{=} S + \sum_{i=1}^m v_{r,j} \in [0, S_{\max}]$.
  - $R' \stackrel{\text{def}}{=} R + \sum_{i=1}^n v_{s,i} \in [0, R_{\max}]$.
- Randomly generate deposit identifiers $\{\mathsf{did}_i\}_{i=1}^n$ and payment identifiers $\{\mathsf{pid}_j\}_{j=1}^m$.
- If $\mathsf{U}_{s,i}$ is corrupt, send $(\text{Deposit}, sid, \mathsf{did}_i, \mathsf{U}_{s,i}, \mathsf{U}, v_{s,i})$ to $\mathcal{S}$, otherwise, send $(\text{Deposit}, sid, \mathsf{did}_i)$ to $\mathcal{S}$.
- If $\mathsf{U}_{r,j}$ is corrupt, send $(\text{Payer}, sid, \mathsf{pid}_j, \mathsf{U}, \mathsf{U}_{r,j}, v_{r,j})$ to $\mathcal{S}$, otherwise, send $(\text{Payer}, sid, \mathsf{pid}_j)$ to $\mathcal{S}$.

2. Receiving $(\text{ok.UpAcc}, sid, \{\mathsf{cn}^{\mathcal{B}}_{s,i}\}_{i=1}^n, \{\mathsf{pid}_j, \mathsf{cn}^{\mathcal{B}}_{r,j}\}_{j=1}^m, \mathsf{acc}^{\mathcal{B}})$ from $\mathcal{S}$:
- Record
  - $\mathsf{Cn}[\mathsf{U}] = \mathsf{Cn}[\mathsf{U}] \setminus \{(\mathsf{U}_{s,i}, v_{s,i})\}_{i=1}^n$.
  - $\mathsf{Tx}[\mathsf{pid}_j] = (\mathsf{U}, \mathsf{U}_{r,j}, v_{r,j}, 0)$.
  - $\mathsf{CnRvk}[\mathsf{cn}^{\mathcal{B}}_{s,i}] = (\mathsf{U}_{s,i}, \mathsf{U}, v_{s,i})$.
  - $\mathsf{CnRvk}[\mathsf{cn}^{\mathcal{B}}_{r,j}] = (\mathsf{U}, \mathsf{U}_{r,j}, v_{r,j})$.
  - $\mathsf{AccRvk}[\mathsf{acc}^{\mathcal{B}}] = (B', B_{\max}, S', S_{\max}, R', R_{\max}, \mathsf{U})$.
- Update
  - $\mathsf{Acc}[\mathsf{U}] = (B', B_{\max}, S', S_{\max}, R', R_{\max})$,
  - $\mathsf{Trace}[\mathsf{U}] \cup \{\{(\mathsf{cn}^{\mathcal{B}}_{r,j}, \mathsf{snd})\}_{j=1}^m, \{(\mathsf{cn}^{\mathcal{B}}_{s,i}, \mathsf{rec})\}_{i=1}^n\}$.
- Output $(\text{AccUpdated}, sid)$ to U.

3. Receiving $(\text{Payee}, sid, \mathsf{U}, v_{r,j})$ from $\mathsf{U}_{r,j}$:
- Assert that $(\mathsf{U}, \mathsf{U}_{r,j}, v_{r,j}, 0)$ is recorded in the mapping Tx.
- Retrieve pid such that $\mathsf{Tx}[\mathsf{pid}] = (\mathsf{U}, \mathsf{U}_{r,j}, v_{r,j}, 0)$, randomize the payment identifier to be pid'.
- If U is corrupt, send $(\text{Payee}, sid, \mathsf{pid}', \mathsf{U}, \mathsf{U}_{r,j}, v_{r,j})$ to $\mathcal{S}$, otherwise, send $(\text{Payee}, sid, \mathsf{pid}')$ to $\mathcal{S}$.

4. Receiving $(\text{ok.Payee}, sid, \mathsf{pid}')$ from $\mathcal{S}$:
- Update $\mathsf{Tx}[\mathsf{pid}] = (\mathsf{U}, \mathsf{U}_{r,j}, v_{r,j}, 1)$ and $\mathsf{Cn}[\mathsf{U}_{r,j}] \cup \{(\mathsf{U}, v_{r,j})\}$.
- Output $(\text{Received}, sid)$ to $\mathsf{U}_{r,j}$.

Auditing.

1. Receiving $(\text{Revoke}, sid, X^j = \mathsf{cn}^{\mathcal{B}}/\mathsf{acc}^{\mathcal{B}}/\mathsf{U})$ from R_j:
- Ignore if $\mathsf{CnRvk}/\mathsf{AccRvk}/\mathsf{Trace}[X^j] = \perp$.
- Send $(\text{Revoke}, sid, X^j, \mathsf{R}_j)$ to $\mathcal{S}$.
- Once $|\{j | X^j = X\}| \geq \alpha$, record X.

2. Receiving $(\text{ok.Revoke}, sid, X)$ from $\mathcal{S}$:
- Send $(\text{Revoked}, sid, \mathsf{CnRvk}/\mathsf{AccRvk}/\mathsf{Trace}[X])$ to $\mathcal{S}$, R_j.

Fig. 6: The ideal functionality $\mathcal{F}_{\mathsf{CBDC}}$.

Integrity. In $\mathcal{F}_{\mathsf{CBDC}}$, the account information can be correctly updated after checking validity. If U receives a coin with value v from another user (e.g., U_s), this coins will be added to $\mathsf{Cn}[\mathsf{U}]$ (i.e., $\mathsf{Cn}[\mathsf{U}] \cup \{(\mathsf{U}_s, v)\}$). When U wants to deposit this coin, $\mathcal{F}_{\mathsf{CBDC}}$ checks if $\{(\mathsf{U}_s, v)\} \subseteq \mathsf{Cn}[\mathsf{U}]$. After receiving the paid message from $\mathcal{S}$, $\mathcal{F}_{\mathsf{CBDC}}$ removes $\{(\mathsf{U}_s, v)\}$ from $\mathsf{Cn}[\mathsf{U}]$. Therefore, each user U cannot deposit a coin not in $\mathsf{Cn}[\mathsf{U}]$, which prevents users from stealing coins or printing new coins.

Privacy. During the n-deposit-m-payment protocol, the simulator $\mathcal{S}$ cannot receive the links between honest transactions, the secret information of an honest transaction, and the account

status of any honest user from $\mathcal{F}_{\mathsf{CBDC}}$. It only learns the fact that there is an ongoing payment or deposit request.

Auditability. $\mathcal{F}_{\mathsf{CBDC}}$ maintains $\mathsf{CnRvk}, \mathsf{AccRvk}, \mathsf{Trace}$. When auditing, $\mathcal{F}_{\mathsf{CBDC}}$ can disclose the private information of accounts and coins, and link all coins involving a user.

High concurrency. In the n-deposit-m-payment phase, $\mathcal{F}_{\mathsf{CBDC}}$ allows the user U to process n deposits and m payments simultaneously. Moreover, $\mathcal{F}_{\mathsf{CBDC}}$ updates the user's account without interacting with the user's counterparties.

Offline confirmation. When the user U pays a receiver (e.g., $\mathsf{U}_{r,j}$) in the n-deposit-m-payment phase, $\mathcal{F}_{\mathsf{CBDC}}$ directly outputs $(\text{Received}, sid)$ to $\mathsf{U}_{r,j}$ without crediting the received value to $\mathsf{U}_{r,j}$'s account. As a result, $\mathsf{U}_{r,j}$ can confirm the receipt offline, without requiring a ledger connection to update the account.

V. Detailed Construction of PulpoPay

System setup. Let the public parameters pp be a description of an (asymmetric) bilinear group, i.e., $pp = (p, \mathbb{G}, g, \hat{\mathbb{G}}, \hat{g}, \mathbb{G}_T, e)$. The entities in PulpoPay are initialized as follows.

- The central bank B generates the public-secret key pair $(\mathsf{pk}_\mathsf{B} = h^{\mathsf{sk}_\mathsf{B}}, \mathsf{sk}_\mathsf{B})$.
- The maintainer M^* generates a verification-signing key pair $(\hat{\mathsf{vk}}_{\mathsf{M}^*}, \hat{\mathsf{sk}}_{\mathsf{M}^*}) \leftarrow \mathsf{SoRC.SKeygen}(pp)$.
- Distributed regulators executes $\mathsf{SoRC.KeyGen}(pp, D, \alpha)$ multiple times to generate $\{\mathsf{ek}_i, \{(\mathsf{ek}_{\mathsf{R}_j,i}, \mathsf{dk}_{\mathsf{R}_j,i})\}_{j=1}^{D}\}_{i=1}^{13}$. Each regulator R_j takes one out of D shares. Different encryption-decryption key pairs are responsible for disclosing different secret information. $\vec{\mathsf{ek}}_{\mathsf{acc}} = (\mathsf{ek}_1, \ldots, \mathsf{ek}_9)$ is for accounts and $\vec{\mathsf{ek}}_{\mathsf{cn}} = (\mathsf{ek}_{10}, \ldots, \mathsf{ek}_{13})$ is for coins.

Furthermore, we assume that all parties' registered public keys, the maintainers' verification keys, and the encryption keys are accessible on demand for any party.

A. User Registration

We adapt the user registration protocol from PEReDi [29], replacing the commitment-based account with an encryption-based one. While commitments do well in providing privacy, an encrypted account further supports practical auditing, as it allows authorized regulators to recover account states (detailed in Section V-C). Focusing on PulpoPay's core payment protocol while ensuring the paper remains self-contained, we present the complete protocol in Figure 15, Appendix B, and provide a high-level overview here. This overview is crucial as the user registration protocol establishes the initial state of the account that underpins the subsequent payments.

Each user generates an initial account $\mathsf{acc} = (B, S, R, B_{\max}, S_{\max}, R_{\max}, \mathsf{pk}_\mathsf{U}, \mathsf{AccTag}, a)$. For the initial account, the balance B, the cumulative sent amount S, and cumulative received amount R are set to 0, while $B_{\max}, S_{\max}, R_{\max}$ are initialized to specify per-user limits on B, S, R, respectively. The public key $\mathsf{pk}_\mathsf{U} = h^{\mathsf{sk}_\mathsf{U}}$ is generated to uniquely identify the user. The tracing key $a \leftarrow\!\$\ \mathbb{Z}_p^*$ and the tracing tag $\mathsf{AccTag} = h^{a^x}$ are used to enable tracing of account update operations, where x is a counter for account updates and is initialized to 1. To support (α, D)-threshold tracing, the user distributes shares $\{a_j\}_{j=1}^{D}$ among D regulators.

The user then sends an encrypted account $\mathsf{acc}^{\mathcal{B}}$ under $\vec{\mathsf{ek}}_{\mathsf{acc}}$, together with the zero-knowledge proof of well-formedness, to the maintainer M^*. The maintainer M^* performs Know-Your-Customer (KYC) verification, binding the digital account to the user's real-world identity (e.g., verifying physical credentials or biometric data). Once verified, the maintainer signs on $\mathsf{acc}^{\mathcal{B}}$, thereby enrolling the user U into the system.

B. n-Deposit-m-Payment

In this section, we explain the n-deposit-m-payment protocol, which enables a user to concurrently deposit n coins and process multiple payments for m receivers. Even better, it enables these receivers to confirm the payments offline and postpone the deposit until the receiver intends to spend them. The detailed procedure is described in Figure 7. Notably, the currency issuance protocol can be viewed as a simplified n-deposit-m-payment protocol and we present it in Appendix C. We describe the operations of the user, the maintainer, and the receivers at a high level as follows.

The user deposits n coins $(\mathsf{cn}_{s,i}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{cn}_{s,i}})$ and proving that it is the appointed receiver of these coins. To initiate m payments, the user U generates m coins, i.e., $\{\mathsf{cn}_{r,j} = (v_{r,j}, \mathsf{pk}_\mathsf{U}, \mathsf{pk}_{r,j}, \mathsf{id}_{r,j})\}_{j=1}^{m}$. Here, $v_{r,j}, \mathsf{pk}_\mathsf{U}, \mathsf{pk}_{r,j}$ are the coin value, the public key of the sender (i.e., the user U), and the public key of the receiver, respectively. $\mathsf{id}_{r,j} = \mathsf{pk}_{r,j}^{\mathsf{sn}_{r,j}}$ is a unique coin id, which is derived from the receiver's public key $\mathsf{pk}_{r,j}$ and a randomly chosen serial number $\mathsf{sn}_{r,j}$. Then, the user encrypts these coins under $\vec{\mathsf{ek}}_{\mathsf{cn}}$ and generates the well-formedness proof. Finally, the user privately updates its account according to the above coins and generates a zero-knowledge proof for validity. All this information is packaged into a transaction by the user to request signatures of the updated account and these new coins from the maintainer M^*.

For the maintainer M^*, it verifies the request and then generates signatures on both the newly updated account and these new coins. Then, the maintainer M^* appends the transaction to the ledger and replies to the user with these signatures.

For the user U, after receiving these signatures from the maintainer, it randomizes the encrypted coins and the account together with the signatures (via $\mathsf{SoRC.Rdm}$) for unlinkability. Then, the user transfers the secret coin information to the corresponding receivers through a secure and anonymous channel. Otherwise, others can trivially link the sender and receiver by observing the channel. *It is worth noting that the secure channel is only necessary for protecting the payment privacy, not the integrity.* Even if the transferred messages are learned by others, they cannot spend the coin, since this coin can only be spent by the targeted receiver.

For the offline receiver $\mathsf{U}_{r,j}$, it checks the coin's validity and uniqueness without querying the online ledger. When $\mathsf{U}_{r,j}$ is online, $\mathsf{U}_{r,j}$ can execute the entire 1-deposit-0-payment protocol to deposit this coin and credit it to its account.

Receiving $(\text{UpAcc}, sid, \{\mathsf{U}_{s,i}, v_{s,i}\}_{i=1}^{n}, \{\mathsf{U}_{r,j}, v_{r,j}\}_{j=1}^{m})$ from $\mathcal{Z}$, the user U:

1. For $j \in [m]$, sets $\mathsf{cn}_{r,j} = (v_{r,j}, \mathsf{pk}_{\mathsf{B}}, \mathsf{pk}_{r,j}, \mathsf{id}_{r,j})$ where $\mathsf{id}_{r,j} = \mathsf{pk}_{r,j}^{\mathsf{sn}_{r,j}}, \mathsf{sn}_{r,j} \leftarrow\$ \mathbb{Z}_p^*$.
2. Computes $\mathsf{cn}_{r,j}^{\mathcal{B}} \leftarrow \mathsf{SoRC.Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{cn}_{r,j}, r_{\mathsf{cn}_{r,j}})$.
3. Discloses the coin tags $\{\mathsf{id}_{s,i}\}_{i=1}^{n}$ of coins to be deposited.
4. Parses $\mathsf{acc} = (B, \cdot, S, \cdot, R, \cdot, \cdot, h^{a^x}, \cdot)$ and defines
   - $B' = B + \sum_{i=1}^{n} v_{s,i} - \sum_{j=1}^{m} v_{r,j}$; $S' = S + \sum_{j=1}^{m} v_{r,j}$; $R' = R + \sum_{i=1}^{n} v_{s,i}$.
5. Updates acc to $\mathsf{acc}^{\mathsf{new}} = (B', \cdot, S', \cdot, R', \cdot, \cdot, h^{a^{x+1}}, \cdot)$.
6. Computes $\mathsf{acc}^{\mathcal{B},\mathsf{new}} \leftarrow \mathsf{SoRC.Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{acc}^{\mathsf{new}}, r_{\mathsf{acc}}^{\mathsf{new}})$ and discloses $\mathsf{AccTag} = h^{a^x}$.
7. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\text{prove}, sid, x, w)$ and receives $(\text{proof}, sid, \pi_{\text{trans}})$. Here, $x = (\vec{\mathsf{ek}}_{\mathsf{cn}}, \vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{AccTag}, \{\mathsf{id}_{s,i}, \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}\}_{i=1}^{n}, \{\mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^{m}, \mathsf{acc}^{\mathcal{B}}, \mathsf{acc}^{\mathcal{B},\mathsf{new}})$ and witness $w = (\{v_{r,j}, \mathsf{sk}_{\mathsf{U}}, r_{\mathsf{cn}_{r,j}}\}_{j=1}^{m}, \{\mathsf{sk}_{\mathsf{U}}, \mathsf{sn}_{s,i}, r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}\}_{i=1}^{n}, \mathsf{acc}, r_{\mathsf{acc}}, r_{\mathsf{acc}}^{\mathsf{new}})$ satisfying
   - *Coin-format correctness:* For all j, $\mathsf{cn}_{r,j}^{\mathcal{B}}$ encrypts $v_{r,j}$, pk_{U}.
   - *Disclose-correctness:* AccTag is encrypted by $\mathsf{acc}^{\mathcal{B}}$. For $i \in [n]$, $\mathsf{id}_{s,i}$ is encrypted by $\mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}$ and $\mathsf{id}_{s,i} = h^{\mathsf{sk}_{\mathsf{U}} \cdot \mathsf{sn}_{s,i}}$.
   - *Account-update correctness:* $\mathsf{acc}^{\mathcal{B},\mathsf{new}}$ encrypts $\mathsf{acc}^{\mathsf{new}}$ which is updated from acc, $\{v_{s,i}\}_{i=1}^{n}$, and $\{v_{r,j}\}_{j=1}^{m}$.
   - *Identity consistency:* The receiver encrypted by $\mathsf{cn}_{s,i}^{\mathcal{B}}$, the sender encrypted by $\mathsf{cn}_{r,j}^{\mathcal{B}}$, and the owner encrypted by $\mathsf{acc}^{\mathcal{B}}$ are the same as pk_{U}, where $\mathsf{pk}_{\mathsf{U}} = h^{\mathsf{sk}_{\mathsf{U}}}$.
   - *Range proof:* For $j \in [m]$, $v_{r,j} \in [0, \mathsf{Max}]$, $B' \in [0, B_{\max}]$, $S' \in [0, S_{\max}]$, $R' \in [0, R_{\max}]$.
8. Packages transaction information $\mathsf{TI}_s = (x, \pi_{\text{trans}}, \{\hat{\sigma}_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}\}_{i=1}^{n}, \hat{\sigma}_{\mathsf{acc}})$ and calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\text{Send}, sid, \mathsf{M}^*, \mathsf{TI}_s)$.

Receiving $(\text{Receive}, sid, \mathsf{TI}_s, \mathsf{mid}_s)$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, the maintainer M^*:

1. Parses TI_s as $(x, \pi_{\text{trans}}, \{\hat{\sigma}_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}\}_{i=1}^{n}, \hat{\sigma}_{\mathsf{acc}})$.
2. Ignores this message if at least one of the conditions holds:
   - AccTag, $\{\mathsf{id}_{s,i}\}_{i=1}^{n}$ exist in the local list of M^*.
   - For (x, π_{trans}), calls $\mathcal{F}_{\mathsf{NIZK}}$ to receive $(\text{Verification}, sid, 0)$.
   - $\mathsf{SoRC.Vf}(\hat{\mathsf{vk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{acc}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{acc}}) = 0$.
   - $\exists i \in [n]$, $\mathsf{SoRC.Vf}(\hat{\mathsf{vk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}, \hat{\sigma}_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}) = 0$.
3. For $j \in [m]$, computes $\hat{\sigma}_{\mathsf{cn}_{r,j}} \leftarrow \mathsf{SoRC.Sign}(\hat{\mathsf{sk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{cn}_{r,j}^{\mathcal{B}})$.
4. Computes $\hat{\sigma}_{\mathsf{acc}}^{\mathsf{new}} \leftarrow \mathsf{SoRC.Sign}(\hat{\mathsf{sk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{acc}^{\mathcal{B},\mathsf{new}})$.
5. Records AccTag, $\{\mathsf{id}_{s,i}\}_{i=1}^{n}$ and appends TI_s into the ledger $\mathcal{L}$.
6. Calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\text{Send}, sid, \mathsf{mid}_s, \{\hat{\sigma}_{\mathsf{cn}_{r,j}}\}_{j=1}^{m}, \hat{\sigma}_{\mathsf{acc}}^{\mathsf{new}})$.

Receiving $(\text{Receive}, sid, \mathsf{M}^*, \{\hat{\sigma}_{\mathsf{cn}_{r,j}}\}_{j=1}^{m}, \hat{\sigma}_{\mathsf{acc}}^{\mathsf{new}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, U:

1. Ignores if $\mathsf{SoRC.Vf}(\hat{\mathsf{vk}}_{\mathsf{M}^*}, \mathsf{acc}^{\mathcal{B},\mathsf{new}}, \hat{\sigma}_{\mathsf{acc}}^{\mathsf{new}}) = 0$, or exists $j \in [m]$, $\mathsf{SoRC.Vf}(\hat{\mathsf{vk}}_{\mathsf{M}^*}, \mathsf{cn}_{r,j}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{cn}_{r,j}}) = 0$.
2. Computes $(\mathsf{cn}_{r,j}^{\mathcal{B},\mathsf{rnd}}, \hat{\sigma}_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}}) \leftarrow \mathsf{SoRC.Rdm}(\mathsf{cn}_{r,j}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{cn}_{r,j}}, r'_{\mathsf{cn}_{r,j}})$, and sets $r_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}} = r_{\mathsf{cn}_{r,j}} + r'_{\mathsf{cn}_{r,j}}$.
3. Computes $(\mathsf{acc}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{acc}}) \leftarrow \mathsf{SoRC.Rdm}(\mathsf{acc}^{\mathcal{B},\mathsf{new}}, \hat{\sigma}_{\mathsf{acc}}^{\mathsf{new}}, r'_{\mathsf{acc}})$, and sets $r_{\mathsf{acc}} = r_{\mathsf{acc}}^{\mathsf{new}} + r'_{\mathsf{acc}}$.
4. Packages secret transaction information $\mathsf{sTI}_{r,j} = (\mathsf{cn}_{r,j}^{\mathcal{B},\mathsf{rnd}}, \hat{\sigma}_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}}, \mathsf{sn}_{r,j}, \mathsf{cn}_{r,j}, r_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}})$ and calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ with input $(\text{Send}, sid, \mathsf{U}, \mathsf{sTI}_{r,j})$.
5. Outputs $(\text{AccUpdated}, sid)$ to $\mathcal{Z}$.

Receiving $(\text{Payee}, sid, \mathsf{U}, v_{r,j})$ from $\mathcal{Z}$, the *offline* receiver $\mathsf{U}_{r,j}$:

1. Waits for $(\text{Receive}, sid, \mathsf{U}, \mathsf{sTI}_{r,j})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ and parses $\mathsf{sTI}_{r,j}$ as $(\mathsf{cn}_{r,j}^{\mathcal{B},\mathsf{rnd}}, \hat{\sigma}_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}}, \mathsf{sn}_{r,j}, \mathsf{cn}_{r,j}, r_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}})$.
2. Ignores if at least one of the following conditions holds:
   - $\mathsf{cn}_{r,j} \neq (v_{r,j}, \mathsf{pk}_{\mathsf{U}}, \mathsf{pk}_{r,j}, \mathsf{pk}_{r,j}^{\mathsf{sn}_{r,j}})$.
   - $\mathsf{id}_{r,j}$ exists in the $U_{r,j}$'s local id list of accepted coins.
   - $\mathsf{cn}_{r,j}^{\mathcal{B},\mathsf{rnd}} \neq \mathsf{SoRC.Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{cn}_{r,j}, r_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}})$.
   - $\mathsf{SoRC.Vf}(\hat{\mathsf{vk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{cn}_{r,j}^{\mathcal{B},\mathsf{rnd}}, \hat{\sigma}_{\mathsf{cn}_{r,j}}^{\mathsf{rnd}}) = 0$.
3. Appends $\mathsf{id}_{r,j}$ to the local id list and outputs $(\text{Received}, sid)$.
4. When online, $\mathsf{U}_{r,j}$ can execute the entire 1-deposit-0-payment protocol to deposit the coin to its account.

Fig. 7: The procedure of n-deposit-m-payment protocol.

C. Auditing

PulpoPay proposes a new n-deposit–m-payment protocol that supports high concurrency and offline confirmation, while retaining the *user tracing* and *privacy revocation of transactions* capabilities of PEReDi [29]. User tracing enables linking all coins owned, sent, or received by a suspicious user, by reconstructing the tracing tag AccTag using the secret shares of a. Privacy revocation leverages threshold decryption to reveal targeted transaction details (e.g., sender, receiver, or transaction amount).

Beyond these inherited features, PulpoPay introduces a more efficient, fine-grained auditing mechanism. Its core advantage is that regulators can directly decrypt specific account attributes (such as a user's latest balance) from the current ledger state. This bypasses the costly process of backtracking to decrypt and trace the entire history to reconstruct the user's balance, as required in PEReDi. Moreover, PulpoPay uses partitioned encryption keys to selectively disclose only desired information (e.g., the balance B only) without exposing other private data. Due to the similarity of underlying techniques, we defer the full auditing protocol to Appendix D.

VI. Security Analysis

Theorem 1. *The* $\mathsf{cn.id} = \mathsf{pk}_r^{\mathsf{sn}}$*, where* pk_r *is the public key of the receiver and* sn *is chosen by the sender, has* ***offline-checkable uniqueness****: for an adversary, namely, a malicious sender providing* $\mathsf{cn.id}$ *and* sn*, if an honest and offline receiver confirms that* $\mathsf{cn.id}$ *is (1) equal to* $\mathsf{pk}_r^{\mathsf{sn}}$*, and (2) new to the receiver, this coin* id *will be unique for our global ledger with an overwhelming probability.*

Proof. If there exists an adversary $\mathcal{A}$ that creates a coin id breaking the offline-checkable uniqueness, we can construct an adversary $\mathcal{B}$ to break the discrete logarithm problem (DLP). During the reduction, $\mathcal{B}$ receives h, pk from the challenger of the discrete logarithm problem. $\mathcal{B}$ assigns the pk to be the public key of an honest user, executes PulpoPay on behalf of honest parties, and interacts with $\mathcal{A}$. Since $\mathcal{B}$ does not know the sk corresponding to pk, $\mathcal{B}$ simulates all zero-knowledge proofs involving the witness sk. When depositing, if a coin id $\mathsf{pk}^{\mathsf{sn}}$ disclosed by the honest user with public key pk has already appeared in the global ledger, $\mathcal{B}$ retrieves that deposit request with the same $\mathsf{cn.id}$. Then, from the disclose-correctness proof, $\mathcal{B}$ extracts the witness w corresponding to coin id, which satisfies $\mathsf{cn.id} = h^w$. Finally, $\mathcal{B}$ sends $\mathsf{sk} = w \cdot \mathsf{sn}^{-1} \bmod p$ to its challenger and solves the DLP assumption. Thus, we have

$$\Pr[\mathcal{A} \text{ breaks the offline-checkable uniqueness}] \leq p_u \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathrm{dl}}(\lambda).$$

where p_u is the number of honest users in the system. □

Theorem 2. *PulpoPay running in the* $(\mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{NIZK}})$*-hybrid model UC-realizes the ideal functionality* $\mathcal{F}_{\mathsf{CBDC}}$*.*

Proof of sketch. The formal UC security proof is postponed to Appendix G due to page limitations. Here, we present a proof of sketch, analyzing the achievement of design goals.

Integrity. If the integrity of PulpoPay is broken, there are the following possible attack cases.

1) *Coin Theft.* In PulpoPay, the receiver's identity is embedded in the coin. To deposit a coin belonging to an honest user, $\mathcal{A}$ needs to prove that $\mathcal{A}$'s identity matches the receiver's identity, thus violating the *knowledge soundness* of the corresponding NIZK.
2) *Counterfeiting*. $\mathcal{A}$ creates coins out of thin air. This means $\mathcal{A}$ can generate a valid signature on a ciphertext encrypting new messages without knowing the signing key of any maintainer. Therefore, this case implies the breaking of SoRC's *unforgeability*.
3) *Double-Deposit.* $\mathcal{A}$ attempts to deposit the same coin more than once. This case is impossible to occur. In PulpoPay, each coin is assigned a unique cn.id. Once a coin is deposited, the maintainer records its cn.id in the ledger and rejects any subsequent attempts involving the same cn.id.
4) *Double-Withdrawal.* $\mathcal{A}$ tries to update an account's state multiple times based on the same old state. This case is also impossible to occur. PulpoPay requires disclosing the old state tag AccTag for each update. After the first update, the maintainer records AccTag in the ledger and rejects any repeated usage of that same tag.

Privacy. If the privacy of PulpoPay is broken, there are the following possible attack cases.

1) *Learning Payment Details.* $\mathcal{A}$ can only learn the payment values or payer/payee identities from communication transcripts that imply these data. However, the user encrypts all relevant data (the old/updated account $\mathsf{acc}/\mathsf{acc}^{\mathsf{new}}$, and the coins $\{\mathsf{cn}_{s,i}\}_{i=1}^{n}, \{\mathsf{cn}_{r,j}\}_{j=1}^{m}$) using the ElGamal encryption scheme under the regulators' public keys, and employs NIZK proofs to ensure correctness. The regulators share the corresponding secret keys by employing SSS. Thus, compromising privacy requires breaking *CPA security* of ElGamal, the *zero-knowledge* property of NIZKs, or the *privacy* of SSS.
2) *Linking Multiple Transactions*. $\mathcal{A}$ identifies that multiple transactions are from the same user or two transactions are for the same payment (one for withdrawal and another for deposit). In PulpoPay, each user's account update is based on a randomized old account, and the old account tag AccTag is disclosed. The regulators share the corresponding tracing key by employing SSS. Moreover, the deposited coin is a ciphertext-signature pair randomized from the withdrawn one. Therefore, violating unlinkability would break the *class-hiding* property or the *signature-adaptation* property of SoRC, the *d-strong Diffie-Hellman* assumption, or the *privacy* of SSS.

Auditability. The support of the *limits policy* stems from the *completeness* of NIZKs. The *tracing* and *privacy revocation* are guaranteed by the *correctness* of SSS and SoRC.

High concurrency. With the n-deposit-m-payment protocol, PulpoPay allows a user to process multiple payments concurrently without depending on the states of counterparties. Moreover, Section VII presents empirical results on PulpoPay's ability to handle high concurrency.

Offline confirmation. If $\mathcal{A}$ breaks offline confirmation, it means that more than one user, where at least one of them is honest, requests to deposit the same coin, i.e., the same id. Once this happens, since only one deposit request can succeed for a given id, the honest receiver could no longer confirm completion offline using id. As discussed in Theorem 1, this implies the *offline-checkable uniqueness* is broken. □

VII. Performance Evaluation

A. Experimental Setup

Implementation details. We instantiate the NIZK ideal functionality $\mathcal{F}_{\mathsf{NIZK}}$ as Sigma protocols (see Appendix E). We instantiate the SoRC scheme using the generalized version of the scheme by Bauer and Fuchsbauer [5]. We set the upper bound of the transaction value to be $\mathsf{Max} = 2^{32} - 1$. We implement all protocols in Rust using the arkworks library on BN254 curve. The source code is available at https://anonymous.4open.science/r/PulpoPay-686E. We conduct experiments on two platforms.

- *Common computer for maintainer and regulators*. Algorithms executed by the maintainer and regulators are run on a laptop equipped with an Intel Core Ultra 9 185H processor (16 cores, 2.3GHz) and 32GB of RAM.
- *Mobile device for end-user*. To assess real-world feasibility, end-user algorithms are executed on a mobile device. We use a Samsung Galaxy S23 running Android 14.

Benchmark and comparison settings. We conduct a comprehensive benchmark of all PulpoPay protocols. Among them, the n-deposit-m-payment protocol is the main focus and is compared with the state-of-the-art CBDC scheme, PEReDi [29]. To ensure a fair comparison, we implemented PEReDi using the same elliptic curve, language, and hardware as our PulpoPay protocols, following the original PEReDi evaluation methodology. Also, to isolate the comparison of protocol-level performance, we assume, for both PEReDi and ours, ideal communication channels with negligible communication latency.

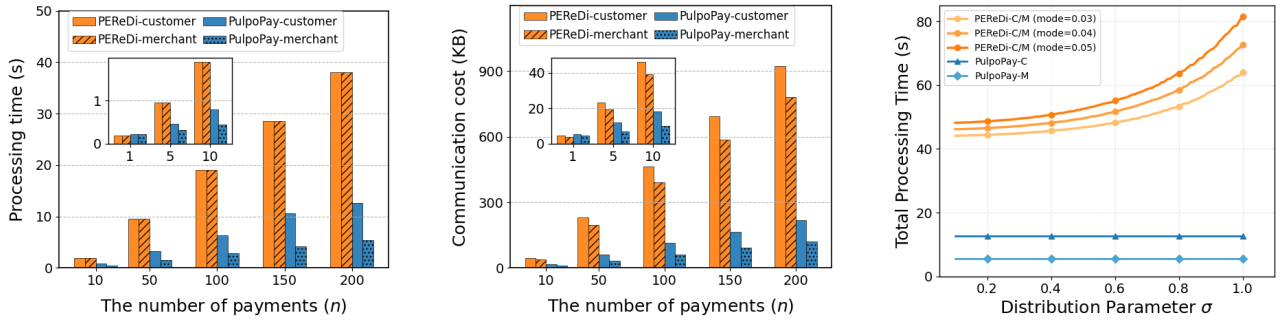
B. Evaluation of n-Deposit-m-Payment

In this section, we evaluate *n*-deposit-*m*-payment on the achievement of offline confirmation and high concurrency.

1) Evaluation of Offline Confirmation: In Section VI, we have proved that PulpoPay achieves offline confirmation securely. To provide empirical evidence, we evaluate the practical efficiency of a single payment involving an offline receiver. Specifically, we measure the processing time and communication costs incurred by three distinct roles: the always-online sender, the offline receiver, and the online-deposit receiver. The result, as shown in Table II, demonstrates the feasibility of PulpoPay for real-time retail and transit use cases.

	Sender	Receiver-offline	Receiver-online
Processing time	0.229 s	0.015 s	0.201 s
Communication cost	5.29 KB	0.34 KB	4.84 KB

TABLE II: The performance of the sender and receiver to fulfill a single payment in PulpoPay.



(a) Processing time comparison between optimistic PEReDi and PulpoPay for the customer's and merchant's concurrent payments.

(b) Communication cost comparison between optimistic PEReDi and PulpoPay for the customer's and merchant's concurrent payments.

(c) Processing time for $n = 200$ in normal PEReDi and PulpoPay under different parameter σ and mode. C: customer; M: merchant.

Fig. 8: The performance comparison of PEReDi and PulpoPay in both optimistic and normal cases.

2) Evaluation of High Concurrency: We evaluate the overhead of a single user processing up to hundreds of concurrent payments to demonstrate the achievement of high concurrency. In what follows, we first explain the evaluation method, and then give the results.

Evaluation method. We evaluate performance by assigning a single user two roles: (1) *customer*, paying n coins; and (2) *merchant*, receiving n coins. The total overhead for a user processing concurrent payments is the sum of costs from both roles. Then, we consider two evaluation cases.

- *Optimistic case.* This case assumes all counterparties cooperate instantly, establishing a performance upper bound. Although less realistic, this remains the sole case adopted in all previous CBDC studies [29], [42].
- *Normal case.* This case provides a more realistic scenario where some counterparties may delay messages.
  - *Modeling.* We adopt the log-normal distribution to model the counterparty delay T_{del}, which is shown in Figure 10. This choice follows the long-standing convention in response time modeling [2], [22], as log-normal distributions effectively capture the compounded delays arising from both request queuing on the counterparty side and the user's retry/backoff mechanisms when the counterparty is unresponsive. We fix the mode (the peak of the probability density function) and enlarge the parameter σ (controlling the spread in the log-space) to indicate less reliable counterparties, leading to a heavier tail. For fair comparison, we also explore other distributions like half-normal or exponential distributions in Appendix H, which demonstrates that the performance gap between PulpoPay and PEReDi remained consistent and distribution-agnostic.
  - *Timeout mechanism.* To prevent infinite delay but guarantee as many successful payments as possible, we set a timeout parameter T_{wt}, e.g., P99 (99th percentile latency) of log-normal distribution in Figure 10. A payment is considered to be *successful* if its $T_{\mathsf{del}} \leq T_{\mathsf{wt}}$; otherwise, once T_{wt} is reached, the payment is revoked and incurs an additional revocation time T_{rvk}.

For PEReDi, n payments involving the same user are processed sequentially, since each payment requires the accounts of both the user and the counterparty to be "locked" until the maintainer sends signatures to both parties (cf. Section I). (1) In the optimistic case, each payment can be completed within a fixed processing time T_{opt} and thus the processing time of the customer/merchant for n payments is $n \cdot T_{\mathsf{opt}}$. Similarly, the communication cost of the customer/merchant for n payments is n times that for each payment. (2) In the normal case, for each payment, the counterparty can delay the account updating request to the maintainer, which may cause the user to receive a delayed signature from the maintainer. In this case, a counterparty delay T_{del} will be added to the optimal processing time T_{opt}. Payments exceeding $T_{\mathsf{opt}} + T_{\mathsf{wt}}$ fail and trigger the revocation protocol.

For PulpoPay, benefiting from the inherent concurrent-friendly design, the evaluation method is relatively much simpler. The user can operate *without depending on the state of any counterparty*, thus fulfilling payments with a guaranteed success rate of 100%. As a customer, the user conducts n concurrent payments with the 0-deposit-n-payment protocol described in Section V-B. As a merchant, the user receives n payments concurrently by confirming the validity of n coins and then executing the n-deposit-0-payment protocol.

Evaluation result. Following the above evaluation methodology, the results are shown in Figure 8.

Optimistic PEReDi vs. PulpoPay. The processing time and the communication cost are shown in Figure 8a and Figure 8b, respectively. When $n = 1$, the customer and merchant overhead in PulpoPay are comparable to those in optimistic PEReDi. As the number of concurrent payments increases, the cost reduction of PulpoPay will become more significant. For example, when $n = 200$, the customer (resp., merchant) in PulpoPay requires 66.89% (resp., 85.66%) less processing time and 76.45% (resp., 84.54%) less communication overhead than the optimistic PEReDi. This efficiency stems from PulpoPay's ability to batch multiple payments into a single account transition, thereby amortizing the overhead of an aggregated zero-knowledge proof across all payments.

Normal PEReDi vs. PulpoPay. We evaluate the total processing time of 200 payments by varying σ from 0.2 to 1 in different

mode cases. Due to the varying counterparty states in reality, we do not attempt to find a true single parameter. Instead, we change σ and mode within a reasonable range to reflect *performance sensitivity on the counterparty uncertainty*. As shown in Figure 8c, in each case with a fixed mode, the processing time of the user in PEReDi is affected by the parameter σ. As σ increases, a larger waiting time is required to achieve 99% success rate, and the overall processing time also increases accordingly. In contrast, PulpoPay maintains constant processing time (12.58 s for the customer and 5.45 s for the merchant) and a 100% success rate. The results demonstrate that while PEReDi's performance is inherently coupled with the counterparty's responsiveness, PulpoPay's performance is independent of the counterparty's state.

Traditional throughput evaluation. While not our primary focus, we also measure throughput as one metric for assessing CBDC concurrency. When considering only valid, independent payments (thereby reflecting PEReDi's optimal throughput scenario), PulpoPay outperforms PEReDi [29], achieving 26 transactions per second (tps) on a 2.3 GHz 16-core laptop, compared to PEReDi's 22 tps. This throughput scales further with more powerful servers or additional shards. For instance, integrating the sharding technique from UTT [40] can scale throughput nearly linearly with the number of shards (e.g., 50 shards can reach at least 1000 tps). Moreover, as discussed in this section, once interdependent transactions and potential payment failures are considered, PulpoPay comprehensively outperforms PEReDi on concurrency.

C. Evaluation of Other Protocols

The performance of the *currency issuance protocol* is very similar to that of the n-deposit-m-payment protocol. Therefore, we skip the evaluation of the currency issuance protocol. In this section, we implement and empirically evaluate the performance of *user registration*, *privacy revocation*, and *tracing* protocols. By measuring both computation and communication costs, we demonstrate their practical feasibility.

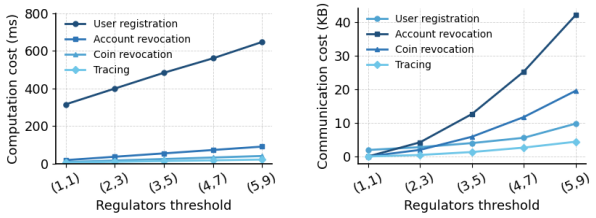


Fig. 9: Performance of user registration, privacy revocation, and tracing.

Evaluation method and results. Unlike the n-deposit-m-payment protocol and the currency issuance protocol, these protocols require regulators' participation. Therefore, we evaluate the performance of these protocols as the threshold (i.e., the number of regulators) increases. The results are shown in Figure 9. The computational overhead of these protocols increases linearly with the number of regulators. The communication cost exhibits a quadratic growth trend relative to the number of regulators. Yet, even with a (5,9) threshold, the computation overhead of these protocols is less than 1 second, and the communication overhead is below 50 KB.

VIII. Further Discussion

Supporting for multiple commercial banks & cross-bank payments. For cross-bank payments, each depositing bank simply obtains the relevant withdrawing bank's SoRC verification key to verify coin deposit validity. Meanwhile, to prevent the same coin from being deposited into two isolated banks, the ledger is maintained by all commercial banks via a consensus protocol (e.g., a BFT scheme).

Distribute the trust on the maintainer. In PulpoPay, we prioritize high concurrency and offline confirmation, assuming the semi-honest maintainer who processes transactions and maintains the ledger. This assumption can be relaxed by distributing tasks among multiple maintainers: each maintainer uploads transactions to a distributed ledger, enabling public verification and regulatory auditing of malicious behavior. Or instead, a threshold version of signatures on randomizable ciphertexts can be introduced among maintainers. As this threshold primitive lies beyond our current scope, it may be explored in future research.

Storage cost and local state management. In PulpoPay, the persistent storage overhead for users primarily stems from the sender maintaining unspent coins and the receiver storing undeposited coins. The size of a single coin is 352 bytes. To minimize this storage cost, users can adopt an on-demand management strategy: senders only withdraw coins from the bank before a transaction, and receivers deposit their collected coins as soon as they regain online connectivity.

Notably, to ensure the security of offline confirmation, the receiver must also maintain a local list of received coin ids to perform stateful deduplication. This list is essential for preventing replay attacks in offline environments where a global ledger is inaccessible. The size of a coin id is 32 bytes. Even for a high-frequency receiver, conducting 10,000 offline payments incurs 320 KB storage cost on modern devices. We emphasize that this stateful maintenance is a mandatory requirement to provide deterministic security guarantees in fully offline scenarios, ensuring that each payment is uniquely processed and verified without central intervention.

Tax payment. Taxation can be seamlessly integrated into our CBDC framework. For instance, the company would pay not only the employee's salary but also the tax calculated at a predetermined rate. The collected taxes are then organized as coins sent to a designated address. The correctness of the tax payment, i.e., the salary indeed incurs the exact amount of tax, can be embedded into our Sigma proof.

Refund procedure. Consider a payment dispute, where the sender pays a coin to the receiver, but the receiver denies to accept this coin. To refund the coin to the sender, there are two methods. For the first method, the receiver deposits the coin into its account and withdraws a coin for the sender via a 1-deposit-1-payment transaction. The main idea of the second method is that each coin is assigned an expiration time at the moment it is withdrawn (much like a credit card

pre-authorization that supports returning of unused funds if a purchase isn't completed in a pre-fixed time). The expiration time can be determined by the regulator's policy based on the transaction type and risk tier (e.g., daily settlement for retail payments). Before the expiration, the receiver can deposit the coin into their own account if they choose to accept it. After the expiration, if the receiver has not yet deposited the coin, the sender is allowed to deposit the coin back into their own account. A secure refund mechanism with an expiration time is left for future work.

Compliance implications of delayed deposits. Delayed deposits may postpone the visibility of taxable income or reportable activities. To this point, PulpoPay does not eliminate this concern but mitigates the delay through expiration and requires online deposit before further spending.

Unique-device assumption. Our unique-device assumption requires a device to maintain a complete, append-only local ID list for its associated account. We agree that offline-checkable uniqueness is guaranteed only under this assumption. Device resets, rollbacks, or unsynchronized multi-device use may violate this assumption and thus require additional system support. We will clarify this limitation and discuss potential safeguards, e.g., secure storage, state backup, and device-specific keys derived from an account master key.

IX. Conclusions

In this work, we present a practical CBDC scheme, named PulpoPay, that achieves high concurrency and offline confirmation while maintaining auditable privacy. We formally prove that PulpoPay is UC-secure. Also, we have provided a prototype implementation. Our evaluation shows that PulpoPay outperforms the state-of-the-art work at both the user and system levels.

Acknowledgment

The authors thank Kaixuan Wang, Weidan Ji, Yanxue Jia, Shi-Feng Sun, and the anonymous NDSS reviewers for their constructive suggestions. This work is supported by the National Science and Technology Major Project (Grant No. 2022ZD0120304) and Shanghai Science and Technology Innovation Action Plan (Grant No. 24BC3200300, 24BC3200500).

References

[1] E. Androulaki, A. De Caro, K. El Khiyaoui, R. Gay, R. Mercer, and A. Sorniotti, "Secure and privacy-preserving cbdc offline payments using a secure element," *Cryptology ePrint Archive*, 2024.

[2] I. Antoniou, V. V. Ivanov, V. V. Ivanov, and P. Zrelov, "On the log-normal distribution of network traffic," *Physica D: Nonlinear Phenomena*, vol. 167, no. 1-2, pp. 72–85, 2002.

[3] R. A. Auer, R. Böhme, J. Clark, and D. Demirag, *Privacy-enhancing technologies for digital payments: mapping the landscape*. Bank for International Settlements, Monetary and Economic Department, 2025.

[4] F. Baldimtsi, M. Chase, G. Fuchsbauer, and M. Kohlweiss, "Anonymous transferable e-cash," in *IACR International Workshop on Public Key Cryptography*. Springer, 2015, pp. 101–124.

[5] B. Bauer and G. Fuchsbauer, "Efficient signatures on randomizable ciphertexts," in *SCN 2020*. Springer, 2020, pp. 359–381.

[6] B. Bauer, G. Fuchsbauer, and C. Qian, "Transferable e-cash: A cleaner model and the first practical instantiation," in *PKC*. Springer, 2021, pp. 559–590.

[7] C. Beer, S. Zingg, K. Kostiainen, K. Wüst, V. Capkun, and S. Capkun, "Payoff: A regulated central bank digital currency with private offline payments," *arXiv 2408.06956*, 2024.

[8] D. Boneh, X. Boyen, and S. Halevi, "Chosen ciphertext secure public key threshold encryption without random oracles," in *Cryptographers' Track at the RSA Conference*. Springer, 2006, pp. 226–243.

[9] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *S&P 2018*. IEEE, 2018, pp. 315–334.

[10] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *Eurocrypt 2005*. Springer, 2005, pp. 302–321.

[11] ——, "Balancing accountability and privacy using e-cash," in *International conference on security and cryptography for networks*. Springer, 2006, pp. 141–155.

[12] S. Canard, D. Pointcheval, O. Sanders, and J. Traoré, "Divisible e-cash made practical," in *PKC*. Springer, 2015, pp. 77–100.

[13] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FoCS*. IEEE, 2001, pp. 136–145.

[14] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *TCC 2007*. Springer, 2007, pp. 61–85.

[15] Capital One Shopping Research. (2025) Amazon marketplace statistics. [Online]. Available: https://capitaloneshopping.com/research/amazon-marketplace-statistics

[16] D. Chaum, "Blind signature system," in *Crypto*, vol. 83. Springer, 1983, p. 153.

[17] D. Chaum, C. Grothoff, and T. Moser, "How to issue a central bank digital currency," *arXiv preprint arXiv:2103.00254*, 2021.

[18] S. Chen, "Progress of research and development of e-cny in china," *Working Group on E-CNY Research and Development, People's Bank of China*, vol. 16, 2021.

[19] Y. Chen, X. Ma, C. Tang, and M. H. Au, "Pgc: Decentralized confidential payment system with auditability," in *ESORICS 2020*. Springer, 2020, pp. 591–610.

[20] A. De Santis, S. Micali, and G. Persiano, "Non-interactive zero-knowledge proof systems," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1987, pp. 52–72.

[21] A. Dogan and K. Bicakci, "Kaime: Central bank digital currency with realistic and modular privacy," *Cryptology ePrint Archive*, 2023.

[22] A. B. Downey, "Lognormal and pareto distributions in the internet," *Comput. Commun.*, vol. 28, no. 7, pp. 790–801, 2005.

[23] B. for International Settlements (BIS), "Central bank digital currencies: foundational principles and core features," *Tech. rep. Bank for International Settlement*, 2020.

[24] J. A. Garay, J. Katz, R. Kumaresan, and H.-S. Zhou, "Adaptively secure broadcast, revisited," in *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, 2011, pp. 179–186.

[25] C. Garman, M. Green, and I. Miers, "Accountable privacy for decentralized anonymous payments," in *FC 2016*. Springer, 2017, pp. 81–98.

[26] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, pp. 51–83, 2007.

[27] C. Gortsos, "Towards a central bank digital currency (cbdc) for the euro area: A primer on the european commission's proposal for a regulation of the eu co-legislators "on the establishment of the digital euro"," *European Banking Institute Working Paper Series*, 2024.

[28] J. Groth, R. Ostrovsky, and A. Sahai, "Perfect non-interactive zero knowledge for np," in *EUROCRYPT 2006*. Springer, 2006, pp. 339–358.

[29] A. Kiayias, M. Kohlweiss, and A. Sarencheh, "Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies," in *ACM CCS*, 2022, pp. 1739–1752.

[30] K. Kurosawa, "Multi-recipient public-key encryption with shortened ciphertext," in *PKC 2002*. Springer, 2002, pp. 48–63.

[31] P. Michalopoulos, A. Mack, C. Clark, L. Chen, J. Sedlmeir, and A. Veneris, "Balancing compliance and privacy in offline cbdc transactions using a secure element-based system," *arXiv preprint arXiv:2509.25469*, 2025.

[32] C. Minwalla, J. Miedema, S. Hernandez, and A. Sutton-Lalani, "A central bank digital currency for offline payments," Bank of Canada, Tech. Rep., 2023.

[33] K. Murphy, "Central Bank Digital Currency Data Use and Privacy Protection," *Fintech Notes*, vol. 2024, no. 004, p. 1, 8 2024. [Online]. Available: http://dx.doi.org/10.5089/9798400286971.063

[34] G. Myre, "How bitcoin has fueled ransomware attacks," 2021, [Online; accessed 2025-10-16]. [Online]. Available: https://www.npr.org/2021/06/10/1004874311/how-bitcoin-has-fueled-ransomware-attacks
[35] D. Pointcheval and O. Sanders, "Short randomizable signatures," in *Cryptographers' Track at the RSA Conference*. Springer, 2016, pp. 111–126.
[36] A. Sarencheh, H. Khoshakhlagh, A. Kavousi, and A. Kiayias, "Dart: Decentralized, anonymous, and regulation-friendly tokenization," *Cryptology ePrint Archive*, 2025.
[37] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 459–474.
[38] A. Shamir, "How to share a secret," *Comm. of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[39] A. Sonnino, M. Al-Bassam, S. Bano, S. Meiklejohn, and G. Danezis, "Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers," *arXiv preprint arXiv:1802.07344*, 2018.
[40] A. Tomescu, A. Bhat, B. Applebaum, I. Abraham, G. Gueta, B. Pinkas, and A. Yanai, "Utt: Decentralized ecash with accountable privacy," *Cryptology ePrint*, 2022.
[41] K. Wüst, K. Kostiainen, V. Čapkun, and S. Čapkun, "Prcash: Fast, private and regulated transactions for digital currencies," in *FC 2019*. Springer, 2019, pp. 158–178.
[42] K. Wüst, K. Kostiainen, N. Delius, and S. Capkun, "Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation," in *ACM CCS*, 2022, pp. 2947–2960.

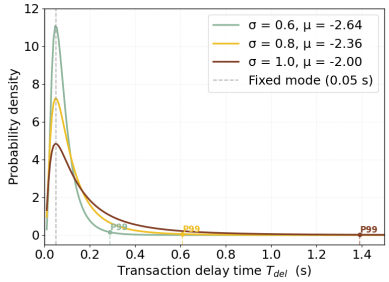


Fig. 10: Fixed-mode Log-normal distribution and the impact of parameter σ. The P99 marker indicates the threshold where the cumulative probability reaches 99%. In other words, there is a 99% probability that a transaction's delay will fall to the left of these points.

APPENDIX A
ADDITIONAL PRELIMINARIES

A. Assumptions

Definition 2. *We say that the d-strong Diffie-Hellman problem is hard relative to $\mathbb{G}$ if for any PPT adversary $\mathcal{A}$ there exists a negligible function such that:* $\mathsf{Adv}_{\mathcal{A}}^{\text{d-sdh}}(\lambda) = |\Pr[\mathcal{A}(h, h^x, \ldots, h^{x^d}) = 1] - \Pr[\mathcal{A}(h, h^{x_1}, \ldots, h^{x_d}) = 1]| \leq \mathsf{negl}(1^\lambda)$, *where* $h \leftarrow\!\!\$\ \mathbb{G}$ *and* $(x, x_1, \ldots, x_d) \leftarrow\!\!\$\ \mathbb{Z}_p^*$.

B. Shamir Secret Sharing

Secret sharing is a method of creating shares of a given secret and later reconstructing the secret itself only if given a threshold number of shares. A secret sharing scheme consists of two algorithms.

- The sharing algorithm $\mathsf{Share}^{D,\alpha}(s)$ takes a secret s and generates shares $(s_1, \ldots, s_D)$.
- The reconstruction algorithm $\mathsf{Agg}^{D,\alpha}(\{s_i\}_{i\in I})$ takes at least α shares as input and outputs the secret s.

A secret sharing scheme should satisfy *correctness* and *privacy*. Correctness guarantees that any α secret-share holders can cooperate to reconstruct the secret s. Privacy requires that *no* information about s is available for less than α shares.

The (D, α)-secret sharing scheme $\mathsf{SSS}^{D,\alpha}$ [38] satisfies the above properties, and the construction is described as follows.

- $\{s_i\}_{i=1}^D \leftarrow \mathsf{SSS.Share}^{D,\alpha}(s)$. Taking the secret $s \in \mathbb{Z}_p$, the distributor randomly selects $\alpha - 1$ integers $a_1, \ldots, a_{\alpha-1}$ from $\mathbb{Z}_p$, sets $a_0 = s$, and defines a $\alpha - 1$ degree polynomial $f(x) = a_0 + a_1 x + \ldots + a_{\alpha-1} x^{\alpha-1} \bmod p$. Then, the distributor sends $s_i = f(i)$ for the i-th holder.
- $s \leftarrow \mathsf{SSS.Agg}^{D,\alpha}(\{s_i\}_{i\in I})$. The Lagrange interpolation formula is used to recover the secret as $s = f(0) = \sum_{i\in I} s_i \lambda_i \bmod p$ where $I \subset [D]$, $|I| = \alpha$, $\lambda_i = \prod_{j\in I, j\neq i} \frac{j}{j-i}$ is the Lagrange coefficient for the i-th share.

C. Threshold Public Key Encryption

In this section, we present a formal definition of chosen plaintext secure (CPA) threshold public key encryption for a static adversary, following the notation from [8]. Then, we describe the construction of threshold ElGamal encryption.

A threshold public key encryption scheme has five algorithms as follows.

- The key generation algorithm $\mathsf{KeyGen}(pp, D, \alpha)$ takes public parameter pp, total number of parties D, and threshold α as input, outputs an verification key vk, an encryption key ek, and decryption key shares $\{j, \mathsf{dk}^{(j)}\}_{j=1}^D$.
- The encryption algorithm $\mathsf{Enc}(\mathsf{ek}, m, r)$ takes an encryption key ek, a message m, and a random factor r as input and outputs a ciphertext ct.
- $\mathsf{ShareDec}(\mathsf{ek}, j, \mathsf{dk}^{(j)}, \mathsf{ct})$ takes the encryption key ek, one of the D decryption key shares $\mathsf{dk}^{(j)}$ ($j \in [D]$), and a ciphertext ct, outputs a decryption share $\mu^{(j)}$ of the encrypted message.
- $\mathsf{ShareVrf}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, \mu^{(j)})$ takes ek, the verification key vk, a ciphertext ct, and a decryption share $\mu^{(j)}$, outputs 1 if the share is valid and 0 otherwise.
- $\mathsf{Combine}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, \{\mu^{(j)}\}_{j\in I})$ takes ek, vk, ct, and α decryption shares $\{\mu^{(j)}\}_{j\in I}$ ($|I| = \alpha$) as input, outputs a message m.

The threshold ElGamal scheme satisfies *correctness*, *IND-CPA*, and *decryption consistency*. Specifically, IND-CPA means that a PPT adversary with less than α number of decryption key shares cannot have any bit information of the message. Decryption consistency means that an adversary cannot produce two sets of valid decryption shares for any ciphertext such that their corresponding plaintexts differ (when applying the combine algorithm).

Correctness. Let $(\mathsf{vk}, \mathsf{ek}, \{j, \mathsf{dk}^{(j)}\}_{j=1}^D)$ be the output of $\mathsf{KeyGen}(pp, D, \alpha)$. We require the following two properties:

- For any ciphertext ct, if $\mu^{(j)} = \mathsf{ShareDec}(\mathsf{ek}, j, \mathsf{dk}^{(j)}, \mathsf{ct})$ where $\mathsf{dk}^{(j)}$ is the j-th private key share, then $\mathsf{ShareVrf}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, \mu^{(j)})$ outputs 1.
- If ct is the output of $\mathsf{Enc}(\mathsf{ek}, m, r)$ and $S = \{\mu^{(j)}\}_{i\in[I]}$ is a set of decryption shares for α distinct decryption keys, then we require that $\mathsf{Combine}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, S) = m$.

Security. Security of threshold public key encryption consists of IND-CPA security and decryption consistency.

Definition 3. *Let IND-CPA be as defined in Figure 11. A threshold public key encryption is IND-CPA secure if for all PPT adversary $\mathcal{A}$, the following function is negligible in λ:*

$$\left|\Pr[\textit{IND-CPA}^{\mathcal{A}}_{\textit{TPKE}}(\lambda,1)=1]-\Pr[\textit{IND-CPA}^{\mathcal{A}}_{\textit{TPKE}}(\lambda,0)=1]\right|.$$

$\text{IND-CPA}^{\mathcal{A}}_{\text{TPKE}}(\lambda, b)$

1 : $S \leftarrow \mathcal{A}(\lambda)$, where $S \subset [D], |S| = \alpha - 1$
2 : $(\mathsf{vk}, \mathsf{ek}, \{j, \mathsf{dk}^{(j)}\}_{j=1}^{D}) \leftarrow \mathsf{KeyGen}(pp, D, \alpha)$
3 : $(m_0, m_1, st) \leftarrow \mathcal{A}(\mathsf{ek}, \mathsf{vk}, \{j, \mathsf{dk}^{(j)}\}_{j\in S})$
4 : $r \leftarrow\!\$\ \mathcal{R}$
5 : $c_b = \mathsf{Enc}(\mathsf{ek}, m_b, r)$
6 : $b' \leftarrow \mathcal{A}(st, c_b)$
7 : **return** b'

Fig. 11: IND-CPA experiment.

Definition 4. *Let DEC-CON be as defined in Figure 12. A threshold public key encryption is decryption consistent if for all PPT adversary $\mathcal{A}$, the probability $\Pr[\textit{DEC-CON}^{\mathcal{A}}_{\textit{TPKE}}=1]$ is negligible in λ.*

$\text{DEC-CON}^{\mathcal{A}}_{\text{TPKE}}(\lambda)$

1 : $S \leftarrow \mathcal{A}(\lambda)$, where $S \subset [D], |S| = \alpha - 1$
2 : $(\mathsf{vk}, \mathsf{ek}, \{j, \mathsf{dk}^{(j)}\}_{j=1}^{D}) \leftarrow \mathsf{KeyGen}(pp, D, \alpha)$
3 : $(\mathsf{ct}, \{\mu^{(j)}\}_{j\in I}, \{(\mu')^{(j)}\}_{j\in I'}) \leftarrow \mathcal{A}(\mathsf{ek}, \mathsf{vk}, \{j, \mathsf{dk}^{(j)}\}_{j\in S})$
I Here, the size of decryption shares satisfies $|I| = |I'| = \alpha$
4 : $b_0 = \wedge_{j\in I}\mathsf{ShareVrf}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, u^{(j)})$
5 : $b_1 = \wedge_{j\in I'}\mathsf{ShareVrf}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, (u')^{(j)})$
6 : $m = \mathsf{Combine}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, \{\mu^{(j)}\}_{j\in I})$
7 : $m' = \mathsf{Combine}(\mathsf{ek}, \mathsf{vk}, \mathsf{ct}, \{(\mu')^{(j)}\}_{j\in I'})$
8 : $b_2 = (m \neq m')$
9 : **return** $b_0 \wedge b_1 \wedge b_2$

Fig. 12: Decryption consistency experiment.

Using a distributed key generation protocol introduced in [26], the (α, D)-threshold ElGamal encryption scheme on vectors of group elements [30] is utilized in our paper, which satisfies the above properties.

- $(\mathsf{ek}, \{(\mathsf{ek}^{(j)}, \mathsf{dk}^{(j)})\}_{j=1}^{D}) \leftarrow \mathsf{KeyGen}(pp, D, \alpha)$. The j-th holder (i.e., the regulator R_j in our paper) derives the decryption key share $\mathsf{dk}^{(j)}$, and the encryption key shares $\mathsf{ek}^{(j)} = g^{\mathsf{dk}^{(j)}}$. Moreover, each regulator can learn the encryption shares of others and the reconstructed encryption key ek.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\{\mathsf{ek}_i, m_i\}_{i=1}^{v})$. To encrypt messages $\vec{m} \in \mathbb{G}^v$, the encryptor (i.e., the user) chooses $r \leftarrow\!\$\ \mathbb{Z}_p^*$ and computes the ciphertext $\mathsf{ct} = (g^r, \{\mathsf{ek}_i^r m_i\}_{i=1}^{v}) \in \mathbb{G}^{v+1}$. Specifically, for message $m_i \in \mathbb{Z}$, the user computes $h^{m_i} \in \mathbb{G}$ and then encrypts it.
- $\mu^{(j)} \leftarrow \mathsf{ShareDec}(\{\mathsf{ek}_i\}_{i=1}^{v}, j, \{\mathsf{dk}_i^{(j)}\}_{i=1}^{v}, \mathsf{ct})$. One of the decryptors (i.e., the j-th regulator) executes this algorithm. To decrypt the ciphertext $\mathsf{ct} = (\phi_1, \ldots, \phi_{v+1})$, the j-th regulator publishes $\{\hat{\mu}_i^{(j)} = \phi_1^{\mathsf{dk}_i^{(j)}}\}_{i=1}^{v}$ and generates π_j proving that $\{\log_g \mathsf{ek}_i^{(j)} = \log_{\phi_1} \hat{\mu}_i^{(j)}\}_{i=1}^{v}$. Here, $\mu^{(j)}$ denotes $(\{\hat{\mu}_i^{(j)}\}_{i=1}^{v}, \pi_j)$.
- $0/1 \leftarrow \mathsf{ShareVrf}(\{\mathsf{ek}\}_{i=1}^{v}, \{\mathsf{ek}^{(j)}\}_{i=1}^{v}\mathsf{ct}, \mu^{(j)})$. One of the decryptors (i.e., the j-th regulator) executes this algorithm. The j-th regulator parses $\mu^{(j)} = (\{\hat{\mu}_i^{(j)}\}_{i=1}^{v}, \pi_j)$, $\mathsf{ct} = (\phi_1, \ldots, \phi_{v+1})$. It checks if $\{\log_g \mathsf{ek}_i^{(j)} = \log_{\phi_1} \hat{\mu}_i^{(j)}\}_{i=1}^{v}$ by verifying the validity of π_j. If π_j is valid, it outputs 1. Otherwise, it outputs 0.
- $\{m_i\}_{i=1}^{k} \leftarrow \mathsf{Combine}(\mathsf{ct}, \{\mu^{(j)}\}_{j\in I})$. Any regulator can execute this algorithm as a combiner. It parses $\mu^{(j)} = (\{\hat{\mu}_i^{(j)}\}_{i=1}^{v}, \pi_j)$, $\mathsf{ct} = (\phi_1, \ldots, \phi_{v+1})$. The messages $\{m_i\}_{i=1}^{k}$ can be computes as $m_i = \phi_{i+1} / \prod_{j\in[I]} (\hat{\mu}_i^{(j)})^{\lambda_j}$ where λ_j is the Lagrange coefficient for the j-th share.

D. Signatures on Randomizable Ciphertexts

In this section, we present the definition of signatures on randomizable ciphertexts and an efficient construction, referring to [5].

Definition 5. *A SoRC scheme is correct if for all public parameters, for all pairs ek and $(\hat{\mathsf{sk}}, \hat{\mathsf{vk}})$ in the range of $\mathsf{KeyGen}(pp, D, \alpha)$ and $\mathsf{SKeyGen}(pp)$, respectively, and all $\mathsf{ct} \in \mathcal{C}$:*

$$\mathsf{Vf}(\hat{\mathsf{vk}}, \mathsf{ek}, \mathsf{ct}, \mathsf{Sign}(\hat{\mathsf{sk}}, \mathsf{ek}, \mathsf{ct})) = 1$$

Definition 6. *Let CL-HID be as defined in Figure 13. A SoRC scheme is fully class-hiding if for all PPT adversary $\mathcal{A}$, the following function is negligible in λ:*

$$\left|\Pr[\textit{CL-HID}^{\mathcal{A}}_{\mathsf{SoRC}}(\lambda,1)=1]-\Pr[\textit{CL-HID}^{\mathcal{A}}_{\mathsf{SoRC}}(\lambda,0)=1]\right|.$$

$\text{CL-HID}^{\mathcal{A}}_{\mathsf{SoRC}}(\lambda, b)$

1 : $S \leftarrow \mathcal{A}(\lambda)$, where $S \subset [D], |S| = \alpha - 1$
2 : $(\mathsf{vk}, \mathsf{ek}, \{j, \mathsf{dk}^{(j)}\}_{j=1}^{D}) \leftarrow \mathsf{KeyGen}(pp, D, \alpha)$
3 : $(\mathsf{ct}, st) \leftarrow \mathcal{A}(\mathsf{ek}, \mathsf{vk}, \{j, \mathsf{dk}^{(j)}\}_{j\in S})$
4 : $\mathsf{ct}_0 \leftarrow\!\$\ \mathcal{C}$
5 : $r \leftarrow\!\$\ \mathcal{R}; (\mathsf{ct}_1, \perp) = \mathsf{Rdm}(\mathsf{ek}, \mathsf{ct}, \perp, r)$
6 : $b' \leftarrow \mathcal{A}(st, \mathsf{ct}_b)$
7 : **return** b'

Fig. 13: CL-HID experiment.

Definition 7. *A SoRC scheme is signature-adaptable under malicious keys if for all public parameters, all $(\hat{\mathsf{vk}}, \hat{\mathsf{sk}}, \mathsf{ct}, \hat{\sigma})$ satisfying $\mathsf{Vf}(\hat{\mathsf{vk}}, \mathsf{ek}, \mathsf{ct}, \hat{\sigma}) = 1$ and all $r \in \mathcal{R}$, the signature in the output $(\hat{\sigma}^{\mathsf{rnd}}, \mathsf{ct}^{\mathsf{rnd}})$ of $\mathsf{Rdm}(\vec{\mathsf{ek}}, \mathsf{ct}, \hat{\sigma}, r)$ is uniformly distributed over the set $\{\hat{\sigma}' | \mathsf{Vf}(\hat{\mathsf{vk}}, \mathsf{ek}, \mathsf{ct}^{\mathsf{rnd}}) = 1\}$.*

Together, full class-hiding and signature-adaptability under malicious keys imply that for an adversary that creates a signature verification key as well as a ciphertext and a signature on it, a randomization of this ciphertext together with an adapted signature looks like a random ciphertext with a fresh signature on it.

Definition 8. *Let EUF be as defined in Figure 14. A SoRC scheme is unforgeable if for all PPT adversary $\mathcal{A}$ the probability* $\Pr[EUF^{\mathcal{A}}_{\mathsf{SoRC}} = 1]$ *is negligible in* λ.

$\mathrm{EUF}^{\mathcal{A}}_{\mathsf{SoRC}}(\lambda)$

$Q = \emptyset$

$(\hat{\mathsf{sk}}, \hat{\mathsf{vk}}) \leftarrow\$ \mathsf{SKeyGen}(pp)$

$((\mathsf{ek}^*, \mathsf{ct}^*), \hat{\sigma}^*) \leftarrow \mathcal{A}^{\mathrm{OSign}(\hat{\mathsf{sk}}, \cdot, \cdot)}(\hat{\mathsf{vk}})$

return $(\mathsf{Vf}(\hat{\mathsf{vk}}, \vec{\mathsf{ek}}^*, c^*, sig^*) = 1 \wedge (\mathsf{ek}^*, \mathsf{ct}^*) \notin Q)$

$\mathrm{OSign}(\hat{\mathsf{sk}}, \mathsf{ek}, \mathsf{ct})$

$Q = Q \cup \{\mathsf{ek}\} \times [\mathsf{ct}]_{\mathsf{ek}}$

return $\mathsf{Sign}(\hat{\mathsf{sk}}, \mathsf{ek}, \mathsf{ct})$

Fig. 14: Unforgeability experiment. Here, $[c]_{\mathsf{ek}}$ are ciphertexts encrypting the same m under ek.

The signatures on the randomizable ciphertexts scheme SoRC [5] is a signature scheme w.r.t the ElGamal encryption scheme on vectors of group elements. Inheriting (KeyGen, Enc, ShareDec, ShareVrf, Combine) from the threshold ElGamal encryption, SoRC has another four algorithms.

- $(\hat{\mathsf{sk}}, \hat{\mathsf{vk}}) \leftarrow \mathsf{SKeygen}(pp)$. On input public parameters pp, set $\hat{\mathsf{sk}} = (\{x_i\}_{i=1}^{v+1}) \leftarrow\$ (\mathbb{Z}_p^*)^{v+1}$ and compute $\hat{\mathsf{vk}} = (\hat{X}_i := \hat{g}^{x_i})_{i=1}^{v+1} \in \hat{\mathbb{G}}^{v+1}$.
- $\hat{\sigma} \leftarrow \mathsf{Sign}(\hat{\mathsf{sk}}, \vec{\mathsf{ek}}, \mathsf{ct})$. Parse ct as $(\phi_1, \ldots, \phi_{v+1})$, $\vec{\mathsf{ek}}$ as $(\mathsf{ek}_1, \ldots, \mathsf{ek}_v)$. The signer chooses $s \leftarrow\$ \mathbb{Z}_p^*$ and computes

$$Z = (g \prod_{i=1}^{v+1} \phi_i^{x_i})^{s^{-1}}, S = g^s, \hat{S} = \hat{g}^s, T = (g^{x_1} \prod_{i=1}^{v} \mathsf{ek}_i^{x_{i+1}})^{s^{-1}}.$$

  Finally, the signer outputs $\hat{\sigma} = (Z, S, \hat{S}, T)$.
- $0/1 \leftarrow \mathsf{Vf}(\hat{\mathsf{vk}}, \vec{\mathsf{ek}}, \mathsf{ct}, \hat{\sigma})$. This algorithm outputs 1 if the following equations hold and 0 otherwise:

$$e(Z, \hat{S}) = e(G, \hat{G}) \prod_{i=1}^{v+1} e(\phi_i, \hat{X}_i), e(g, \hat{S}) = e(S, \hat{g}),$$

$$e(T, S) = e(g, \hat{X}_0) \prod_{i=1}^{v} e(\mathsf{ek}_i, \hat{X}_{i+1}).$$

- $(\mathsf{ct}^{\mathsf{rnd}}, \hat{\sigma}^{\mathsf{rnd}}) \leftarrow \mathsf{Rdm}(\vec{\mathsf{ek}}, \mathsf{ct}, \hat{\sigma}, r^{\mathsf{rnd}})$. Parse $\mathsf{ct} := (\phi_1, \ldots, \phi_{v+1})$, $\hat{\sigma} := (Z, S, \hat{S}, T)$. This algorithm computes $\mathsf{ct}^{\mathsf{rnd}} = (\phi_1 g^{r^{\mathsf{rnd}}}, \phi_2 \mathsf{ek}_1^{r^{\mathsf{rnd}}}, \ldots, \phi_{v+1} \mathsf{ek}_v^{r^{\mathsf{rnd}}})$. Then, choose $s' \leftarrow\$ \mathbb{Z}_p^*$ and compute

$$Z' = (ZT^{r^{\mathsf{rnd}}})^{(s')^{-1}}, S' = S^{s'}, \hat{S}' = \hat{S}^{s'}, T' = T^{(s')^{-1}}.$$

  Finally, it outputs $\mathsf{ct}^{\mathsf{rnd}}$ and $\hat{\sigma}^{\mathsf{rnd}} = (Z', S', \hat{S}', T')$.

APPENDIX B
USER REGISTRATION PROTOCOL

Receiving $(\mathrm{GenAcc}, sid, \mathsf{U}, B_{\max}, S_{\max}, R_{\max})$ from $\mathcal{Z}$, the user U:

1. Sets $\mathsf{acc} = (0, B_{\max}, 0, S_{\max}, 0, R_{\max}, \mathsf{pk}_{\mathsf{U}}, h^a, a)$ where $a \leftarrow\$ \mathbb{Z}_p^*$ and computes $\mathsf{acc}^{\mathcal{B}} \leftarrow \mathsf{SoRC.Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{acc}, r_{\mathsf{acc}})$.
2. Executes $\{a_j\}_{j=1}^{D} \leftarrow \mathsf{SSS.Share}^{D,\alpha}(a)$ to share a, computes $\mathsf{com}_j = g_1^{a_j} g_2^{r_j}$ for $r_j \leftarrow\$ \mathbb{Z}_p^*$, and sets $\mathsf{com}_{\mathsf{R}} = \{\mathsf{com}_j\}_{j=1}^{D}$.
3. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathrm{prove}, sid, x, w)$ and receives $(\mathrm{proof}, sid, \pi_{\mathrm{trace}})$. Here, $x = (\mathsf{acc}^{\mathcal{B}}, \mathsf{com}_{\mathsf{R}})$ and witness $w = (r_{\mathsf{acc}}, a, \{a_j\}_{j=1}^{D}, \{r_j\}_{j=1}^{D})$ satisfying that the trace key a encrypted by $\mathsf{acc}^{\mathcal{B}}$ is the same as the element reconstructed from $\{a_j\}_{j=1}^{D}$ committed in $\mathsf{com}_{\mathsf{R}}$.
4. Packages the tracing information $\mathsf{TRI}_j = (\mathsf{acc}^{\mathcal{B}}, a_j, r_j, \mathsf{com}_{\mathsf{R}}, \pi_{\mathrm{trace}})$, calls $\mathcal{F}_{\mathsf{BC}}$ with $(\mathrm{Broadcast}, sid, \mathsf{com}_{\mathsf{R}})$, and calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$ for $j \in [D]$ with input $(\mathrm{Send}, sid, \mathsf{R}_j, \mathsf{TRI}_j)$.

Receiving $(\mathrm{Receive}, sid, \mathsf{U}, \mathsf{TRI}_j)$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$, $(\mathrm{Bdcasted}, sid, \mathsf{U}', \mathsf{com}'_{\mathsf{R}})$ from $\mathcal{F}_{\mathsf{BC}}$, the regulator R_j:

1. Parses TRI_j as $(\mathsf{acc}^{\mathcal{B}}, a_j, r_j, \mathsf{com}_{\mathsf{R}}, \pi_{\mathrm{trace}})$. Ignores if calls $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathrm{Verify}, sid, (\mathsf{acc}^{\mathcal{B}}, \mathsf{com}_{\mathsf{R}}), \pi_{\mathrm{trace}})$ but receives $(\mathrm{Verification}, sid, 0)$.
2. Asserts $\mathsf{com}_j = g_1^{a_j} g_2^{r_j}$, $\mathsf{U}' = \mathsf{U}$, and $\mathsf{com}'_{\mathsf{R}} = \mathsf{com}_{\mathsf{R}}$.
3. Records $(\mathsf{U}, \mathsf{acc}^{\mathcal{B}}, a_j)$ in list $\mathcal{L}_{\mathsf{R}_j}$ and calls the authenticated channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with input $(\mathrm{Send}, sid, \mathsf{U}, ok)$.

Receiving up to α messages $(\mathrm{Receive}, sid, \mathsf{R}_j, ok)$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, the user U:

1. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathrm{prove}, sid, x, w)$ and receives $(\mathrm{proof}, sid, \pi_{\mathrm{reg}})$. Here, $x = (\mathsf{pk}_{\mathsf{U}}, \mathsf{acc}^{\mathcal{B}}, B_{\max}, S_{\max}, R_{\max})$ and witness $w = (\mathsf{sk}_{\mathsf{U}}, a, r_{\mathsf{acc}})$ satisfying
   - *Identity correctness:* $\mathsf{pk}_{\mathsf{U}} = h^{\mathsf{sk}_{\mathsf{U}}}$;
   - *Account-format correctness:* $\mathsf{acc}^{\mathcal{B}}$ is a well-formed ciphertext encrypting $(0, B_{\max}, 0, S_{\max}, 0, R_{\max}, \mathsf{pk}_{\mathsf{U}}, h^a, a)$.
2. Packages the registration information $\mathsf{RI} = (\mathsf{pk}_{\mathsf{U}}, \mathsf{acc}^{\mathcal{B}}, B_{\max}, S_{\max}, R_{\max}, \pi_{\mathrm{reg}})$ and calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with input $(\mathrm{Send}, sid, \mathsf{M}^*, \mathsf{RI})$.

Receiving $(\mathrm{Receive}, sid, \mathsf{U}, \mathsf{RI})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, the maintainer M^*:

1. Parses RI as $(x = (\mathsf{pk}_{\mathsf{U}}, \mathsf{acc}^{\mathcal{B}}, B_{\max}, S_{\max}, R_{\max}), \pi_{\mathrm{reg}})$. Ignores the message if upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathrm{Verify}, sid, x, \pi_{\mathrm{reg}})$, it receives $(\mathrm{Verification}, sid, 0)$.
2. Calls the authenticated channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ for $j \in [D]$ with input $(\mathrm{Send}, sid, \mathsf{R}_j, \mathsf{acc}^{\mathcal{B}})$ to confirm that up to α regulators have recorded $\mathsf{acc}^{\mathcal{B}}$ and tracing key shares.
3. Computes $\hat{\sigma}_{\mathsf{acc}} \leftarrow \mathsf{SoRC.Sign}(\hat{\mathsf{sk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{acc}^{\mathcal{B}})$ and appends RI to the ledger $\mathcal{L}$.
4. Calls the authenticated channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with input $(\mathrm{Send}, sid, \mathsf{U}, \hat{\sigma}_{\mathsf{acc}})$.

Receiving $(\mathrm{Receive}, sid, \mathsf{M}^*, \hat{\sigma}_{\mathsf{acc}})$, the user U:

1. Ignores the message if $\mathsf{SoRC.Vf}(\hat{\mathsf{vk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{acc}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{acc}}) = 0$.
2. Computes $(\mathsf{acc}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{acc}}) \leftarrow \mathsf{SoRC.Rdm}(\vec{\mathsf{ek}}_{\mathsf{acc}}, \mathsf{acc}^{\mathcal{B}}, \hat{\sigma}_{\mathsf{acc}}, r'_{\mathsf{acc}})$ and sets $r_{\mathsf{acc}} = r_{\mathsf{acc}} + r'_{\mathsf{acc}}$ for subsequent account updates.
3. Outputs $(\mathrm{AccGened}, sid, \mathsf{U})$ (to $\mathcal{Z}$).

Fig. 15: The procedure of the user registration protocol.

APPENDIX C
CURRENCY ISSUANCE PROTOCOL

In this section, we explain the currency issuance protocol, which enables the central bank to concurrently issue money to multiple receivers. The detailed procedure is shown in Figure 16. We roughly describe the operations of the central bank, the maintainer, and the receivers as follows.

For the central bank, to issue money for m receivers, it generates m coins, i.e., $\{\mathsf{cn}_{r,j} = (v_{r,j}, \mathsf{pk}_{\mathsf{B}}, \mathsf{pk}_{r,j}, \mathsf{id}_{r,j})\}_{j=1}^{m}$. Here, $v_{r,j}, \mathsf{pk}_{\mathsf{B}}, \mathsf{pk}_{r,j}$ are the coin value, the public key of the

central bank, and the public key of the receiver, respectively. $\mathsf{id}_{r,j}$ is a unique tag of the coin designed for double-spending prevention. The central bank encrypts these new coins under $\vec{\mathsf{ek}}_{\mathsf{cn}}$. Thereafter, the central bank sends the encrypted coins to the maintainer M^*.

For the maintainer M^*, after receiving the issuance request from the central bank, it verifies the validity of these encrypted coins and generates signatures on them. Then, the maintainer M^* appends the issuance transaction to the public ledger and replies to the central bank with these signatures.

For the central bank, after receiving these signatures from the maintainer M^*, the operations and the interactions with receivers are the same as a sender-receiver pair in the n-deposit-m-payment protocol. Therefore, we postpone the following explanation to the n-deposit-m-payment protocol.

Receiving $(\text{Issue}, sid, \{\mathsf{U}_{r,j}, v_{r,j}\}_{j=1}^m)$ from $\mathcal{Z}$, B:

1. For each $j \in [m]$, sets $\mathsf{cn}_{r,j} = (v_{r,j}, \mathsf{pk}_{\mathsf{B}}, \mathsf{pk}_{r,j}, \mathsf{id}_{r,j})$ where $\mathsf{id}_{r,j} = \mathsf{pk}_{r,j}^{\mathsf{sn}_{r,j}}, \mathsf{sn}_{r,j} \leftarrow\$ \mathbb{Z}_p^*$ and computes $\mathsf{cn}_{r,j}^{\mathcal{B}} \leftarrow \mathsf{SoRC.Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{cn}_{r,j}, r_{\mathsf{cn}_{r,j}})$.
2. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\text{prove}, sid, x, w)$ and receives $(\text{proof}, sid, \pi_{\text{iss}})$. Here, a NIZK is a zero-knowledge proof for statement $x = (\vec{\mathsf{ek}}_{\mathsf{cn}}, \{\mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^m)$ and witness $w = (\{v_{r,j}, \mathsf{sk}_{\mathsf{B}}, r_{\mathsf{cn}_{r,j}}\}_{j=1}^m)$ satisfying
   - *Coin-format correctness:* For $j \in [m]$, $\mathsf{cn}_{r,j}^{\mathcal{B}}$ is a well-formed ciphertext encrypting $v_{r,j}$ and pk_{B}. And it knows sk_{U} corresponding to pk_{U}.
   - *Range proof:* For $j \in [m]$, $v_{r,j} \in [0, \mathsf{Max}]$.
3. Packages transaction information $\mathsf{TI}_{\mathsf{B}} = (x, \pi_{\text{iss}})$ and calls the authenticated channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with $(\text{Send}, sid, \mathsf{M}^*, \mathsf{TI}_{\mathsf{B}})$.

Receiving $(\text{Receive}, sid, \mathsf{TI}_{\mathsf{B}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, the maintainer M^*:

1. Parses TI_{B} as (x, π_{iss}).
2. Ignores this message if receives $(\text{Verification}, sid, 0)$ upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\text{Verify}, sid, x, \pi_{\text{iss}})$,.
3. Computes $\hat{\sigma}_{\mathsf{cn}_{r,j}} \leftarrow \mathsf{SoRC.Sign}(\hat{\mathsf{sk}}_{\mathsf{M}^*}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{cn}_{r,j}^{\mathcal{B}})$ and appends TI_{B} into the public ledger $\mathcal{L}$.
4. Calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with $(\text{Send}, sid, \mathsf{B}, \{\hat{\sigma}_{\mathsf{cn}_{r,j}}\}_{j=1}^m)$.

The following procedures are identical to those in the n-deposit-m-payment protocol.

Fig. 16: The procedure of currency issuance protocol.

APPENDIX D
AUDITING PROTOCOL

User tracing. Recall that a tracing tag $\mathsf{AccTag} = h^{a^x}$ should be disclosed when updates an account, where a is a secret key for tracing and x is a counter for updating the account. To link coins involving a target user without revealing a, regulators use the secret shares of a to compute all related tags.

Receiving $(\text{Revoke}, sid, \mathsf{U}_j)$ from $\mathcal{Z}$, the j-th regulator R_j:

1. Finds $(\mathsf{U}_j, a_j, r_j, \mathsf{com}_{\mathsf{R}})$ saved in $\mathcal{L}_{\mathsf{R}_j}$ and sets $\tilde{h} = h$.
2. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\text{prove}, sid, x_j, w_j)$ and receives $(\text{proof}, sid, \pi_j)$. Here, NIZK is an argument of knowledge for statement $x_j = (\mathsf{com}_j, \tilde{h}^{a_j}, \tilde{h})$ and $w_j = (a_j, r_j)$ satisfying that com_j is the commitment to the same a_j as $\tilde{h}^{a_j}$.
3. Calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with the input $(\text{Send}, sid, \mathsf{R}_j, (x_j, \pi_j))$ for all $i \neq j$.
4. Receiving α valid tracing shares, i.e., $\tilde{h}^{a_i}$, $i \in I$, $I \subset [D]$, and $|I| = \alpha$, computes $\tilde{h}^a = \prod_{j \in I} (\tilde{h}^{a_j})^{\prod_{i \in I, i \neq j} (i/(i-j))}$.
5. Reads the ledger $\mathcal{L}$ to find the transactions including $\mathsf{AccTag} = \tilde{h}^a$ and records the associated $\{\mathsf{cn}^{\mathcal{B}}\}$ and the role of U_j.
6. Proceeds from step 2 with $\tilde{h} = \tilde{h}^a$ until no transaction includes the computed AccTag.
7. Outputs $(\text{Revoked}, sid, \{(\mathsf{cn}^{\mathcal{B}}, \mathsf{snd/rec})\})$ to $\mathcal{Z}$.

Fig. 17: The procedure of the user tracing protocol.

Privacy Revocation. In our system, all private information, including transactions and account information, is consistently hidden by some ciphertexts, i.e., $\mathsf{cn}^{\mathcal{B}}$ and $\mathsf{acc}^{\mathcal{B}}$. Taking advantage of threshold decryption, distributed regulators can cooperate to decrypt private information. For example, the coin revocation procedure is shown as follows. The account revocation is similar.

- Upon receiving $(\text{Revoke}, sid, \mathsf{cn}^{\mathcal{B}})$ (from $\mathcal{Z}$), each regulator R_j executes $\mathsf{ShareDec}(j, \{\mathsf{ek}_i, \mathsf{dk}_{\mathsf{R}_j,i}\}_{i \in \{10,11,12,13\}}, \mathsf{cn}^{\mathcal{B}})$ to derive the decryption share $\mu^{(j)}$ and calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with the input $(\text{Send}, sid, \mathsf{R}_i, \mu^{(j)})$ for all $i \neq j$.
- Receiving decryption shares from the other regulators (e.g., R_j), each regulator verifies the validity of the decryption share by executing $\mathsf{ShareVrf}(\{\mathsf{ek}_i, \mathsf{ek}_{\mathsf{R}_j,i}\}_{i \in \{10,11,12,13\}}, \mathsf{cn}^{\mathcal{B}}, \mu^{(j)})$.
- Then, each regulator can execute $\mathsf{Combine}(\mathsf{cn}^{\mathcal{B}}, \{\mu^{(j)}\}_{j \in I})$ to derive $(h^v, \mathsf{pk}_s, \mathsf{pk}_r, \mathsf{id})$. Here, $I \subset [D]$ and $|I| = \alpha$. The value v can be computed from h^v with a small space.

APPENDIX E
IMPLEMENTATION OF ZERO-KNOWLEDGE PROOFS

NIZK in *n*-deposit-*m*-payment. We give the formal definition of the relation $\mathsf{R}_{\text{trans}}$ involved in n-deposit-m-payment protocol and the corresponding zero-knowledge proof to generate π_{trans}. To facilitate a clear understanding of the zero-knowledge proof, we summarize the correspondence among the encryption keys, the account/coin attributes, and the account/coin ciphertexts in Figure 18. The relation $\mathsf{R}_{\text{trans}}$ can be decomposed as $\mathsf{R}_{\text{format}}$, $\mathsf{R}_{\text{disclose}}$, $\mathsf{R}_{\text{update}}$, $\mathsf{R}_{\text{identity}}$, $\mathsf{R}_{\text{range}}$:

$$\left\{ \begin{array}{c} (\vec{\mathsf{ek}}_{\mathsf{cn}}, \{\mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^m; \{v_{r,j}, r_{\mathsf{cn}_{r,j}}\}_{j=1}^m) : \\ \forall j \in [m], \mathsf{cn}_{r,j}^{\mathcal{B}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, (v_{r,j}, \cdot, \cdot, \cdot), r_{\mathsf{cn}_{r,j}}) \end{array} \right\}$$

$$\left\{ \begin{array}{c} (\vec{\mathsf{ek}}_{\mathsf{acc}}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{AccTag}, \mathsf{acc}^{\mathcal{B}}, \{\mathsf{id}_{s,i}, \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}\}_{i=1}^n; \\ \{\mathsf{sk}_{\mathsf{U}} \cdot \mathsf{sn}_{s,i}, r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}\}_{i=1}^n, r_{\mathsf{acc}}) : \\ \mathsf{acc}^{\mathcal{B}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, (\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{AccTag}, \cdot), r_{\mathsf{acc}}), \\ \forall i \in [n], \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, (\cdot, \cdot, \cdot, \mathsf{id}_{s,i}), r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}), \\ \mathsf{id}_{s,i} = h^{\mathsf{sk}_{\mathsf{U}} \cdot \mathsf{sn}_{s,i}} \end{array} \right\}$$

$$\left\{ \begin{array}{c} (\mathsf{acc}^{\mathcal{B}}, \mathsf{acc}^{\mathcal{B},\mathsf{new}}, \prod_{i=1}^n \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}, \prod_{j=1}^m \mathsf{cn}_{r,j}^{\mathcal{B}}, \mathsf{AccTag}; \\ r_{\mathsf{acc}}, r_{\mathsf{acc}}^{\mathsf{new}}, \sum_{i=1}^n v_{s,i}, \sum_{i=1}^n r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}, \sum_{j=1}^m v_{r,j}, \sum_{j=1}^m r_{\mathsf{cn}_{r,j}}) : \\ \prod_{i=1}^n \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, (\sum_{i=1}^n v_{s,i}, \cdot, \cdot, \cdot), \sum_{i=1}^n r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}), \\ \prod_{j=1}^m \mathsf{cn}_{r,j}^{\mathcal{B}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, (\sum_{j=1}^m v_{r,j}, \cdot, \cdot, \cdot), \sum_{j=1}^m r_{\mathsf{cn}_{r,j}}), \\ \mathsf{acc}^{\mathcal{B},\mathsf{new}}/\mathsf{acc}^{\mathcal{B}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, (\sum_{i=1}^n v_{s,i} - \sum_{j=1}^m v_{r,j}, 0, \\ \sum_{j=1}^m v_{r,j}, 0, \sum_{i=1}^n v_{s,i}, 0, \cdot, \cdot, 0), r_{\mathsf{acc}}^{\mathsf{new}} - r_{\mathsf{acc}}), \\ \mathsf{acc}^{\mathcal{B},\mathsf{new}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, (\cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{AccTag}^a, a), r_{\mathsf{acc}}^{\mathsf{new}}) \end{array} \right\}$$

$\vec{\mathsf{ek}}_{\mathsf{acc}}$	ek_1	ek_2	ek_3	ek_4	ek_5	ek_6	ek_7	ek_8	ek_9	$\vec{\mathsf{ek}}_{\mathsf{cn}}$	ek_{10}	ek_{11}	ek_{12}	ek_{13}
acc	B	$B_{\max}$	S	$S_{\max}$	R	$R_{\max}$	pk_{U}	AccTag	a	cn	v	pk_s	pk_r	id
$\mathsf{acc}^{\mathcal{B}}$	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6	ψ_7	ψ_8	ψ_9	ψ_{10}	$\mathsf{cn}^{\mathcal{B}}$	ϕ_2	ϕ_3	ϕ_4	ϕ_5

Fig. 18: The correspondence among the encryption keys, the account/coin attributes, and the account/coin ciphertexts.

$$\left\{\begin{array}{c}(\vec{\mathsf{ek}}_{\mathsf{acc}}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{acc}^{\mathcal{B}}, \mathsf{acc}^{\mathcal{B},\mathsf{new}}, \{\mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}\}_{i=1}^n, \{\mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^m;\\ \mathsf{acc}, r_{\mathsf{acc}}, r_{\mathsf{acc}}^{\mathsf{new}}, \mathsf{sk}_{\mathsf{U}}, \{r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}\}_{i=1}^n, \{r_{\mathsf{cn}_{r,j}}\}_{j=1}^m) :\\ \mathsf{acc}^{\mathcal{B}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, (\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{pk}_{\mathsf{U}},\cdot,\cdot), r_{\mathsf{acc}}),\\ \mathsf{acc}^{\mathcal{B},\mathsf{new}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, (\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\mathsf{pk}_{\mathsf{U}},\cdot,\cdot), r_{\mathsf{acc}}^{\mathsf{new}}),\\ \forall i \in [n], \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, (\cdot,\cdot,\mathsf{pk}_{\mathsf{U}},\cdot), r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}),\\ \forall j \in [m], \mathsf{cn}_{r,j}^{\mathcal{B}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, (\cdot,\mathsf{pk}_{\mathsf{U}},\cdot,\cdot), r_{\mathsf{cn}_{r,j}})\end{array}\right\}$$

$$\left\{\begin{array}{c}(\vec{\mathsf{ek}}_{\mathsf{acc}}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{acc}^{\mathcal{B},\mathsf{new}}, \{\mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^m;\\ \mathsf{acc}^{\mathsf{new}}, r_{\mathsf{acc}}^{\mathsf{new}}, \{v_{r,j}, r_{\mathsf{cn}_{r,j}}\}_{j=1}^m) :\\ \mathsf{acc}^{\mathcal{B},\mathsf{new}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{acc}}, (B, B_{\max}, S, S_{\max}, R, R_{\max}, \cdot,\cdot,\cdot), r_{\mathsf{acc}}^{\mathsf{new}}),\\ \forall j \in [m], \mathsf{cn}_{r,j}^{\mathcal{B}} = \mathsf{Enc}(\vec{\mathsf{ek}}_{\mathsf{cn}}, (v_{r,j},\cdot,\cdot,\cdot), r_{\mathsf{cn}_{r,j}}),\\ v_{r,j} \in [0, \mathsf{Max}], B \in [0, B_{\max}], S \in [0, S_{\max}], R \in [0, R_{\max}]\end{array}\right\}$$

Let $\pi_{\text{trans}} = \pi_{\text{format}} \circ \pi_{\text{disclose}} \circ \pi_{\text{update}} \circ \pi_{\text{identity}} \circ \pi_{\text{range}}$, where $\circ$ denotes sequential composition. By the property of NIZK for conjunctive statements, π_{trans} is a NIZK proof for $\mathsf{R}_{\text{trans}}$. Therefore, the task breaks down to design $\pi_{\text{format}}, \pi_{\text{disclose}}, \pi_{\text{update}}, \pi_{\text{identity}}, \pi_{\text{range}}$. To prove the above relations, the prover and verifier execute the interactive Sigma protocol as follows, which can be transformed to be a non-interactive proof by the Fiat-Shamir transformation.

A. Zero-knowledge Proof for R_{format}

On statement $\vec{\mathsf{ek}}_{\mathsf{cn}}, \{\mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^m$, the prover P and verifier V parse $\mathsf{cn}_{r,j}^{\mathcal{B}} = (\phi_{r,j,1}, \ldots, \phi_{r,j,5})$ and interact as below:

- V randomly chooses $\{\gamma_j\}_{j=1}^m \leftarrow\$ \mathbb{Z}_p$ and sends them to P.
- P and V compute $\rho_{r,1} = \prod_{j=1}^m \phi_{r,j,1}^{\gamma_j}$, $\rho_{r,2} = \prod_{j=1}^m \phi_{r,j,2}^{\gamma_j}$.
- P picks $\eta_{r,1}, \eta_{r,2} \leftarrow\$ \mathbb{Z}_p$, sends $\rho'_{r,1} = g^{\eta_{r,1}}, \rho'_{r,2} = \mathsf{ek}_{10}^{\eta_{r,1}} h^{\eta_{r,2}}$ to V.
- V picks $c \leftarrow\$ \mathbb{Z}_p$ and sends it to P as the challenge.
- P computes $\omega_{r,1} = \eta_{r,1} - c(\sum_{j=1}^m \gamma_j r_{\mathsf{cn}_{r,j}})$, $\omega_{r,2} = \eta_{r,2} - c(\sum_{j=1}^m \gamma_j v_{r,j})$, then sends $(\omega_{r,1}, \omega_{r,2})$ to V. V accepts iff $\rho'_{r,1} = g^{\omega_{r,1}} \rho_{r,1}^c$, $\rho'_{r,2} = \mathsf{ek}_{10}^{\omega_{r,1}} h^{\omega_{r,2}} \rho_{r,2}^c$.

Lemma 1. *The above protocol is a public-coin SHVZK proof of knowledge for* R_{format}.

Proof. We prove that all three properties required are met. Perfect completeness is oblivious from simple calculation. To show the knowledge soundness, fix the initial message $(\{\gamma_j\}_{j=1}^m, \eta_{r,1}, \eta_{r,2})$, suppose there are two accepting scripts $(c, \omega_{r,1}, \omega_{r,2})$ and $(c', \omega'_{r,1}, \omega'_{r,2})$ with $c \neq c'$, the witness can be extracted as below. From the verification equations, we have $\eta_{r,1} = \omega_{r,1} + c(\sum_{j=1}^m \gamma_j r_{\mathsf{cn}_{r,j}})$, $\eta_{r,1} = \omega'_{r,1} + c'(\sum_{j=1}^m \gamma_j r_{\mathsf{cn}_{r,j}})$, which implies $\sum_{j=1}^m \gamma_j r_{\mathsf{cn}_{r,j}} = (\omega_{r,1} - \omega'_{r,1})/(c' - c)$. Further, we have $\eta_{r,2} = \omega_{r,2} + c(\sum_{j=1}^m \gamma_j v_{r,j})$, $\eta_{r,2} = \omega'_{r,2} + c'(\sum_{j=1}^m \gamma_j v_{r,j})$, which implies $\sum_{j=1}^m \gamma_j v_{r,j} = (\omega_{r,2} - \omega'_{r,2})/(c' - c)$. Defining $\{\gamma_j\}_{j=1}^m$ as a vector with 1 in the j-th position and 0 elsewhere, we can reconstruct $r_{\mathsf{cn}_{r,j}}$ and $v_{r,j}$.

To show SHVZK, for a fixed challenge c, the simulator Sim works as below: picks $\omega_{r,1}, \omega_{r,2} \leftarrow\$ \mathbb{Z}_p$, computes $\rho'_{r,1} = g^{\omega_{r,1}} \rho_{r,1}^c$, $\rho'_{r,2} = \mathsf{ek}_{10}^{\omega_{r,1}} h^{\omega_{r,2}} \rho_{r,2}^c$. Clearly, $(\{\gamma_j\}_{j=1}^m, \rho'_{r,1}, \rho'_{r,2}, c, \omega_{r,1}, \omega_{r,2})$ is an accepting transcript, and it is distributed as in the real protocol. □

B. Zero-knowledge Proof for $\mathsf{R}_{disclose}$

On statement $\vec{\mathsf{ek}}_{\mathsf{acc}}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{AccTag}, \mathsf{acc}^{\mathcal{B}}, \{\mathsf{id}_{s,i}, \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}\}_{i=1}^n$, P and V parse $\mathsf{cn}_{s,i}^{\mathcal{B}} = (\phi_{s,i,1}, \ldots, \phi_{s,i,5})$, $\mathsf{acc}^{\mathcal{B}} = (\psi_1, \ldots, \psi_{10})$, and interact as below:

- V randomly chooses $\{\beta_i\}_{i=1}^n \leftarrow\$ \mathbb{Z}_p$ and sends them to P.
- P and V compute $\rho_{s,1} = \prod_{i=1}^n \phi_{s,i,1}^{\beta_i}$, $\rho_{s,5} = \prod_{i=1}^n \phi_{s,i,5}^{\beta_i}$.
- P picks $\eta_{s,1}, \eta_1 \leftarrow\$ \mathbb{Z}_p$, sends $\rho'_{s,1} = g^{\eta_{s,1}}$, $\rho'_{s,5} = \mathsf{ek}_{13}^{\eta_{s,1}}$, $\psi'_1 = g^{\eta_1}$, $\psi'_9 = \mathsf{ek}_8^{\eta_1}$ to V.
- V picks $c \leftarrow\$ \mathbb{Z}_p$ and sends it to P as the challenge.
- P computes $\omega_{s,1} = \eta_{s,1} - c(\sum_{i=1}^n \beta_i r_{\mathsf{cn}_{s,i}})$, $\omega_1 = \eta_1 - c r_{\mathsf{acc}}$, then sends $(\omega_{s,1}, \omega_1)$ to V. V accepts iff $\rho'_{s,1} = g^{\omega_{s,1}} \rho_{s,1}^c$, $\rho'_{s,5} = \mathsf{ek}_{13}^{\omega_{s,1}} (\rho_{s,5} / \prod_{i=1}^n \mathsf{id}_{s,i}^{\beta_i})^c$, $\psi'_1 = g^{\omega_1} \rho_1^c$, $\psi'_9 = \mathsf{ek}_{13}^{\omega_1} (\psi_9 / \mathsf{AccTag})^c$.

Lemma 2. *The above protocol is a public-coin SHVZK proof of knowledge for* $\mathsf{R}_{disclose}$.

Proof. The proof technique of this lemma is the same as that of Lemma 1, therefore we ignore the detail. □

C. Zero-knowledge Proof for R_{update}

On statement $\vec{\mathsf{ek}}_{\mathsf{acc}}$, $\vec{\mathsf{ek}}_{\mathsf{cn}}$, $\mathsf{acc}^{\mathcal{B}}$, $\mathsf{acc}^{\mathcal{B},\mathsf{new}}$, $\prod_{i=1}^n \mathsf{cn}_{s,i}^{\mathcal{B},\mathsf{rnd}}$, $\prod_{j=1}^m \mathsf{cn}_{r,j}^{\mathcal{B}}$, AccTag, P and V parse $\mathsf{cn}_{s,i}^{\mathcal{B}} = (\phi_{s,i,1}, \ldots, \phi_{s,i,5})$, $\mathsf{cn}_{r,j}^{\mathcal{B}} = (\phi_{r,j,1}, \ldots, \phi_{r,j,5})$, $\mathsf{acc}^{\mathcal{B}} = (\psi_1, \ldots, \psi_{10})$, $\mathsf{acc}^{\mathcal{B},\mathsf{new}} = (\psi_1^{\mathsf{new}}, \ldots, \psi_{10}^{\mathsf{new}})$. P and V interact as below:

- V randomly chooses $\{\beta_i\}_{i=1}^n, \{\gamma_j\}_{j=1}^m \leftarrow\$ \mathbb{Z}_p$ and sends to the prover P.
- P and V compute $\rho_{s,1} = \prod_{i=1}^n \phi_{s,i,1}^{\beta_i}$, $\rho_{s,2} = \prod_{i=1}^n \phi_{s,i,2}^{\beta_i}$, $\rho_{r,1} = \prod_{j=1}^m \phi_{r,j,1}^{\gamma_j}$, $\rho_{r,2} = \prod_{j=1}^m \phi_{r,j,2}^{\gamma_j}$.
- P picks $\eta_{s,1}, \eta_{s,2}, \eta_{r,1}, \eta_{r,2}, \eta_1, \eta_2, \eta_3 \leftarrow\$ \mathbb{Z}_p$, sends $\rho'_{s,1} = g^{\eta_{s,1}}$, $\rho'_{s,2} = \mathsf{ek}_{10}^{\eta_{s,1}} h^{\eta_{s,2}}$, $\rho'_{r,1} = g^{\eta_{r,1}}$, $\rho'_{r,2} = \mathsf{ek}_{10}^{\eta_{r,1}} h^{\eta_{r,2}}$, $\psi'_1 = g^{\eta_1}$, $\psi'_2 = \mathsf{ek}_1^{\eta_1} h^{\eta_{s,2} - \eta_{r,2}}$, $\psi'_3 = \mathsf{ek}_2^{\eta_1}$, $\psi'_4 = \mathsf{ek}_3^{\eta_1} h^{\eta_{r,2}}$, $\psi'_5 = \mathsf{ek}_4^{\eta_1}$, $\psi'_6 = \mathsf{ek}_5^{\eta_1} h^{\eta_{s,2}}$, $\psi'_7 = \mathsf{ek}_6^{\eta_1}$, $\psi'_{10} = \mathsf{ek}_9^{\eta_1}$, $\psi'_{\mathsf{acc},1} = g^{\eta_2}$, $\psi'_{\mathsf{acc},9} = \mathsf{ek}_8^{\eta_2} \mathsf{AccTag}^{\eta_3}$, $\psi'_{\mathsf{acc},10} = \mathsf{ek}_9^{\eta_2} h^{\eta_3}$ to the verifier V.
- V picks $c \leftarrow\$ \mathbb{Z}_p$ and sends it to P as the challenge.

- P computes $\omega_{s,1} = \eta_{s,1} - c(\sum_{i=1}^{n} \beta_i r_{\mathsf{cn}_{s,i}})$, $\omega_{s,2} = \eta_{s,2} - c(\sum_{i=1}^{n} \beta_i v_{s,i})$, $\omega_{r,1} = \eta_{r,1} - c(\sum_{j=1}^{m} \gamma_j r_{\mathsf{cn}_{r,j}})$, $\omega_{r,2} = \eta_{r,2} - c(\sum_{j=1}^{m} \gamma_j v_{r,j})$, $\omega_1 = \eta_1 - c(r^{\mathsf{new}}_{\mathsf{acc}} - r_{\mathsf{acc}})$, $\omega_2 = \eta_2 - cr^{\mathsf{new}}_{\mathsf{acc}}$, $\omega_3 = \eta_3 - ca$, then sends $(\omega_{s,1}, \omega_{s,2}, \omega_{r,1}, \omega_{r,2}, \omega_1, \omega_2, \omega_3)$ to V. V accepts iff $\rho'_{s,1} = g^{\omega_{s,1}} \rho^c_{s,1}$, $\rho'_{s,2} = \mathsf{ek}_{10}^{\omega_{s,1}} h^{\omega_{s,2}} \rho^c_{s,2}$, $\rho'_{r,1} = g^{\omega_{r,1}} \rho^c_{r,1}$, $\rho'_{r,2} = \mathsf{ek}_{10}^{\omega_{r,1}} h^{\omega_{r,2}} \rho^c_{r,2}$, $\psi'_1 = g^{\omega_1} (\psi^{\mathsf{new}}_1/\psi_1)^c$, $\psi'_2 = \mathsf{ek}_1^{\omega_1} h^{\omega_{s,2} - \omega_{r,2}} (\psi^{\mathsf{new}}_2/\psi_2)^c$, $\psi'_3 = \mathsf{ek}_2^{\omega_1} (\psi^{\mathsf{new}}_3/\psi_3)^c$, $\psi'_4 = \mathsf{ek}_3^{\omega_1} h^{\omega_{r,2}} (\psi^{\mathsf{new}}_4/\psi_4)^c$, $\psi'_5 = \mathsf{ek}_4^{\omega_1} (\psi^{\mathsf{new}}_4/\psi_4)^c$, $\psi'_6 = \mathsf{ek}_5^{\omega_1} h^{\omega_{s,2}} (\psi^{\mathsf{new}}_6/\psi_6)^c$, $\psi'_7 = \mathsf{ek}_6^{\omega_1} (\psi^{\mathsf{new}}_7/\psi_7)^c$, $\psi'_{10} = \mathsf{ek}_9^{\omega_1} (\psi^{\mathsf{new}}_{10}/\psi_{10})^c$, $\psi'_{\mathsf{acc},1} = g^{\omega_1} (\psi^{\mathsf{new}}_1)^c$, $\psi'_{\mathsf{acc},9} = \mathsf{ek}_8^{\omega_2} \mathsf{AccTag}^{\omega_3} (\psi^{\mathsf{new}}_9)^c$, $\psi'_{\mathsf{acc},10} = \mathsf{ek}_9^{\omega_2} h^{\omega_3} (\psi^{\mathsf{new}}_{10})^c$.

Lemma 3. *The above protocol is a public-coin SHVZK proof of knowledge for* R_{update}.

Proof. The proof technique of this lemma is the same as that of Lemma 1, therefore we ignore the detail. □

D. Zero-knowledge Proof for $\mathsf{R}_{identity}$

On statement $\vec{\mathsf{ek}}_{\mathsf{acc}}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{acc}^{\mathcal{B},\mathsf{new}}, \{\mathsf{cn}^{\mathcal{B},\mathsf{rnd}}_{s,i}\}^n_{i=1}$, $\{\mathsf{cn}^{\mathcal{B}}_{r,j}\}^m_{j=1}$, P and V parse $\mathsf{cn}^{\mathcal{B}}_{s,i} = (\phi_{s,i,1}, \ldots, \phi_{s,i,5})$, $\mathsf{cn}^{\mathcal{B}}_{r,j} = (\phi_{r,j,1}, \ldots, \phi_{r,j,5})$, $\mathsf{acc}^{\mathcal{B}} = (\psi_1, \ldots, \psi_{10})$, $\mathsf{acc}^{\mathcal{B},\mathsf{new}} = (\psi^{\mathsf{new}}_1, \ldots, \psi^{\mathsf{new}}_{10})$. Then, P and V interact as below:

- P picks $\{\eta_{s,i}\}^n_{i=1}, \{\eta_{r,j}\}^m_{j=1}, \eta_1, \eta_2, \eta_3 \leftarrow\$ \mathbb{Z}_p$, sends $\{\phi'_{s,i,1} = g^{\eta_{s,i}}, \phi'_{s,i,4} = \mathsf{ek}_3^{\eta_{s,i}} h^{\eta_3}\}^n_{i=1}$, $\{\phi'_{r,j,1} = g^{\eta_{r,j}}, \phi'_{r,j,3} = \mathsf{ek}_2^{\eta_{r,j}} h^{\eta_3}\}^m_{j=1}$, $\psi'_1 = g^{\eta_1}$, $\psi'_8 = \mathsf{ek}_7^{\eta_1} h^{\eta_3}$, $(\psi^{\mathsf{new}}_1)' = g^{\eta_2}$, $(\psi^{\mathsf{new}}_8)' = \mathsf{ek}_7^{\eta_2} h^{\eta_3}$ to V.
- V picks $c \leftarrow\$ \mathbb{Z}_p$ and sends it to P as the challenge.
- P computes $\{\omega_{s,i} = \eta_{s,i} - cr_{\mathsf{cn}_{s,i}}\}^n_{i=1}$, $\{\omega_{r,j} = \eta_{r,j} - cr_{\mathsf{cn}_{r,j}}\}^m_{j=1}$, $\omega_1 = \eta_1 - cr_{\mathsf{acc}}$, $\omega_2 = \eta_2 - cr^{\mathsf{new}}_{\mathsf{acc}}$, $\omega_3 = \eta_3 - c\mathsf{sk}_{\mathsf{U}}$, then sends these elements to V. V accepts iff $\{\phi'_{s,i,1} = g^{\omega_{s,i}} \phi^c_{s,i,1}\}$, $\{\phi'_{s,i,4} = \mathsf{ek}_3^{\omega_{s,i}} h^{\omega_3} \phi^c_{s,i,4}\}$, $\{\phi'_{r,j,1} = g^{\omega_{r,j}} \phi^c_{r,j,1}\}$, $\{\phi'_{r,j,3} = \mathsf{ek}_2^{\omega_{r,j}} h^{\omega_3} \phi^c_{r,j,3}\}$, $\psi'_1 = g^{\omega_1} \psi^c_1$, $\psi'_8 = \mathsf{ek}_7^{\omega_1} h^{\omega_3} \psi^c_1$, $(\psi^{\mathsf{new}}_1)' = g^{\omega_2} (\psi^{\mathsf{new}}_1)^c$, $(\psi^{\mathsf{new}}_8)' = \mathsf{ek}_7^{\omega_2} h^{\omega_3} (\psi^{\mathsf{new}}_8)^c$.

Lemma 4. *The above protocol is a public-coin SHVZK proof of knowledge for* $\mathsf{R}_{identity}$.

Proof. The proof technique of this lemma is the same as that of Lemma 1, therefore we ignore the detail. □

E. Zero-knowledge Proof for R_{range}

On statement $\vec{\mathsf{ek}}_{\mathsf{acc}}, \vec{\mathsf{ek}}_{\mathsf{cn}}, \mathsf{acc}^{\mathcal{B},\mathsf{new}}, \{\mathsf{cn}^{\mathcal{B}}_{r,j}\}^m_{j=1}$, P and V parse $\mathsf{cn}^{\mathcal{B}}_{r,j} = (\phi_{r,j,1}, \ldots, \phi_{r,j,5})$, $\mathsf{acc}^{\mathcal{B},\mathsf{new}} = (\psi^{\mathsf{new}}_1, \ldots, \psi^{\mathsf{new}}_{10})$. Then, P and V interact as below:

- P computes $\{\mathsf{com}_k = \mathsf{ek}_{10}^{r_k} h^{\Delta_k}\}_{k \in \{B,S,R\}}$, where $\Delta_B = \mathsf{acc}.B_{\max} - \mathsf{acc}^{\mathsf{new}}.B$, $\Delta_S = \mathsf{acc}.S_{\max} - \mathsf{acc}^{\mathsf{new}}.S$, $\Delta_R = \mathsf{acc}.R_{\max} - \mathsf{acc}^{\mathsf{new}}.R$, and sends them to V.
- P picks $\{\eta_i\}^{10}_{i=1} \leftarrow\$ \mathbb{Z}_p$, sends $(\psi^{\mathsf{new}}_1)' = g^{\eta_1}$, $(\psi^{\mathsf{new}}_k)' = \mathsf{ek}_{k-1}^{\eta_1} h^{\eta_k}$, for $k \in \{2,3,4,5,6,7\}$, $\mathsf{com}'_B = \mathsf{ek}_{10}^{\eta_8} h^{\eta_3 - \eta_2}$, $\mathsf{com}'_S = \mathsf{ek}_{10}^{\eta_9} h^{\eta_5 - \eta_4}$, $\mathsf{com}'_R = \mathsf{ek}_{10}^{\eta_{10}} h^{\eta_7 - \eta_6}$ to V.
- V picks $c \leftarrow\$ \mathbb{Z}_p$ and sends it to P as the challenge.
- P computes $\omega_1 = \eta_1 - cr^{\mathsf{new}}_{\mathsf{acc}}$, $\omega_2 = \eta_2 - cB$, $\omega_3 = \eta_3 - cB_{\max}$, $\omega_4 = \eta_4 - cS$, $\omega_5 = \eta_5 - cS_{\max}$, $\omega_6 = \eta_6 - cR$, $\omega_7 = \eta_7 - cR_{\max}$, $\omega_8 = \eta_8 - cr_B$, $\omega_9 = \eta_9 - cr_S$, $\omega_{10} = \eta_{10} - cr_R$, then sends $\{\omega_k\}^{10}_{k=1}$ to V. V accepts iff $(\psi^{\mathsf{new}}_1)' = g^{\omega_1} (\psi^{\mathsf{new}}_1)^c$, $(\psi^{\mathsf{new}}_k)' = \mathsf{ek}_{k-1}^{\omega_1} h^{\omega_k} (\psi^{\mathsf{new}}_k)^c$, for $k \in \{2,3,4,5,6,7\}$, $\mathsf{com}'_B = \mathsf{ek}_{10}^{\omega_8} h^{\omega_3 - \omega_2} \mathsf{com}^c_B$, $\mathsf{com}'_S = \mathsf{ek}_{10}^{\omega_9} h^{\omega_5 - \omega_4} \mathsf{com}^c_S$, $\mathsf{com}'_R = \mathsf{ek}_{10}^{\omega_9} h^{\omega_7 - \omega_6} \mathsf{com}^c_R$.
- P and V execute Bulletproof [9] to prove that the values committed in $\{\phi_{r,j,2}\}^m_{j=1}, \psi^{\mathsf{new}}_2, \{\mathsf{com}_k\}_{k \in \{B,S,R\}}$, namely, $v_{r,j}, \mathsf{acc}^{\mathsf{new}}.B, \Delta_B, \Delta_S, \Delta_R$ are in $[0, \mathsf{Max}]$.

Lemma 5. *The above protocol is a public-coin SHVZK proof of knowledge for* R_{range}.

Proof. Perfect completeness is oblivious from simple calculation. To show the knowledge soundness, fix the initial message $(\{\mathsf{com}_k\}_{k \in \{B,S,R\}}, (\psi^{\mathsf{new}}_1)', (\psi^{\mathsf{new}}_k)'$, for $k \in \{2,3,4,5,6,7\}$, com'_B, com'_S, $\mathsf{com}'_R)$. Supposing there are two accepting scripts $(c, \{\omega_k\}^{10}_{k=1})$ and $(c, \{\omega'_k\}^{10}_{k=1})$ with $c \neq c'$, the witness can be extracted as below, which satisfies the relation we defined. From the verification equations, we have $g^{\omega_1} (\psi^{\mathsf{new}}_1)^c = g^{\omega'_1} (\psi^{\mathsf{new}}_1)^{c'}$, which implies $r^{\mathsf{new}}_{\mathsf{acc}} = (\omega_1 - \omega'_1)/(c' - c)$. For $k \in \{2, \cdots, 7\}$, define $\Delta\omega_k = \omega_k - \omega'_k$, $\Delta c = c' - c$. We have $\mathsf{ek}_{k-1}^{\omega_1} h^{\omega_k} (\psi^{\mathsf{new}}_k)^c = \mathsf{ek}_{k-1}^{\omega'_1} h^{\omega'_k} (\psi^{\mathsf{new}}_k)^c$, which implies $B = \Delta\omega_2/\Delta c$, $B_{\max} = \Delta\omega_3/\Delta c$, $S = \Delta\omega_4/\Delta c$, $S_{\max} = \Delta\omega_5/\Delta c$, $R = \Delta\omega_6/\Delta c$, $R_{\max} = \Delta\omega_7/\Delta c$. Further, we have $\mathsf{ek}_{10}^{\omega_8} h^{\omega_3 - \omega_2} \mathsf{com}^c_B = \mathsf{ek}_{10}^{\omega'_8} h^{\omega'_3 - \omega'_2} \mathsf{com}^{c'}_B$, which implies $r_B = \Delta\omega_8/\Delta c$, $\Delta_B = (\Delta\omega_3 - \Delta\omega_2)/\Delta c$. Similarly, the witness $r_S, \Delta_S, r_R, \Delta_R$ are extracted. The knowledge soundness of Bulletproof [9] guarantees that there exists $\{v^*_{r,j}, r^*_{\mathsf{cn}_{r,j}}\}^m_{j=1}$, $\mathsf{acc}^{\mathsf{new}}.B^*$, $r^{\mathsf{new},*}_{\mathsf{acc}}$, Δ^*_B, r^*_B, Δ^*_S, r^*_S, Δ^*_R, r^*_R satisfying

1) For $j \in [m]$, $\phi_{r,j,2} = \mathsf{ek}_{10}^{r^*_{\mathsf{cn}_{r,j}}} h^{v_{r,j}*}$;
2) $\psi^{\mathsf{new}}_2 = \mathsf{ek}_1^{r^{\mathsf{new},*}_{\mathsf{acc}}} h^{\mathsf{acc}^{\mathsf{new}}.B^*}$; $\{\mathsf{com}_k = \mathsf{ek}_{10}^{r^*_k} h^{\Delta^*_k}\}_{k \in \{B,S,R\}}$;
3) $v^*_{r,j}, \mathsf{acc}^{\mathsf{new}}.B^*, \Delta^*_B, \Delta^*_S, \Delta^*_R$ are in $[0, \mathsf{Max}]$.

We also have $\mathsf{acc}^{\mathsf{new}}.B^* = B$, $r^{\mathsf{new},*}_{\mathsf{acc}} = r^{\mathsf{new}}_{\mathsf{acc}}$, $\Delta^*_B = \Delta_B$, $r^*_B = r_B$, $\Delta^*_S = \Delta_S$, $r^*_S = r_S$, $\Delta^*_R = \Delta_R$, $r^*_R = r_R$, otherwise, we can solve some discrete logarithm problems.

The SHVZK property is from the SHVZK property of Bulletproof and the proof technique of Lemma 1. □

Putting all the sub-protocols described as above, we derive the zero-knowledge proof π_{trans} for $\mathsf{R}_{\text{trans}}$ involved in n-deposit-m-payment protocol.

NIZKs in user registration and auditing. The formal definitions of the relations and corresponding Sigma protocols to generate $\pi_{\text{trace}}, \pi_j$ ($j \in [D]$) involved in user registration and auditing protocols can be referred to Section 5.5 of [29]. To generate π_{reg} in the user registration protocol, the formal definition of the relation and the NIZK protocol are similar to (and simpler than) the NIZK in the n-deposit-m-payment protocol. The implementation details of these Sigma protocols can also be found in our source code.

APPENDIX F
DEFINITIONS OF IDEAL FUNCTIONALITIES

The Ideal Functionality of NIZK. $\mathcal{F}_{\mathsf{NIZK}}$ defines a non-interactive zero knowledge proof system (NIZK) parameterized by a relation $\mathsf{R}(x; w)$, where x is a statement and w is a corresponding witness. It was first formalized by Groth *et al.* [28]. This ensures that proofs can only be verified if valid witnesses exist, while maintaining zero-knowledge by hiding the witness during proof generation.

The functionality interacts with some party U and its adversary $\mathcal{A}$ (i.e., simulator). It is parameterized by a relation $\mathsf{R} = (x, w)$.

Proof. Upon receiving $(\text{Prove}, sid, x, w)$ from U, ignore if $(x, w) \notin \mathsf{R}$. Else, send (Prove, sid, x) to $\mathcal{A}$. Upon receiving (Proof, sid, π) from $\mathcal{A}$, store (x, π) and send (Proof, sid, π) to U.

Verify. Upon receiving $(\text{Verify}, sid, x, \pi)$ from U, check whether (x, π) is stored. If not send $(\text{Verify}, sid, x, \pi)$ to $\mathcal{A}$. Upon receiving $(\text{Witness}, sid, w)$ from $\mathcal{A}$, check $(x, w) \in \mathsf{R}$ and if so, store (x, π). If (x, π) has been stored, output $(\text{Verification}, sid, 1)$ to U, else output $(\text{Verification}, sid, 0)$.

Fig. 19: Ideal Functionality $\mathcal{F}_{\mathsf{NIZK}}$

Functionalities of communication channels. Our protocol uses different types of communication channels $\mathcal{F}_{\mathsf{Ch}}$ to deliver messages, which are formalized in [29].

The functionality interacts with the sender S, receiver R, and its adversary $\mathcal{A}$. Δ is flexibly defined based on the parameters of functionality. Message identifier mid is selected freshly by the functionality.

Upon receiving (Send, sid, R, m) from S, output $(\text{Send}, sid, \Delta, \mathsf{mid})$ to $\mathcal{A}$. Upon receiving $(\text{ok}, sid, \mathsf{mid})$ from $\mathcal{A}$, send $(\text{Receive}, sid, S, m)$ to R.

The parameter Δ is set based on the following parameterized functions:

- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, set $\Delta = (S, R, m)$.
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$, set $\Delta = (|m|)$.
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$, set $\Delta = (S, R, |m|)$.
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, set $\Delta = (R, |m|)$. Furthermore, unlike other functionalities, $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ proceeds as follows. Upon receiving $(\text{ok}, sid, \mathsf{mid})$ from $\mathcal{A}$, send $(\text{Receive}, sid, m, \mathsf{mid})$ to R. Upon receiving $(\text{Send}, sid, \mathsf{mid}, m')$ from R, output $(\text{Send}, sid, R, m', \mathsf{mid})$ to $\mathcal{A}$. Upon receiving $(\text{ok}, sid, \mathsf{mid})$ from $\mathcal{A}$, send $(\text{Receive}, sid, R, m')$ to S.

Fig. 20: Ideal Functionality $\mathcal{F}_{\mathsf{Ch}}$

We also define the standard Broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ from [24]. It does not guarantee secrecy for the message m.

The functionality interacts with the sender S a set of parties $\mathbb{R} = (\mathsf{R}_1, \ldots, \mathsf{R}_D)$, and its adversary $\mathcal{A}$.

Upon receiving $(\text{Broadcast}, sid, m)$ from S, send $(\text{Bdcasted}, sid, S, m)$ to all parties in $\mathbb{R}$ and to $\mathcal{A}$.

Fig. 21: Ideal Functionality $\mathcal{F}_{\mathsf{BC}}$

APPENDIX G
SECURITY PROOF

In this section, we give the proof of Theorem 2. The currency issuance protocol is actually a specific case of n-deposit-m-payment protocol, which removes the account-related operations and sets $n = 0$. Therefore, we provide the proof for the more general n-deposit-m-payment protocol, which inherently covers the security of the currency issuance protocol.

A. The Simulation Procedure

User Registration.

Case 1: Honest user U. Receiving $(\text{GenAcc}, sid, \mathsf{U}, B_{\max}, S_{\max}, R_{\max})$ from $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$:

1. Executes the original user registration protocol on behalf of U, M^*, honest regulators $\mathsf{R}_{\mathcal{H}}$ and communication channels.

2. Sends $(\text{ok.GenAcc}, sid, \mathsf{U}, \mathsf{acc}^{\mathcal{B}})$ to $\mathcal{F}_{\mathsf{CBDC}}$.

Case 2: Corrupt user U. $\mathcal{S}$ executes the original protocol on behalf of honest regulators $\mathsf{R}_{\mathcal{H}}$ and communication channels.

1. Receiving $(\text{Receive}, sid, \mathsf{U}, \mathsf{RI})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ on behalf of M^*, $\mathcal{S}$:
- Parses RI as $(\mathsf{pk}_{\mathsf{U}}, \mathsf{acc}^{\mathcal{B}}, B_{\max}, S_{\max}, R_{\max}, \pi_{\text{reg}})$.
- Executes the protocol on behalf of M^* and $\mathsf{R}_{\mathcal{H}}$.
- If there is no abort, sends $(\text{GenAcc}, sid, \mathsf{U}, B_{\max}, S_{\max}, R_{\max})$ to $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of U.

2. Receiving $(\text{GenAcc}, sid, \mathsf{U}, B_{\max}, S_{\max}, R_{\max})$ from $\mathcal{F}_{\mathsf{CBDC}}$, sends $(\text{ok.GenAcc}, sid, \mathsf{U}, \mathsf{acc}^{\mathcal{B}})$ to $\mathcal{F}_{\mathsf{CBDC}}$.

***n*-Deposit-*m*-Payment.**

Case 1: Honest user U, honest senders $\mathsf{U}_{s,\mathcal{H}}$, corrupt senders $\mathsf{U}_{s,\mathcal{C}}$, honest receivers $\mathsf{U}_{r,\mathcal{H}}$, corrupt receivers $\mathsf{U}_{r,\mathcal{C}}$.

1. From $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$ receives
- $(\text{Deposit}, sid, \mathsf{did}_i)$ for $\mathsf{U}_{s,i} \in \mathsf{U}_{s,\mathcal{H}}$.
- $(\text{Deposit}, sid, \mathsf{did}_i, \mathsf{U}_{s,i}, \mathsf{U}, v_i)$ for for $\mathsf{U}_{s,i} \in \mathsf{U}_{s,\mathcal{C}}$.
- $(\text{Payer}, sid, \mathsf{pid}_j)$ for $\mathsf{U}_{r,j} \in \mathsf{U}_{r,\mathcal{H}}$.
- $(\text{Payer}, sid, \mathsf{pid}_j, \mathsf{U}, \mathsf{U}_{r,j}, v_j)$ for $\mathsf{U}_{r,j} \in \mathsf{U}_{r,\mathcal{C}}$.

2. $\mathcal{S}$ initiates on behalf of U (U is randomly chosen if $\{\mathsf{U}_{s,i}\}_{i=1}^n \subseteq \mathsf{U}_{s,\mathcal{H}}, \{\mathsf{U}_{r,j}\}_{j=1}^m \subseteq \mathsf{U}_{r,\mathcal{H}}$):
- For $\mathsf{U}_{s,i} \in \mathsf{U}_{s,\mathcal{H}}$, sets $\mathsf{cn}_{s,i} = (0, h^0, \mathsf{pk}_{\mathsf{U}}, \mathsf{id}_i)$ where $\mathsf{id}_{s,i} = h^{y_i}$, $(y_i \leftarrow\$ \mathbb{Z}_p^*)$, encrypts these coins $\{\mathsf{cn}_{s,i}^{\mathcal{B}}\}_{i=1}^n$ and signs on it.
- For $\mathsf{U}_{r,j} \in \mathsf{U}_{r,\mathcal{H}}$, sets $\mathsf{cn}_{r,j} = (0, \mathsf{pk}_{\mathsf{U}}, h^0, \mathsf{id}_j)$ where $\mathsf{id}_{r,j} = h^{y_j}$, $(y_j \leftarrow\$ \mathbb{Z}_p^*)$ and encrypts these dummy coins $\{\mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^m$.
- For $\mathsf{U}_{s,i} \in \mathsf{U}_{s,\mathcal{C}}$, $\mathsf{U}_{r,j} \in \mathsf{U}_{r,\mathcal{C}}$, executes the original protocol.
- Defines $\mathsf{acc}^{\mathsf{new}} = (0, 0, 0, 0, 0, 0, \mathsf{pk}_{\mathsf{U}}, h^{z_{e+1}}, h^a)$, where $z_{e+1} \leftarrow\$ \mathbb{Z}_p^*$ and encrypts the dummy account $\mathsf{acc}^{\mathcal{B},\mathsf{new}}$.
- Discloses the deposited dummy coin tags $\{\mathsf{id}_{s,i}\}_{i=1}^n$ and the dummy old account tag $\mathsf{AccTag} = h^{z_e}$ $(z_e \leftarrow\$ \mathbb{Z}_p^*)$.
- Calls internally run adversary $\mathcal{A}$ with input (prove, sid, x) on behalf of $\mathcal{F}_{\mathsf{NIZK}}$ and receives $(\text{Proof}, sid, \pi_{\text{trans}})$.
- Packages transaction message TI_s calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with input $(\text{Send}, sid, \mathsf{M}^*, \mathsf{TI}_s)$.

3. Executes the following protocol on behalf of M^* and emulates communication channels.

4. Receiving $(\text{Receive}, sid, \mathsf{M}^*, \{\hat{\sigma}_{\mathsf{cn}_{r,j}}\}_{j=1}^m, \hat{\sigma}_{\mathsf{acc}}^{\mathsf{new}})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, $\mathcal{S}$:
- Sends $(\text{ok.UpAcc}, sid, \{\mathsf{cn}_{s,i}^{\mathcal{B}}\}_{i=1}^n, \{\mathsf{pid}_j, \mathsf{cn}_{r,j}^{\mathcal{B}}\}_{j=1}^m, \mathsf{acc}^{\mathcal{B},\mathsf{new}})$ to $\mathcal{F}_{\mathsf{CBDC}}$.
- For $\mathsf{U}_{r,j} \in \mathsf{U}_{r,\mathcal{C}}$, executes the original protocol on behalf of U and emulates the communication channels.

5. Receiving $(\text{Payee}, sid, \mathsf{pid}')$ from $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$ replies $\mathcal{F}_{\mathsf{CBDC}}$ with $(\text{ok.Payee}, sid, \mathsf{pid}')$.

Case 2: Corrupt U, honest senders $\mathsf{U}_{s,\mathcal{H}}$, corrupt senders $\mathsf{U}_{s,\mathcal{C}}$, honest receivers $\mathsf{U}_{r,\mathcal{H}}$, corrupt receivers $\mathsf{U}_{r,\mathcal{C}}$.

1. Receiving $(\text{Send}, sid, \mathsf{M}^*, \mathsf{TI}_s)$ on behalf of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, $\mathcal{S}$:
- Proceeds as the original protocol on behalf of M^*.
- Extracts the witness w involved by TI_s. Specifically, checks if (x, π_{trans}) is stored which means that someone has called $\mathcal{F}_{\mathsf{NIZK}}$ (emulated by $\mathcal{S}$) with input $(\text{Prove}, sid, x, w)$. Otherwise, $\mathcal{S}$ sends $(\text{Verify}, sid, x, \pi_{\text{trans}})$ to internally run $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{NIZK}}$ and receives $(\text{Witness}, sid, w)$ from $\mathcal{A}$.
- Parses w as $(\{v_{r,j}, \mathsf{sk}_{\mathsf{U}}, r_{\mathsf{cn}_{r,j}}\}_{j=1}^m, \{\mathsf{sk}_{\mathsf{U}}, \mathsf{sn}_{s,i}, r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}\}_{i=1}^n, \cdot)$.
- Parses $\mathsf{cn}_j^{\mathcal{B}}$ as $(\psi_{j,1}, \psi_{j,2}, \psi_{j,3}, \psi_{j,4}, \psi_{j,5})$.
- Computes $\mathsf{pk}_{\mathsf{U}_{r,j}} = \psi_{j,4}/(\mathsf{ek}_{\mathsf{cn},3}^{r_{\mathsf{cn}_j}})$, and retrieves the identity $\mathsf{U}_{r,i}$ of $\mathsf{pk}_{\mathsf{U}_{r,i}}$. If $\mathsf{pk}_{\mathsf{U}_{r,i}}$ is invalid, set $\mathsf{U}_{r,i} = \perp$. Similarly, extracts $\{\mathsf{U}_{s,i}, v_i\}_{i=1}^n$ with $r_{\mathsf{cn}_{s,i}}^{\mathsf{rnd}}$.
- Sends $(\text{Payer}, sid, \{\mathsf{U}_{s,i}, v_{s,i}\}_{i=1}^n, \{\mathsf{U}_{r,j}, v_{r,j}\}_{j=1}^m)$ to $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of U.

2. From $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$ receives $\{(\text{Deposit}, sid, \mathsf{did}_i, \cdot)\}_{i=1}^n$ and $\{(\text{Payer}, sid, \mathsf{pid}_j, \cdot)\}_{j=1}^m$, then sends $(\text{ok.UpAcc}, sid, \cdot)$ to $\mathcal{F}_{\mathsf{CBDC}}$.

3. For $\mathsf{U}_{r,j} \in \mathsf{U}_{r,\mathcal{H}}$, upon receiving $(\text{Payee}, sid, \mathsf{pid}', \mathsf{U}_{r,j}, v_{r,j})$ from $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$ executes the original protocol on behalf of

honest $\mathsf{U}_{r,j}$. If $\mathcal{S}$ aborts when emulating $\mathsf{U}_{r,j}$, stop. Otherwise, $\mathcal{S}$ sends $(\text{ok.Payee}, sid, \mathsf{pid}')$ to $\mathcal{F}_{\mathsf{CBDC}}$.

4. For $\mathsf{U}_{r,j} \in \mathsf{U}_{r,\mathcal{C}}$, $\mathcal{S}$ sends $(\text{Payee}, sid, \mathsf{U}, v_{r,j})$ to $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of $\mathsf{U}_{r,j}$. Upon receiving $(\text{Payee}, sid, \mathsf{pid}', \mathsf{U}_{r,j}, v_{r,j})$ from $\mathcal{F}_{\mathsf{CBDC}}$, replies with $(\text{ok.Payee}, sid, \mathsf{pid}')$.

Privacy Revocation.

Case 1: Honest U_s, honest U_r. Receiving $(\text{Revoke}, sid, \mathsf{cn}^{\mathcal{B},j}, \mathsf{R}_j)$ from $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$ replies with $(\text{ok.Revoke}, sid, \mathsf{cn}^{\mathcal{B},j})$. Upon receiving $(\text{Revoked}, sid, \mathsf{U}_s, \mathsf{U}_r, v)$ from $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$:

1. Parses $\mathsf{cn}^{\mathcal{B},j}$ as $(\psi_1, \psi_2, \psi_3, \psi_4, \psi_5)$.
2. Defining the aggregated results ψ_2/h^v, ψ_3/pk_s and ψ_4/pk_r, $\mathcal{S}$ computes w-th honest regulator's decryption share as follows with knowing malicious regulators shares $\psi_1^{\mathsf{dk}_{\mathsf{R}_t,10}}$, $\psi_1^{\mathsf{dk}_{\mathsf{R}_t,11}}$, and $\psi_1^{\mathsf{dk}_{\mathsf{R}_t,12}}$ for $\forall t \in \mathcal{C}$. For example, for the w-th honest regulation's decryption share for the payment value, $\mathcal{S}$ computes

$$\psi_1^{\mathsf{dk}_{\mathsf{R}_w,10}} = (\psi_2/g^v)^{\prod_{k\in\mathcal{C}}(k-w)/k} \cdot \prod_{t\in\mathcal{C}} \left(\psi_1^{\mathsf{dk}_{\mathsf{R}_t,10}}\right)^{(w/t)\prod_{k\in\mathcal{C},k\neq t}(w-t)/(t-k)}$$

3. $\mathcal{S}$ sets $x_w = (\{\mathsf{ek}_{\mathsf{R}_w,i}, \psi_1, \psi_1^{\mathsf{dk}_{\mathsf{R}_w,i}}\}_{i\in\{10,11,12\}})$, sends (Prove, sid, x_w) to the dummy (internally run) adversary $\mathcal{A}$ of $\mathcal{F}_{\mathsf{NIZK}}$, and receives $(\text{Proof}, sid, \pi_w)$.
4. Calls the authenticated channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with input $(\text{Send}, sid, \mathsf{R}_j, x_w, \pi_w)$ for $j \in [D] \wedge j \neq w$ on behalf of R_w.

Case 2: Honest (resp. corrupt) U_s, corrupt (resp. honest) U_r, or corrupt $\mathsf{U}_s, \mathsf{U}_r$. The simulation of this case is similar to the case of honest U_s and honest U_r except the fact that there is no need for faking the shares. This is because, in this case, the identities of participants U_s, U_r, and transaction value v encrypted by ciphertexts involved in $\mathsf{cn}^{\mathcal{B}}$ are consistent with that received from $\mathcal{F}_{\mathsf{CBDC}}$.

Tracing.

Case 1: Honest user U. Upon receiving $(\text{Revoke}, sid, \mathsf{U}_j, \mathsf{R}_j)$ from $\mathcal{F}_{\mathsf{CBDC}}$, $\mathcal{S}$ replies with $(\text{ok.Revoke}, sid, \mathsf{U}_j)$. Upon receiving $(\text{Revoked}, sid, \{(\mathsf{cn}^{\mathcal{B}}, \mathsf{role})\})$,

1. Finds $(\mathsf{U}_j, a_j, r_j, \mathsf{com}_{\mathsf{R}})$ saved in $\mathcal{L}_{\mathsf{R}_j}$ and sets $e = 1, \tilde{h} = h$.
2. Retrieve $\{\mathsf{AccTag}_\tau\}_{\tau=1}^{x}$ from $\mathcal{L}$ referring to $\{(\mathsf{cn}^{\mathcal{B}}, \mathsf{role})\}$.
3. Computes w-th honest regulator's first tracing tag shares:

$$\tilde{h}^{a_w} = (\mathsf{AccTag}_e)^{\prod_{k\in\mathcal{C}}(k-w)/k} \cdot \prod_{t\in\mathcal{C}} \left(\tilde{h}^{a_t}\right)^{(w/t)\prod_{k\in\mathcal{C},k\neq t}(w-t)/(t-k)}$$

4. Sets $x_w = (\mathsf{com}_w, \tilde{h}^{a_w}, \tilde{h})$. Then, $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{NIZK}}$ and sends (Prove, sid, x_w) to the dummy (internally run) adversary $\mathcal{A}$ and receives $(\text{Proof}, sid, \pi_w)$.
5. Calls the authenticated channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with input $(\text{Send}, sid, x_w, \pi_w)$ for $j = 1, \ldots, D \wedge j \neq w$ on behalf of R_w.
6. $\mathcal{S}$ proceeds from step 3 with $\tilde{h} = \mathsf{AccTag}_e$ and $e = e + 1$. Aborts if $e = x + 1$.

Case 2: Corrupt user U. The simulation of this case is similar to the above case except the fact that there is no need for faking the shares. This is because, in this case, the tags generated by corrupt U are consistent with that received from $\mathcal{F}_{\mathsf{CBDC}}$.

Fig. 22: The simulation procedure.

B. Sequence of Games

Through a sequence of games, we show that the random variables $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathsf{Ch}},\mathcal{F}_{\mathsf{BC}},\mathcal{F}_{\mathsf{NIZK}}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ are indistinguishable. We denote by $\Pr[\mathsf{Game}^i]$ the probability that the environment $\mathcal{Z}$ outputs 1 in Game^i. Each game Game^i has its own $\mathcal{F}_{\mathsf{CBDC}}^i$ and $\mathcal{S}^i$. We start from the most leaky functionality $\mathcal{F}_{\mathsf{CBDC}}^0$ and the associated simulator $\mathcal{S}^0$ and gradually toward the main functionality $\mathcal{F}_{\mathsf{CBDC}}$ and $\mathcal{S}$. Without loss of generality and for expression, we set the number of malicious regulators to equal to t.

Game^0. Initially, $\mathcal{F}_{\mathsf{CBDC}}^0$ does nothing but transmit the most leaky messages to the simulator $\mathcal{S}^0$. And $\mathcal{S}^0$ executes the real-world protocol on behalf of honest parties.

Game^1. $\mathcal{F}_{\mathsf{CBDC}}^1$ executes as $\mathcal{F}_{\mathsf{CBDC}}$ and still transmits the most leaky messages to the simulator $\mathcal{S}^1$. And $\mathcal{S}^1$ on behalf of honest parties executes the real-world protocol and extracts the behavior of corrupt parties. Game^1 is the same as Game^0 except that Game^1 would abort if two events occur.

- event_1: $\mathcal{F}_{\mathsf{CBDC}}^1$ aborts for two cases. The first case is that the user U wants to deposit coins from the air. Specifically, when a corrupt U requests a deposit transaction, $\mathcal{S}^1$ extracts coin information $(\{\mathsf{U}_{s,i}, v_{s,i}\}_{i=1}^n)$ and sends $(\text{Deposit}, sid, \{\mathsf{U}_{s,i}, v_{s,i}\}_{i=1}^n)$ to $\mathcal{F}_{\mathsf{CBDC}}^1$ on behalf of U, but $\mathcal{F}_{\mathsf{CBDC}}^1$ aborts for $\{\mathsf{U}_{s,i}, v_i\}_{i=1}^n \subseteq \mathsf{Cn}[\mathsf{U}]$ does not hold. The second one is that a maliciously generated account is updated in the real world. Specifically, when a corrupt user U requests payment or deposit transactions, $\mathcal{S}^1$ extracts and sends $(\text{Payer}, sid, \{\mathsf{U}_{s,i}, v_{s,i}\}_{i=1}^n, \{\mathsf{U}_{r,j}, v_{r,j}\}_{j=1}^m)$ to $\mathcal{F}_{\mathsf{CBDC}}^1$ on behalf of U, but $\mathcal{F}_{\mathsf{CBDC}}^1$ aborts for account-related checks do not hold.
- event_2: When more than one user requests (including one honest user) to deposit the same coin, i.e., the same id. Although double-spending can be prevented by the uniqueness check of id, event_2 means the honest receiver is convinced to finish the payment offline, but the coin id has already appeared in the global ledger. In Game^1, when such an event occurs, $\mathcal{S}^1$ would abort.

Therefore, we have

$$\left|\Pr[\mathsf{Game}^1] - \Pr[\mathsf{Game}^0]\right| \leq \Pr[\mathsf{event}_1] + \Pr[\mathsf{event}_2].$$

To prove the difference between Game^0 and Game^1 is negligible, we prove that $\Pr[\mathsf{event}_1]$ and $\Pr[\mathsf{event}_2]$ are both negligible. If event_1 occurs, we can construct an adversary $\mathcal{B}_1$ to break the unforgeability of SoRC. During the reduction from the *unforgeability* of SoRC to event_1 in Game^1, $\mathcal{B}_1$ receives $\hat{\mathsf{vk}}_{\mathsf{M}^*}$ from the challenger of the unforgeability game. When $\mathcal{B}_1$, on behalf of M^*, should generate a signature on valid ciphertexts, it queries the signing oracle to derive a signature $\hat{\sigma}_{\mathsf{M}^*}$. If event_1 occurs, $\mathcal{B}_1$ derives the encrypted coin/account and the corresponding signature and sends them to the challenger of the unforgeability game. Therefore, $\mathcal{B}_1$ wins if and only if event_1 occurs and we have

$$\Pr[\mathsf{event}_1] \leq \mathsf{Adv}_{\mathcal{B}_1,\mathsf{SoRC}}^{\text{unforge}}(\lambda).$$

event_2 means the coin id breaks the *offline-checkable uniqueness*. From Theorem 1, we have

$$\Pr[\mathsf{event}_2] \leq p_u \cdot \mathsf{Adv}_{\mathcal{B}_2}^{\text{dl}}(\lambda).$$

Game^2. Same as Game^1 except that in Game^2, when operating the auditing protocol,

- Instead of computing $\{\psi_1^{\mathsf{dk}_{\mathsf{R}_w,i}}\}_{i=1}^{13}$ with honest users' secret key shares, $\mathcal{S}^2$ simulates the decryption shares with the help

of non-threshold ciphertexts $\mathsf{cn}^{\mathcal{B}}$, $\mathsf{acc}^{\mathcal{B}}$, secret key shares (randomly chosen by $\mathcal{S}^2$) of corrupt regulators $\{\mathsf{sk}_{\mathsf{R}_t,i}\}_{i=1}^{13}$ ($t \in \mathcal{C}$) and the plaintexts encrypted by the cipertexts. Here, the plaintexts are either generated by $\mathcal{S}^2$ on behalf of honest users or extracted from the corresponding proof.
- Instead of computing $\tilde{h}^{a_w}$ with the share a_w of the tracing key, $\mathcal{S}^3$ simulates the tracing shares with the help of the message $(\text{Traced}, sid, \{\mathsf{cn}^{\mathcal{B}}, \mathsf{role}\}_{\tau=1}^{x})$ (to retrieve tracing tags $\{\mathsf{AccTag}_\tau = g^{z_\tau}\}_{\tau=1}^{x}$) and tracing key shares of corrupted regulators.

Since the threshold ElGamal encryption scheme has perfect *decryption consistency*, we have

$$\Pr[\mathsf{Game}^1] = \Pr[\mathsf{Game}^2].$$

$\underline{\mathsf{Game}^3}$. Same as Game^2 except that in Game^3 we change the tracing tags of honest users to the dummy values selected by $\mathcal{S}^3$. Any difference between Game^2 and Game^3 is because of distinguishing g^{a^x} values from g^z values. We define $\mathsf{Game}_0^2 = \mathsf{Game}^2$ and p_u as the upper bound on the number of all honest users. We also define Game_1^2 as a game similar to Game_0^2 except that in Game_1^2, we change the tracing tags of the first user from real-world values to ideal-world dummy values. Then, in Game_i^2 ($i \in [p_u]$), we replace the tracing tags of the i-th user with a dummy values. And $\mathsf{Game}_{p_u}^2 = \mathsf{Game}^3$. Any difference between the two adjacent games is upper bounded by $\mathsf{Adv}_{\mathcal{B}_3}^{\text{d-sddh}}(\lambda)$. Below we present the reduction between Game_{i-1}^2 and Game_i^2 as an example.

Associated reduction between Game_{i-1}^2 and Game_i^2. If the environment $\mathcal{Z}$ can distinguish Game_{i-1}^2 and Game_i^2, we can construct an adversary $\mathcal{B}_3$ to break the hardness of d-strong Diffie-Hellman problem. During the reduction, $\mathcal{B}_3$ receives a challenge instance from its challenger and uses it as tracing tags of the i-th honest user. If it is a d-strong Diffie-Hellman instance, $\mathcal{B}_3$ simulates Game_{i-1}^2. If it is a random instance, $\mathcal{B}_3$ simulates Game_i^2. If $\mathcal{Z}$ distinguishes Game_{i-1}^2 and Game_i^2, $\mathcal{B}_3$ can solve d-strong Diffie-Hellman problem. Therefore,

$$\left|\Pr[\mathsf{Game}_{i-1}^2] - \Pr[\mathsf{Game}_i^2]\right| \leq \mathsf{Adv}_{\mathcal{B}_3}^{\text{d-sddh}}(\lambda).$$

Thereafter, the following inequality holds

$$\left|\Pr[\mathsf{Game}^3] - \Pr[\mathsf{Game}^2]\right| \leq (p_u) \cdot \mathsf{Adv}_{\mathcal{B}_3}^{\text{d-sddh}}(\lambda).$$

$\underline{\mathsf{Game}^4}$. Same as Game^3 except that in Game^4:
- When simulating the account update operation on behalf of an honest user U, the simulator $\mathcal{S}^4$, on behalf of U, sends a dummy account and the corresponding signature to M^*.
- When simulating the payment between an honest pair $(\mathsf{U}_s, \mathsf{U}_r)$, the simulator $\mathcal{S}^4$, on behalf of U_s, sends a dummy coin to U_r. Thereafter, $\mathcal{S}$, on behalf of U_r, deposits the dummy coin.

During privacy revocation, the decryption shares of honest regulators are computed by $\mathcal{S}^4$ with real-world plaintexts. Besides, $\mathcal{F}_{\mathsf{NIZK}}$ emulation allows $\mathcal{S}^4$ to provide faked proofs about the dummy coins and accounts. Therefore, the difference between Game^3 and Game^4 is because of the *class-hiding* and *signature-adaption* of SoRC.

We define $\mathsf{Game}_0^3 = \mathsf{Game}^3$ and p_a, p_c as the upper bound on the number of account update requests and honest coins, respectively. We also define Game_1^3 as a game similar to Game_0^3 except that in Game_1^3, when simulating the first account update operation of an honest party U, $\mathcal{S}_1^3$ replaces a randomized account with a dummy account. Then, in Game_i^3 ($i \in [p_a]$), we replace the i-th account with a dummy account. In $\mathsf{Game}_{p_a+j}^3$ ($j \in [p_c]$), when simulating the j-th payment between an honest pair $(\mathsf{U}_s, \mathsf{U}_r)$, $\mathcal{S}_{p_a+j}^3$ replaces the randomized coin with a dummy coin. And $\mathsf{Game}_{p_a+p_c}^3 = \mathsf{Game}^4$. Any difference between the two adjacent games is because of the class-hiding and signature-adaption of SoRC. Below we present the reduction between $\mathsf{Game}_{p_a}^3$ and $\mathsf{Game}_{p_a+1}^3$ as an example.

Associated reduction between $\mathsf{Game}_{p_a}^3$ and $\mathsf{Game}_{p_a+1}^3$. If the environment $\mathcal{Z}$ can distinguish $\mathsf{Game}_{p_a}^3$ and $\mathsf{Game}_{p_a+1}^3$, we can construct an adversary $\mathcal{B}_4$ to break the class-hiding security of SoRC. During the reduction, $\mathcal{B}_4$ receives $\{\mathsf{ek}_{\mathsf{R},i}\}_{i=10}^{13}$ from the challenger of class-hiding game. $\mathcal{B}_4$ generates the coin $\mathsf{cn}^{\mathcal{B}}$ on behalf of the honest user U_s and generates signatures $\hat{\sigma}_{\mathsf{cn}}$ on $\mathsf{cn}^{\mathcal{B}}$ on behalf of the maintainer M^*. Then, $\mathcal{B}$ sends $\mathsf{cn}^{\mathcal{B}}$ to the challenger of the class-hiding game and receives $\mathsf{cn}^{\mathcal{B},0}, \mathsf{cn}^{\mathcal{B},1}$. When $\mathcal{B}_3$ deposits the coin on behalf of honest U_r, it sets the coin as $\mathsf{cn}^{\mathcal{B},b}$ and generates the signature $\hat{\sigma}_{\mathsf{cn},b}$ as $\mathsf{SoRC.Sign}(\hat{\mathsf{sk}}_{\mathsf{M}^*}, \{\mathsf{ek}_{\mathsf{R},i}\}_{i=10}^{13}, \mathsf{cn}^{\mathcal{B},b})$. Considering the signature-adaptability of SoRC, if $b = 0$, $\mathcal{B}_4$ perfectly simulates $\mathsf{Game}_{p_a}^3$. And if $b = 1$, $\mathcal{B}_4$ perfectly simulates $\mathsf{Game}_{p_a+1}^3$. If $\mathcal{Z}$ distinguishes $\mathsf{Game}_{p_a}^3$ and $\mathsf{Game}_{p_a+1}^3$, $\mathcal{B}_4$ can win class-hiding game. Therefore, we have

$$\left|\Pr[\mathsf{Game}_{p_a}^3] - \Pr[\mathsf{Game}_{p_a+1}^3]\right| \leq \mathsf{Adv}_{\mathcal{B}_4,\mathsf{SoRC}}^{\text{class-hiding}}(\lambda).$$

Thereafter, the following inequality holds

$$\left|\Pr[\mathsf{Game}^4] - \Pr[\mathsf{Game}^3]\right| \leq (p_a + p_c) \cdot \mathsf{Adv}_{\mathcal{B}_4,\mathsf{SoRC}}^{\text{class-hiding}}(\lambda).$$

$\underline{\mathsf{Game}^5}$. Same as Game^4 except that in Game^5 we replace all newly generated ciphertexts of honest pairs with ciphertexts encrypting dummy values. Similar to Game_4, with the simulated decryption shares of honest regulators and $\mathcal{F}_{\mathsf{NIZK}}$, any difference between Game^4 and Game^5 is because of breaking the IND-CPA property of the encryption scheme used in our construction.

We define $\mathsf{Game}_0^4 = \mathsf{Game}^4$ and p_a, p_c as the upper bound on the number of account update requests and honest coins, respectively. We also define Game_1^4 as a game similar to Game_0^4 except in Game_1^4 we change the plaintexts of the first account from the real-world values to dummy values. Then, in Game_i^4 ($i \in [p_a]$), we replace the plaintexts of the i-th account with dummy values. In $\mathsf{Game}_{p_a+j}^4$ ($j \in [p_c]$), we replace the plaintexts of the j-th coin generated by honest users with dummy values. And $\mathsf{Game}_{p_a+p_c}^4 = \mathsf{Game}^5$. Any difference between the two adjacent games is because of breaking the IND-CPA property of the encryption scheme in our construction. And below we present the reduction between $\mathsf{Game}_{p_a}^4$ and $\mathsf{Game}_{p_a+1}^4$ as an example.

Associated reduction between $\mathsf{Game}^4_{p_a}$ and $\mathsf{Game}^4_{p_a+1}$. If the environment $\mathcal{Z}$ can distinguish $\mathsf{Game}^4_{p_a}$ and $\mathsf{Game}^4_{p_a+1}$, we can construct an adversary $\mathcal{B}_5$ to break the IND-CPA security of non-threshold ElGamal encryption. During the reduction, $\mathcal{B}_5$ receives $\{\mathsf{ek}_{\mathsf{R},i}\}_{i=10}^{13}$ from the challenger of IND-CPA game. When $\mathcal{B}_5$ needs to generate the first coin on behalf of honest users, $\mathcal{B}_5$ sends the real-world plaintexts and dummy plaintexts to the challenger of the IND-CPA game and receives ciphertexts. If the ciphertexts encrypt the real-world plaintexts, $\mathcal{B}_4$ simulates $\mathsf{Game}^4_{p_a}$. If the ciphertexts encrypt the dummy plaintexts, $\mathcal{B}_5$ simulates $\mathsf{Game}^3_{p_a+1}$. If $\mathcal{Z}$ distinguishes Game^4_0 and Game^4_1, $\mathcal{B}_5$ can win IND-CPA game. Therefore, we have

$$\left|\Pr[\mathsf{Game}^4_{p_a}] - \Pr[\mathsf{Game}^4_{p_a+1}]\right| \leq \mathsf{Adv}^{\text{ind-cpa}}_{\mathcal{B}_5,\mathrm{ElGamal}}(\lambda).$$

Thereafter, the following inequality holds

$$\left|\Pr[\mathsf{Game}^5] - \Pr[\mathsf{Game}^4]\right| \leq (p_a + p_c) \cdot \mathsf{Adv}^{\text{ind-cpa}}_{\mathcal{B}_5,\mathrm{ElGamal}}(\lambda).$$

$\underline{\mathsf{Game}^6}$. Game^6 is identical to ideal world. Same as Game^5 except that in Game^6, $\mathcal{F}^6_{\mathsf{CBDC}}$ leaks information to $\mathcal{S}^6$ as $\mathcal{F}_{\mathsf{CBDC}}$. Therefore, we have

$$\Pr[\mathsf{Game}^5] = \Pr[\mathsf{Game}^6].$$

Till now, we have $\mathcal{F}^6_{\mathsf{CBDC}} = \mathcal{F}_{\mathsf{CBDC}}$ and $\mathcal{S}^6 = \mathcal{S}$ which means Game^6 corresponds to the ideal world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$. Therefore, the probability for any PPT environment $\mathcal{Z}$ to distinguish $\mathsf{EXEC}^{\mathcal{F}_{\mathsf{Ch}},\mathcal{F}_{\mathsf{BC}},\mathcal{F}_{\mathsf{NIZK}}}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ from $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ is upper bounded by

$$\begin{aligned}&\mathsf{Adv}^{\text{unforge}}_{\mathcal{B}_1,\mathsf{SoRC}}(\lambda) + p_u \cdot \mathsf{Adv}^{\text{dl}}_{\mathcal{B}_2}(\lambda) + p_u \cdot \mathsf{Adv}^{\text{d-sddh}}_{\mathcal{B}_3}(\lambda)\\ &+(p_a + p_c) \cdot \mathsf{Adv}^{\text{class-hiding}}_{\mathcal{B}_4,\mathsf{SoRC}}(\lambda) + (p_a + p_c) \cdot \mathsf{Adv}^{\text{ind-cpa}}_{\mathcal{B}_5,\mathrm{ElGamal}}(\lambda).\end{aligned}$$

Appendix H
Other Distributions for Normal Case

Recall that we use log-normal distribution to model the counterparty delay in the main body. For fair comparison, we explore half-normal and exponential distributions to model the delay in this section. As shown in Figure 23, for both distributions, the mode is fixed to be 0. A larger parameter σ in the half-normal distribution or a larger mean value in the exponential distribution indicates less reliable counterparties, leading to a heavier tail in the delay distribution.

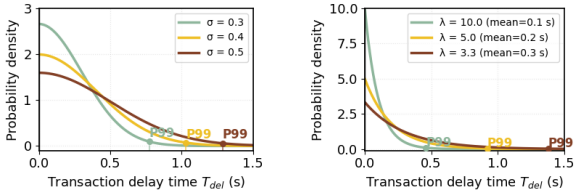


Fig. 23: Half-normal distribution and exponential distribution. The P99 marker indicates the threshold where the cumulative probability reaches 99%.

Following the same evaluation methodology for PEReDi and PulpoPay in the normal case, the results are shown in Figure 24. For half-normal distribution, we evaluate the total processing time of 200 payments by varying σ from 0.1 to 0.5. For exponential distribution, we vary mean value from 0.1 s to 0.3 s. Figure 24 shows that the processing time of the user in PEReDi is significantly affected by the parameter σ or mean value, while PulpoPay maintains constant processing time. The above exploration of distributions shows that the performance gap between PulpoPay and PEReDi is distribution-agnostic and arises solely from whether the payment is decoupled from the counterparty's responsiveness.

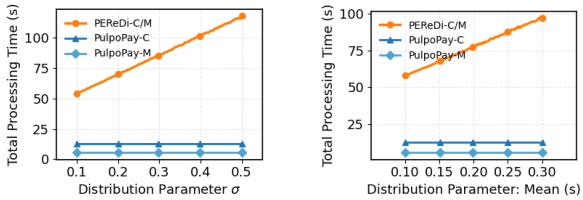


Fig. 24: Processing time for $n = 200$ in normal case PEReDi and PulpoPay under half-normal and exponential distributions.

Appendix I
System Evaluation

Beyond the user-level evaluation in the main body, we also explore the high concurrency performance from the system level. It means that we explore the cost of the whole CBDC system when processing concurrent payments involving multiple users. Below, we share the evaluation method and results.

Dataset collection. To evaluate the system-level performance, we construct a transaction topology for our experiment. In such a topology, the involved users are represented as nodes, and a directed edge from node A to node B indicates a payment from A to B. Since CBDC data are private and not disclosed to the public, we make use of available public data from similar systems to simulate the transaction topology:

- First, we obtain the Ethereum transaction data[1] and filter the transactions to consider only those with a gas fee of exactly 21,000. Transactions exceeding this threshold are commonly associated with smart contract execution and thus are not representative of standard payments.
- Second, to better reflect traditional cash-based payment systems, we remove several super nodes specific to the Ethereum setting, such as BinanceWallet and Nanopoo1, which involve 57.60% of all transactions. Otherwise, the simulated topology containing several super nodes would significantly favor PulpoPay, as its design particularly benefits from handling multiple concurrent transactions involving the same user to amortize costs. Such an extreme topology may lead to an unfair evaluation of PEReDi.
- Third, for simulating tens of thousands of concurrent payments, we ignore the time interval of these transactions to assume they occur simultaneously.

Thereafter, a transaction topology consisting of 12,261 nodes and 12,422 edges is available. In this topology, the nodes involved in a single payment account for 68.55% of all nodes, and the largest node accounts for 16.88% of all payments. Based on this topology, we then design evaluation methods for PEReDi and PulpoPay.

Evaluation method and result. With the above dataset, we consider the optimistic case and normal case for PEReDi and

[1] https://www.kaggle.com/datasets/paul92s/ethereum-data

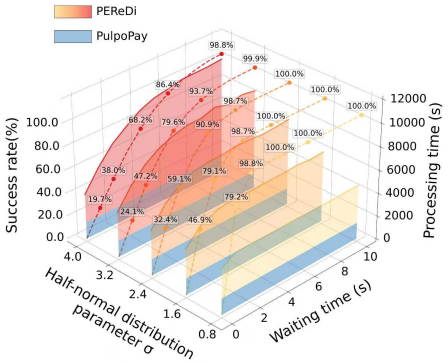


Fig. 25: Processing time and success rate for the simulated concurrent payments in normal PEReDi.

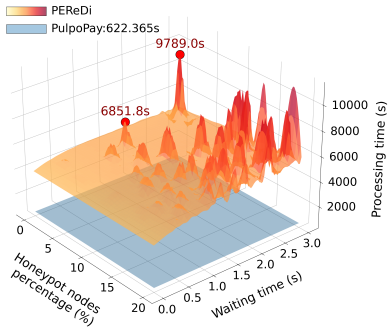


Fig. 26: Processing time comparison for the simulated concurrent payments under honeypot attacks.

PulpoPay at the system-level, adopting definitions analogous to those at the user level. Specifically, each of the 12,261 nodes processes payments either with the optimistic case at the user level or with the normal case. We assume that the maintainer, which usually has substantial local computing power, can seamlessly support simultaneous requests. Then, our evaluation methodology of PEReDi and PulpoPay in the above cases is as follows.

For PEReDi, when processing these concurrent payments, each node is identified as either idle or busy. An idle node is not currently engaged in any payment and is free to handle new payments, while a busy node is unavailable for new payments. In PEReDi, a payment can be carried out only if the connected nodes along an edge are both idle. Therefore, only edges without joint nodes can be processed concurrently. (1) In the optimistic case, the processing time for 12,422 edges can be evaluated by simulating the process flow outlined above. The communication cost of the system is 12,422 times that of a single payment. (2) In the normal case, the processing time is evaluated by further considering the half-normal distribution parameter σ and the waiting time.

For PulpoPay, each node in the transaction topology processes the d-deposit-p-payment transaction, where d is the number of coins for deposit and p is the number of coins for payment. This operation is independent of the state of the counterparties, allowing all nodes to handle payments concurrently. Furthermore, in PulpoPay, even if the receiver of one payment is focusing on other payments, the sender can also send the coin message to the receiver and finish its payment procedure. Essentially, this feature minimizes the sequential relationship between concurrent transaction requests. As a result, the processing time of PulpoPay for 12,422 edges is actually the time it takes for the last node to complete its payment procedure. The communication cost for 12,422 edges is the sum of the communication costs of all d-deposit-p-payment transactions.

With the evaluation method, the result is shown in Figure 25. Furthermore, we highlight an interesting finding. That is, at the system level, we observe the normal PEReDi may be vulnerable to the *honeypot attacks*. Specifically, a malicious honeypot node may attract many other users to conduct payments with it (i.e., as a merchant via exaggerated discounts or as a customer via pretending to purchase from many merchants), and then become unresponsive. Such an attack can be launched instantly and at virtually no cost.

To demonstrate the impact, we randomly select nodes as honeypot nodes (Figure 26). The processing time in PEReDi is highly sensitive to the node's degree; for instance, if a "hub" node is among the 2.5% honeypot nodes, the system's latency escalates to 9789s—15$\times$ that of PulpoPay. This confirms that the unresponsiveness of a single hub in PEReDi can paralyze a significant portion of the network due to synchronous locking. In contrast, PulpoPay's asynchronous coin-based design effectively mitigates such cascading delays through performance isolation.