PEReDi: Privacy-Enhanced, Regulated and Distributed Central Bank Digital Currencies

Amirreza Sarencheh, Aggelos Kiayias, and Markulf Kohlweiss*

The University of Edinburgh, and IOG
Edinburgh, UK
firstname.lastname@ed.ac.uk

Abstract. Central Bank Digital Currencies (CBDCs) aspire to offer a digital replacement for physical cash and, as such, must address two fundamental yet conflicting requirements. On the one hand, they should be *private* to prevent the emergence of a financial "panopticon." On the other hand, they must be *regulation-friendly*, facilitating threshold-limiting, tracing, and counterparty auditing functionalities necessary for compliance with regulations such as Know Your Customer (KYC), Anti-Money Laundering (AML), and Combating the Financing of Terrorism (CFT), as well as financial stability considerations.

In this work, we propose PEReDi, a new asynchronous model for CBDCs and present an efficient construction that, for the first time, simultaneously addresses these challenges in full. Moreover, recognizing the necessity of avoiding a *single point of failure*, our construction is distributed to ensure that all its properties remain intact even when a bounded number of entities are corrupted by an adversary. Achieving all the above properties efficiently is technically involved; among others, our construction employs suitable cryptographic tools to thwart man-in-the-middle attacks, introduces a novel traceability mechanism with significant performance gains over previously known techniques, and, perhaps surprisingly, shows how to obviate Byzantine agreement or broadcast from the optimistic execution path of a payment, something that results in an essentially optimal communication pattern and minimal communication overhead. We demonstrate the efficiency of our payment system by presenting detailed computational and communication cost analyses.

Beyond "simple" payments, we also discuss how our scheme can support one-off large transfers while complying with Know Your Transaction (KYT) disclosure requirements. Our CBDC concept is expressed and realized within the Universal Composition (UC) framework, providing a modular and secure way for integration into a broader financial ecosystem.

Keywords: Privacy, Anonymity, Regulatory Compliance, CBDC, Cryptography, KYC, AML, CFT, Universal Composition.

* The author order follows the Blockchain Technology Laboratory policy with junior and corresponding author Amirreza Sarencheh and senior authors Aggelos Kiayias and Markulf Kohlweiss.

Table of Contents

1 Introduction

The development of cryptocurrencies provided a strong motivation for the development of "central bank digital currency" (CBDC) systems. A CBDC is *central bank money* but more widely accessible and transferable than central bank reserves and banknotes (see e.g., Bank of England [45] for an overview of the basic principles of such systems). This type of money can also be interest bearing (with a different rate than that on reserves) [14] and has a different operational structure than other forms of central bank money [63]. It was early on observed that CBDCs solve a different problem than general cryptocurrencies such as Bitcoin and/or Ethereum. The first construction that exploited this distinction is RSCoin [35] which was followed by designs explored by a number of central banks [1,32,13]. In such systems the verification of transactions relies on a distributed set of independent authorities (we call them "maintainers"). Such entities are empowered to enforce the monetary and regulatory policies of the system that are dictated by the central bank and regulatory entities. A distinguishing characteristic of CBDC systems compared to cryptocurrencies is that the monetary policy is decoupled from the monetary exchange system. The integrity and soundness of the former remains in the purview of the central bank, while the integrity of the latter is distributed across a set of entities. Therefore, the CBDC system's state is maintained in a distributed manner by the maintainers such that the central bank as well as any regulatory entities can be offline during the time users transact.

A common concern expressed in the context of CBDCs is that, contrary to other forms of central bank money, a CBDC may transform the central bank into a "panopticon" that is continuously aware of all transactional data. Such concerns have also been highlighted in the context of cryptocurrencies. First generation cryptocurrencies such as Bitcoin and Ethereum are only pseudonymous in the sense that a user's transactions are linkable to a (set of) pseudonym(s) that the user can generate. Privacy enhanced cryptocurrencies (e.g., Zerocash [11] or Monero [51]) were developed to hide the value of transactions and offer unlinkable transactions to a certain degree or under plausible assumptions. Note that such systems enjoy a level of anonymity that does not reveal directly any information about payment counterparties and transaction values and, hence, may be attractive and be used for illegal activities such as money laundering, financing terrorism, and so on. As a result, privacy-preserving systems using such techniques can be problematic in settings where comprehensive regulatory compliance is required. CBDCs constitute such setting and hence it is imperative to have built-in features by which, while full anonymity can be offered for most circumstances, at the same time *conditional disclosure* to regulators and law enforcement in case of misbehavior can be facilitated, cf. [5].

Privacy in payment systems can interfere with three main regulatory obligations: (i) Know-Your-Customer (KYC), which requires the positive identification of counterparties before they are able to transact. (ii) Anti-Money Laundering (AML), which requires that sources of funds should be legitimate. (iii) Combating Financing of Terrorism (CFT), which requires that the recipients of funds should not engage in terrorism. To appreciate the way such requirements interfere with privacy, it helps to imagine the set of all payments as a hidden directed graph where vertices correspond to counterparties and edges to payments between them weighted by their value. Using this abstraction, it follows that introducing vertices in the graph should be subject to KYC, while it should be possible to reveal the incoming or outgoing edges to any vertex which is suspected for illicit or terrorism activity, as well as trace selectively particular paths in the graph from source to destination and vice versa to address AML and CFT considerations. Beyond these opening and tracing operations it is widely recognized in the CBDC context, cf. [1,8,13], that it is desirable to restrict both the volume of payments that a particular vertex can make (so that "hoarding" CBDC currency is tempered) as well as limit the amount of value that can be transferred between two counterparties in a single transaction, without triggering additional auditing regarding the funds of the sender (what is referred to as KYT - know your transaction, cf. [3]). Unfortunately,

currently no existing CBDC design offers privacy combined with such "regulation friendly" capabilities.

1.1 Our results

We put forth a model and construction that for the first time addresses all the issues identified above simultaneously. In PEReDi each user has an account which is approved during onboarding (i.e., it undergoes KYC) and can subsequently be issued currency by the central bank (following its monetary policy) as well as receive or transmit funds to other users. Our design approach applies a novel combination of cryptographic primitives and distributed organization that, perhaps surprisingly, shows how we can remove the requirement for (byzantine) agreement or broadcast from the optimistic path of payment execution. PEReDi features an encrypted ledger maintained separately by each maintainer, transactions are identified by transaction identifiers and leave encrypted fingerprints in the ledger of each maintainer that under normal circumstances are completely opaque. Transaction senders and receivers independently update their private accounts, leaving the above traces, while only in the case of a transaction abort the maintainers need to engage in an agreement protocol to ensure consistency. In this way, PEReDi offers a digital equivalent of physical cash: payments do take place with double-spending prevention without anyone in the system becoming aware of the precise value transferred or the counterparties involved. Moreover, both sender and receiver need to engage for the payment, something that prevents "dusting" attacks[1]. At the same time (and contrary to physical cash) the transaction value is subject to constraints in terms of sending and receiving limits of the two counterparties and maximum transaction size, while the counterparties themselves are preconditioned to proper KYC onboarding. Tracing and opening operations are accommodated by the design elements of the encrypted ledgers. Given adequate evidence about suspicious activities of a specific user or a particular transaction (indexed by its unique transaction identifier), the authorities can trace transactions made by that user or reveal the metadata of a given transaction by unlocking the real world identities of the counterparties or the total value transferred. Combining these opening and tracing operations, authorities can identify the labels of specific vertices in the payment graph as well as trace paths of payment from source to destination and vice-versa. We stress that such operations require a quorum of entities to agree and hence cannot be unilaterally invoked by any individual entity hence precluding a single point of failure.

To summarize, our contributions are as follows:

- To the best of our knowledge, this is the first time that a fully privacy-preserving and comprehensively regulation friendly CBDC is modeled formally. Our formal model is in the Universal Composition (UC) setting [24]. This modeling enables the composition of the system as payment infrastructure within larger systems.
- We review the regulatory compliance in the context of payment systems (KYC, AML, CFT, auditing, etc.) and argue how our ideal functionality for CBDCs, $\mathcal{F}_{\mathsf{CBDC}}$, captures such requirements in a privacy preserving setting.
- We put forth a distributed construction Π_{PEReDi} that realizes our CBDC ideal functionality $\mathcal{F}_{\mathsf{CBDC}}$ in an efficient manner based on standard cryptographic assumptions. Notably Π_{PEReDi} demonstrates that neither Byzantine broadcast nor agreement is needed in the optimistic execution path of a payment instance, resulting in an optimal communication pattern and message size in the case when both sender and receiver are online and willing to finalize a payment.
- We introduce a novel simulatable approach for tracing suspicious users in the auditing protocol which is employed for double-spending prevention as well and may be of independent interest as it is more efficient than previously known techniques in the broader

[1] Dusting attacks were observed in 2022 after the ruling of OFAC to blacklist the anonymization service Tornado Cash [55].

context of tracing users in conditionally anonymous payment systems. Moreover, the introduced auditing mechanism does not require Byzantine agreement or broadcast.
- We describe how our efficient CBDC construction Π_{PEReDi} can facilitate additional features such as protocol support for concurrent digital currency issuance by the central bank for different users, aborting transactions, and Know Your Transaction (KYT) operations.

It is worth noting that even though we describe our results in the context of CBDCs, it is immediate that our system can be used to implement any "stablecoin" or more generally fungible digital token which has a centrally managed supply. In such case, the role of the central bank is played by the issuer of the digital token, who is capable to introduce new tokens increasing the supply as determined by the issuer's policy. It is also straightforward to return such tokens to the issuer by sending them to a designated account for that purpose.

In the proceedings version of our paper [43], we assumed that the total number of maintainers required for the system was $D = 3t + 1$ where t represents the maximum number of maintainers that can be corrupted by the adversary. Upon further analysis, we identified a need to revise this assumption — we describe a lower bound on D in Section 4.4, arguing that $5t + 1$ maintainers is necessary to prevent adversary-induced faults in the pessimistic execution path of a payment for any efficient realization of $\mathcal{F}_{\mathsf{CBDC}}$ in the asynchronous setting. In Section 5, we demonstrate that $5t + 1$ is also sufficient. We establish that $5t + 1$ maintainers are necessary in our construction to ensure security in the pessimistic execution path, as the optimistic path ***eliminates*** communication between maintainers entirely to enable *fast* settlement of payments.

1.2 Related works

The first system for anonymous electronic cash was introduced by Chaum [29] and focused on sender anonymity, while disclosing the recipient's identity and the amount transferred. The system also required users to hold information linear in the number of coins they possess, a performance consideration that was addressed in follow-up work [27,20]. Regarding the problem of revealing the transaction value to the bank, transferable e-cash [23,7] introduced a mechanism for double-spending prevention. In this mechanism, coins can be transferred to various users without communicating with the bank. Hence, coins expand in size depending on how frequently they are used, which might be inefficient for retail payments. Additionally, in these schemes, coins are distinguishable based on the number of transfers performed. Camenisch et al. [21] proposed a token-based e-payment solution in which the bank can enforce simple rules such as per-user payment limits. The privacy of senders of transactions is preserved; nonetheless, the recipient's identity and payment amount are leaked.

Zerocash [11] represents a UTxO-based anonymous payment system characterized by a full anonymity set. This set encompasses every coin within the system. Similarly, in the context of PEReDi's model of full anonymity, the anonymity set comprises all accounts within the system. Notably, in both systems, the sender's transaction size is constant $\mathcal{O}(1)$ (unaffected by the size of the anonymity set). Garman et al. [39] addressed how regulation rules could be enforced in constructions like Zerocash [11]. The disadvantage of payment systems similar to the Zerocash approach is that they result in privacy-preserving transactions that are unsuitable for resource-constrained users. Users must prove knowledge of the path of a transaction output in a Merkle tree; hence, they must maintain an up-to-date version of this tree (to achieve full anonymity at any given time). Moreover, users are required to download the entire ledger and decrypt all transactions to determine whether they are recipients of transactions. Instead, in our construction, there is no need to download the ledger. The necessity for users to be up-to-date with the whole ledger makes distributed blockchain-ledger-based constructions less efficient than our scheme, which is based on signatures of distributed (known) maintainers on the updated account of each user.

Monero [51], which is a UTxO-based anonymous payment system, uses ring signatures, where the anonymity set is constituted by a group of public keys. The sender proves possession of the secret key corresponding to one of the public keys within the ring (without revealing which one). Monero does not provide full anonymity. However, it could support larger anonymity sets at the cost of sacrificing efficiency, unlike PEReDi's approach, where the underlying zero-knowledge proof is independent of the anonymity set size.

Danezis and Meiklejohn [35] introduced RSCoin, a central bank currency framework built around an efficient broadcast mechanism. In RSCoin, the central bank delegates the responsibility of verifying transactions to a set of entities called mintettes. Unlike traditional cryptocurrency miners, in their framework, mintettes are known and may eventually be held responsible for any misconduct. RSCoin focuses on the scalability of broadcast rather than privacy or regulatory compliance. Performance was improved further with the Fastpay design [9], even though privacy remained unaddressed.

Wüst et al. [62] proposed an anonymous payment scheme called PRCash, in which transactions are verified in a distributed manner. It achieves privacy and some degree of regulatory compliance. However, the main drawbacks of PRCash are that it does not provide full anonymity, as validators can link different transactions, and it lacks auditability. Hence, the authorities cannot investigate suspicious transactions or counterparties on demand.

Androulaki et al. [6] introduced a privacy-preserving auditable token management system. Their proposed scheme uses a UTxO model in a permissioned blockchain. In contrast to our construction, which is account-based, they target business-to-business scenarios and do not offer a comprehensive approach to regulatory compliance as we do.

Zether, proposed by Bünz et al. [17], is a privacy-preserving payment design that hides the user balance, transaction value, and sender and receiver identities. Zether is account-based, where each public key holder is associated with an ElGamal encryption of its balance under its public key. The sender generates a transaction by encrypting the value of the transaction under the receiver's public key, the negative value of the transaction under her public key, and zero under some random public keys. These randomly chosen public keys, along with the sender's and receiver's public keys, generate an anonymity set in which the identities of the sender and receiver are hidden. The sender proves in zero knowledge that she has done so correctly (e.g., all encryptions are well-formed). We highlight drawbacks of Zether compared to PEReDi: i) The sender can only initiate one transaction per epoch (each k consecutive blocks form an epoch), which reduces the speed of transaction generation by senders. ii) In Zether, each sender, before generating their zero-knowledge proof, must query the blockchain (at the beginning of each epoch) to obtain their most updated state. This is necessary because, at the end of each epoch, the blockchain rolls over all pending states to permanent ones. If other users have included the sender's public key in their anonymity set, the sender's state will change at the end of the epoch. iii) Moreover, Zether has a much smaller anonymity set of size k because the transaction size, $\mathcal{O}(k)$, is linear in the anonymity set size, and the efficiency of the zero-knowledge proof is also affected by k. The transaction size is upper bounded by the block size. However, the tag space used for double-spending and replay attack prevention in Zether does not grow, as it is reset to empty at the end of each epoch. In PEReDi, the tag space grows (one group element per transaction); however, the tag serves the additional purpose of tracing, eliminating the need to introduce new elements to manage the tracing of malicious users (thus reducing both communication and computation costs by preventing attacks and offering tracing via recording one group element at the same time). iv) Furthermore, there is no mechanism to capture regulatory-related rules in their system design, such as tracing malicious user functionality.

Damgård et al.'s work [34] addressed the problem of balancing accountability with privacy. Nevertheless, their work is in the identity layer for blockchain systems, and they do not study various features necessary for a CBDC system (e.g., currency issuance, transactions between users, financial and regulatory policies, and so on) in their transaction layer framework. The tracing mechanism in [34], for each account generation, requires the account

holder to compute a pseudorandom function PRF using their secret key. There is no concrete implementation for tracing in their work, as they use secure multi-party computation for PRF in a black-box manner. More importantly, the input of PRF is restricted to a range of values, making tracing inherently inefficient, as authorities are supposed to generate the PRF values for *all* possible inputs in the range. In contrast, we achieve tracing complexity per user proportional to the actual number of transactions issued by that specific user.

Wüst et al. [61] introduced Platypus, which is a privacy-preserving and centralized payment system. Platypus relies on a single authority, whereas our scheme is distributed, making it robust against single points of failure with respect to regulation enforcement and capable of functioning even if the central bank is completely offline. Furthermore, our scheme offers encrypted (distributed) ledgers, which allow compliance with regulations like AML and CFT by enabling the set of authorities to trace a malicious user and discover the transfer value and identities of the counterparties in any suspicious transaction. Platypus [61] does not offer such a capability. We stress that it is quite delicate to add efficient tracing and opening mechanisms to a CBDC design, as various attacks—such as man-in-the-middle attacks where the sender's transaction information is not tied to the receiver's identity and vice versa—can occur and should be addressed through careful design and modeling choices, as we do here. Moreover, the security properties of a CBDC system in their work are defined via a game-based approach, which may limit the composability of their construction; cf. [25].

Tomescu et al. [60] introduced a decentralized payment system called UTT. Their construction relies on Byzantine fault-tolerant infrastructure. However, PEReDi obviates Byzantine agreement and Byzantine broadcast from the optimistic execution path of a transaction. Hence, we achieve an essentially optimal communication pattern and communication overhead when transaction participants are honest. In UTT, the receiver of a transaction has to scan all transactions on a ledger—similar to blockchain-ledger-based anonymous payment systems—to successfully receive the currency, which increases the load on users' sides. Regarding regulation enforcement, the amount of money that can be anonymously sent in the UTT setting is limited by a monthly budget. PEReDi, on the other hand, allows for comprehensive regulatory compliance and can also enforce regulations from the recipient's standpoint.

2 Preliminaries

2.1 Notations

In this paper, for uniquely identifying parties, we denote the central bank by B, the user and its key pair by U and $(\mathsf{pk}_\mathsf{U}, \mathsf{sk}_\mathsf{U})$, respectively. The user U also possesses another secret key a used for generating a *per-transaction* tracing tag, denoted by T. The account of U is represented by acc. The notation M_j is used for the j-th maintainer, and $\mathbb{M}$ denotes the set of all maintainers. Each maintainer (e.g., M_j) has two pairs of keys for threshold encryption: $(\mathsf{pk}_{1,j}, \mathsf{sk}_{1,j})$ and $(\mathsf{pk}_{2,j}, \mathsf{sk}_{2,j})$. Additionally, each maintainer M_j has a pair of keys for threshold signature: $(\mathsf{pk}_j, \mathsf{sk}_j)$. We assume $|\mathbb{M}| = D$ and define two threshold values:

- α represents the threshold number of maintainers required for signing transactions on behalf of the central bank and the regulator.
- β represents the threshold number of maintainers required for the *auditing* protocol. Maintainers for which β is required are called the *audit committee*.

The set of honest and malicious maintainers is denoted by $\mathbf{H}$ and $\mathbf{C}$, with their associated identifiers (indexes) given by $\mathcal{H}$ and $\mathcal{C}$, respectively. We assume $|\mathbf{C}| = t$. An honest maintainer is represented by M_w, and a malicious maintainer by M_t. Each maintainer M_j maintains a ledger $\mathcal{L}_j$ (local state), which is initially empty. The user record stored in $\mathcal{L}_j$ is denoted by UR. The sender and receiver of a payment are denoted by U_s and U_r, respectively. For example, the key pair of the sender is $(\mathsf{pk}_s, \mathsf{sk}_s)$, and its tracing key is a_s. The transaction value transferred from a sender (B or U_s) to a recipient is denoted by v, and the transaction

identifier by t_{id}. The balance of U is represented by B, while the total sum of all sent and received values of U is denoted by S and R, respectively. The following are regulatory limits:

- B_{max}: Maximum allowed account balance.
- S_{max}: Maximum allowed sum of all sent values.
- R_{max}: Maximum allowed sum of all received values.
- V_{max}: Maximum allowed transaction value.

The transaction counter of a user, incremented for each transaction (*currency issuance* or *payment*), is denoted by x. The statement of the zero-knowledge proof is denoted by $\mathtt{x}$. The notation $\{e_i\}_{i=1}^{N}$ represents a set $\{e_1, \ldots, e_N\}$ of N elements. We use $\mathbb{F}_q$ to denote a field with q elements. PPT stands for *probabilistic polynomial time*. A function negl is said to be *negligible* if, for every positive polynomial p, there exists an integer i_0 such that for all integers $i > i_0$, the following holds: $\mathsf{negl}(i) < \frac{1}{p(i)}$.

We provide the definition of our main building block in the following section. Additional definitions can be found in Appendix B and Appendix C.

2.2 Digital signatures

Definition 1 (Digital signature scheme). *A digital signature scheme $\Gamma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is defined by the following components:*

- *$\mathcal{K}$: A probabilistic algorithm that, given a security parameter λ, outputs a key pair $(\mathsf{sk}, \mathsf{vk})$, where:*
  - *sk: The private signing key used to generate signatures.*
  - *vk: The public verification key used to verify signatures.*
- *$\mathcal{S}$: A (possibly probabilistic) algorithm that, given the private signing key sk and a message $\mu \in \mathcal{M}$, produces a signature $\sigma \in \Xi$. Formally: $\sigma \leftarrow \mathcal{S}_{\mathsf{sk}}(\mu)$, where $\mathcal{M}$ is the message space and Ξ is the signature space.*
- *$\mathcal{V}$: A deterministic algorithm that, given the public verification key vk, a message $\mu \in \mathcal{M}$, and a purported signature $\sigma' \in \Xi$, outputs a binary decision $\beta \in \{0, 1\}$. Formally: $\beta = \mathcal{V}_{\mathsf{vk}}(\mu, \sigma')$, where $\beta = 1$ indicates that σ' is a valid signature for μ under vk, and $\beta = 0$ indicates rejection.*

A digital signature scheme Γ satisfies correctness if, for all key pairs $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{K}(1^\lambda)$ and all messages $\mu \in \mathcal{M}$, the following holds: $\mathcal{V}_{\mathsf{vk}}(\mu, \mathcal{S}_{\mathsf{sk}}(\mu)) = 1$. In other words, any signature σ generated for μ using the private key sk must be accepted as valid by the verification algorithm $\mathcal{V}$ when using the corresponding public key vk.

Definition 2 (EU–CMA Security). *A digital signature scheme $\Gamma = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ is said to be existentially unforgeable under a chosen message attack ($\mathsf{EU\text{-}CMA}$) if, for all probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}CMA}}(\lambda)$ is negligible in the security parameter λ:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}CMA}}(\lambda) \leq \mathsf{negl}(\lambda).$$

The $\mathsf{EU\text{-}CMA}$ experiment $\mathsf{EU\text{-}CMA}_{\Gamma}(\mathcal{A}, \lambda)$ proceeds as follows:

1. *The challenger generates a key pair $(\mathsf{sk}, \mathsf{vk})$ by running $\mathcal{K}(1^\lambda)$ and sends the public verification key vk to the adversary $\mathcal{A}$.*
2. *The adversary $\mathcal{A}$ queries a signing oracle $\mathcal{S}_{\mathsf{sk}}(\cdot)$, providing messages $\mu \in \mathcal{M}$. For each query, the challenger returns the signature $\sigma = \mathcal{S}_{\mathsf{sk}}(\mu)$. The challenger records all queried messages in a set $\mathcal{L}$ (the query list).*
3. *Eventually, the adversary $\mathcal{A}$ outputs a forgery attempt, consisting of a message-signature pair (μ^*, σ^*), where $\mu^* \in \mathcal{M}$ and $\sigma^* \in \Xi$.*
4. *The adversary $\mathcal{A}$ wins if the following conditions are satisfied:*
   - *$\mu^* \notin \mathcal{L}$ (i.e., μ^* was not queried to the signing oracle), and*
   - *$\mathcal{V}_{\mathsf{vk}}(\mu^*, \sigma^*) = 1$ (i.e., σ^* is a valid signature for μ^* under vk).*

The adversary's advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}CMA}}(\lambda)$ *in this experiment is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EU\text{-}CMA}}(\lambda) = \Pr[\mathsf{EU\text{-}CMA}_{\Gamma}(\mathcal{A}, \lambda) = 1].$$

Definition 3 (Pointcheval-Sanders signature scheme). *The Pointcheval-Sanders signature scheme [53] is defined as a tuple of algorithms* $\mathsf{PS} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*, operating over a bilinear group setup. The algorithms are defined as follows:*

- $(\mathsf{sk}, \mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda, q)$*: Run a pairing group setup algorithm* $\mathcal{G}(1^\lambda)$ *to generate:*

$$\mathsf{par} = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g}),$$

where $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_T$ *is a bilinear map,* p *is a prime order, and* $g \in \mathbb{G}$*,* $\tilde{g} \in \tilde{\mathbb{G}}$ *are generators.*
The secret key is:

$$\mathsf{sk} = (x, \{y_i\}_{i=1}^q), \quad x, y_i \xleftarrow{\$} \mathbb{Z}_p.$$

The public key is:

$$\mathsf{pk} = (\mathsf{par}, \alpha, \{\beta_i\}_{i=1}^q), \quad \alpha = \tilde{g}^x,\ \beta_i = \tilde{g}^{y_i} \text{ for } i = 1, \ldots, q.$$

- $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}, \{m_i\}_{i=1}^q)$*: To sign a message vector* $\{m_i\}_{i=1}^q \subset \mathbb{Z}_p^q$*, perform the following:*
  1. *Select a random scalar* $r \xleftarrow{\$} \mathbb{Z}_p$*.*
  2. *Compute* $h = g^r$*.*
  3. *Compute the signature component* s*:*

$$s = h^{x + \sum_{i=1}^q y_i m_i}.$$

  Output the signature $\sigma = (h, s)$*.*
- $(1, 0) \leftarrow \mathsf{Verify}(\mathsf{pk}, \sigma, \{m_i\}_{i=1}^q)$*: To verify a signature* $\sigma = (h, s)$ *on a message vector* $\{m_i\}_{i=1}^q$*:*
  1. *Check that* $h \neq 1$*.*
  2. *Verify the pairing equation:*

$$e(h, \alpha \cdot \prod_{i=1}^{q} \beta_i^{m_i}) = e(s, \tilde{g}).$$

  If both checks hold, output 1 *(accept); otherwise, output* 0 *(reject).*

2.2.1 Threshold blind signature Coconut [59] is an optional declaration credential construction supporting distributed threshold issuance based on the Pointcheval-Sanders signature [53] (Definition 3). Unlinkable optional attribute disclosures, as well as public and private attributes, are supported by the framework of [59], even when some of the issuing authorities are malicious or offline. Recently, Rial et al. [54] analyzed the security properties of Coconut [59] by introducing an ideal functionality that captures all the security properties of a threshold blind signature TBS. They proposed a new construction that builds upon Coconut with a few modifications to realize the TBS ideal functionality. These modifications involve changes to the issuing of blind signatures and the signature showing process.

Informally, the TBS scheme satisfies unforgeability, unlinkability, and blindness. Unforgeability guarantees that a corrupted user cannot convince an honest verifier that it has a valid signature if, in fact, it does not. Blindness ensures that a corrupted signer cannot learn any information about the message m during the execution of the signing protocol ($\mathsf{IssueSig}$), except that m satisfies a predicate. Unlinkability ensures that a corrupted signer or verifier cannot learn anything about the message m, beyond the fact that it satisfies a predicate,

nor can they link the execution of randomizing a signature (ProveSig) with either another execution of ProveSig or with the execution of IssueSig.

We use the improved version of Coconut [59] introduced in [54] with some modifications (such as modeling the communication between the user and the signing maintainer, and embedding the zero-knowledge proofs needed throughout the TBS scheme into proofs generated in our construction Π_{PEReDi}, as described in Sec. 4.2) as a TBS scheme.

Definition 4 (Threshold blind signature). *The threshold blind signature scheme* $\mathsf{TBS} = (\mathsf{TBS.KeyGen}, \mathsf{IssueSig}, \mathsf{TBS.Agg}, \mathsf{ProveSig}, \mathsf{VerifySig})$ *consists of the following algorithms.*

- $(\{(\mathsf{pk}_j, \mathsf{sk}_j)\}_{j=1}^{D}, \mathsf{pk}) \xleftarrow{\$} \mathsf{TBS.KeyGen}(1^\lambda, D, \alpha)$ [2]
  - *Run* $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}) \leftarrow \mathcal{G}(1^\lambda)$ *and pick* q *random generators* $\{h_\tau\}_{\tau=1}^{q} \subset \mathbb{G}$. *Set the parameters:*
$$par = (p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\tau\}_{\tau=1}^{q}).$$
  - *Choose* $(q+1)$ *polynomials* $(v, \{w_\tau\}_{\tau=1}^{q})$ *of degree* $(\alpha - 1)$ *with random coefficients in* $\mathbb{Z}_p$. *Set:*
$$(x, \{y_\tau\}_{\tau=1}^{q}) = \big(v(0), \{w_\tau(0)\}_{\tau=1}^{q}\big).$$
  - *For* $j = 1$ *to* D*:*
    - *Set the secret key* sk_j *of each signer* M_j *as:*
$$\mathsf{sk}_j = \big(x_j, \{y_{j,\tau}\}_{\tau=1}^{q}\big) = \big(v(j), \{w_\tau(j)\}_{\tau=1}^{q}\big).$$
    - *Set the verification key* pk_j *of each signer* M_j *as:*
$$\mathsf{pk}_j = \big(\tilde{\alpha}_j, \{\beta_{j,\tau}, \tilde{\beta}_{j,\tau}\}_{\tau=1}^{q}\big) = \big(\tilde{g}^{x_j}, \{g^{y_{j,\tau}}, \tilde{g}^{y_{j,\tau}}\}_{\tau=1}^{q}\big).$$
  - *Set the aggregate verification key* pk *as:*
$$\mathsf{pk} = \big(par, \tilde{\alpha}, \{\beta_\tau, \tilde{\beta}_\tau\}_{\tau=1}^{q}\big) = \big(par, \tilde{g}^{x}, \{g^{y_\tau}, \tilde{g}^{y_\tau}\}_{\tau=1}^{q}\big).$$
- IssueSig *consists of the following three sub-algorithms:* $(\mathsf{PrepareBlindSign}, \mathsf{BlindSign}, \mathsf{Unblind})$.
  - $(\mathsf{acc}^{\mathfrak{B}}, \pi_s, \{o_\tau\}_{\tau=1}^{q}) \xleftarrow{\$} \mathsf{PrepareBlindSign}(\mathsf{acc}, \phi)$
    - *Parse* $\mathsf{acc} = \{m_\tau\}_{\tau=1}^{q} \subset \mathbb{Z}_p$. *Pick a random value* $o \xleftarrow{\$} \mathbb{Z}_p$ *and compute:*
$$\mathsf{com} = g^o \prod_{\tau=1}^{q} h_\tau^{m_\tau}.$$
      Send com *to* $\mathcal{F}_{\mathrm{RO}}$ [3] *and receive* h *from* $\mathcal{F}_{\mathrm{RO}}$.
    - *Compute commitments to each of the messages. For* $\tau \in \{1, \ldots, q\}$, *pick random* $o_\tau \xleftarrow{\$} \mathbb{Z}_p$ *and compute:*
$$\mathsf{com}_\tau = g^{o_\tau} h^{m_\tau}.$$
    - *Compute a NIZK proof* π_s *for the following relation:*
$$\pi_s = \mathsf{NIZK}\left\{ \begin{array}{l} \big(\{m_\tau\}_{\tau=1}^{q}, o, \{o_\tau\}_{\tau=1}^{q}\big) : \\ \quad \mathsf{com} = g^o \prod_{\tau=1}^{q} h_\tau^{m_\tau} \wedge \quad \{\mathsf{com}_\tau = g^{o_\tau} h^{m_\tau}\}_{\tau=1}^{q} \wedge \quad \phi\big(\{m_\tau\}_{\tau=1}^{q}\big) = 1 \end{array} \right\}.$$
    - *Set:*
$$\mathsf{acc}^{\mathfrak{B}} = \big(\mathsf{com}, \{\mathsf{com}_\tau\}_{\tau=1}^{q}, h\big).$$

[2] This algorithm can be replaced by a distributed key generation protocol using, e.g., [36,42,40,26].

[3] $\mathcal{F}_{\mathrm{RO}}$ denotes the functionality of a random oracle, which provides a truly random response from an output domain for every unique request.

– $\sigma_j^{\mathfrak{B}} \leftarrow \mathsf{BlindSign}(\mathsf{sk}_j, \phi, \pi_s, \mathsf{acc}^{\mathfrak{B}})$
  * *Send* com *to* $\mathcal{F}_{\mathrm{RO}}$ *and receive* h' *from* $\mathcal{F}_{\mathrm{RO}}$. *Abort if* $h \neq h'$ *or* π_s *is not correct.*
  * *Compute:*

$$c_j = h^{x_j} \prod_{\tau=1}^{q} \mathsf{com}_\tau^{y_{j,\tau}}.$$

  Set the blind signature share as:

$$\sigma_j^{\mathfrak{B}} = \big(h, c_j\big).$$

– $\sigma_j \leftarrow \mathsf{Unblind}(\{o_\tau\}_{\tau=1}^{q}, \sigma_j^{\mathfrak{B}})$
  * *Parse* $\sigma_j^{\mathfrak{B}}$ *as* (h', c_j). *Abort if* $h \neq h'$.
  * *Compute:*

$$\sigma_j = (h, s_j) = \big(h, c_j \prod_{\tau=1}^{q} \beta_{j,\tau}^{-o_\tau}\big).$$

  * *Abort if the following pairing equation does not hold:*

$$e\big(h, \tilde{\alpha}_j \prod_{\tau=1}^{q} \tilde{\beta}_{j,\tau}^{m_\tau}\big) = e(s_j, \tilde{g}).$$

- $\sigma_{\mathbb{M}} \leftarrow \mathsf{TBS.Agg}(\{\sigma_j\}_{j=1}^{\alpha}, \mathsf{pk})$
  – *Let* $E \subseteq [1, D]$ *be a set of* α *indices of signers in* $\mathbb{M}$.
  – *For all* $j \in E$, *evaluate at 0 the Lagrange basis polynomials:*

$$l_j = \prod_{\substack{i \in E \\ i \neq j}} \frac{i}{i - j} \bmod p.$$

  – *For all* $j \in E$, *take* $\sigma_j = (h, s_j)$ *and compute the aggregated signature:*

$$\sigma_{\mathbb{M}} = (h, s) = \big(h, \prod_{j \in E} s_j^{l_j}\big).$$

  – *Abort if the following pairing equation does not hold:*

$$e\big(h, \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_\tau^{m_\tau}\big) = e(s, \tilde{g}).$$

- $(\sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \pi_v, \varphi) \xleftarrow{\$} \mathsf{ProveSig}(\varphi, \sigma_{\mathbb{M}}, \{m_\tau\}_{\tau=1}^{q}, \mathsf{pk})$
  – *Parse* $\sigma_{\mathbb{M}}$ *as* (h, s), *pick* $r \xleftarrow{\$} \mathbb{Z}_p$ *and* $r' \xleftarrow{\$} \mathbb{Z}_p$.
  – *Compute:*

$$\sigma_{\mathbb{M}}^{\mathsf{int}} = (h', s') = \big(h^{r'}, s^{r'}(h')^r\big).$$

  – *Parse* acc *as* $\{m_\tau\}_{\tau=1}^{q}$ *and compute:*

$$\kappa = \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^r.$$

  – *Compute the NIZK proof* π_v *for the following relation:*

$$\pi_v = \mathsf{NIZK}\big\{(\{m_\tau\}_{\tau=1}^{q}, r) : \kappa = \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^r \wedge \varphi(\{m_\tau\}_{\tau=1}^{q}) = 1\big\}.$$

  – *Set:*

$$\sigma_{\mathbb{M}}^{\mathsf{Rnd}} = (\sigma_{\mathbb{M}}^{\mathsf{int}}, \kappa) = \big((h', s'), \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^r\big).$$

- $(1, 0) \leftarrow \mathsf{VerifySig}(\sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \pi_v, \varphi, \mathsf{pk})$
  – *Parse* $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ *as* $(\sigma_{\mathbb{M}}^{\mathsf{int}}, \kappa)$ *and abort with output 0 if:*

$$h' = 1 \quad or \quad e(h', \kappa) \neq e(s', \tilde{g}).$$

  – *Verify* π_v. *Output 0 if the proof is not correct. Otherwise, output 1.*

3 CBDC desiderata and modeling

We abstract a CBDC system into three separate classes of entities: the central bank, a set of maintainers (e.g., commercial banks and financial institutions), and users. Role separation is an important element in CBDC design; cf. [1]. The description of these roles, together with the relevant assumptions made about them, is as follows.

- *Central bank:* The central bank issues the digital currency and is responsible for monetary policy. The monetary supply at any given time is in the purview of the central bank. However, the state of all users' accounts is not under its control. Moreover, due to the potential threat of mass surveillance [32], the central bank is also not trusted for privacy, i.e., it has no ability to deanonymize the sender or recipient of a transaction or reveal the transferred values associated with a specific transaction. Finally, the central bank is not responsible for enforcing the regulatory rules that govern payments. We refer to [13] and [32] for more context on the role of central banks.
- *Maintainers:* The authority to validate transactions and facilitate various auditing operations needed for regulatory compliance is delegated to a number of approved institutions that we call the maintainers. As a result, the central bank and regulator are not required to be active in any of the system's day-to-day operations (except for issuing currency in the case of the former). The maintainers share the state of the system and are responsible for continuously updating it as users issue transactions. In a real-world deployment, maintainers can be organizations with an existing connection to the central bank, such as commercial banks, financial institutions, etc. Note that, contrary to, e.g., miners in a cryptocurrency blockchain, the set of all maintainers is public and known to all network participants. The basic properties of the system, such as integrity, regulatory compliance, and privacy of transactions, emanate from the actions of the maintainers. We note that the system's security and liveness objectives will be met as long as the adversary controls less than a certain threshold number of maintainers. In any financial system, various operations are subject to regulatory rules. Examples of relevant entities developing and/or enforcing such rules include the Financial Conduct Authority (FCA) in the UK and the Securities and Exchange Commission (SEC) in the US. One important aspect of regulatory compliance is KYC; in our CBDC system abstraction, we assume maintainers are responsible for onboarding users to the system, i.e., all accounts in the system are introduced subject to the approval of the maintainers.
- *Users and payment interface providers (PIPs):* As in any digital currency system, in a CBDC system, users can act as either the sender (a.k.a. buyer, payer, or customer) or the recipient (a.k.a. seller, payee, or merchant) of digital currency in a transaction. Users of the currency can be private individuals or organizations. Note that users engage with the system through software and/or hardware provided by a PIP. The distinction between users and PIPs will not be essential for our analysis and modeling, and we will not pursue it further. We assume that any number of users in the system are untrusted, i.e., they may behave maliciously against honest users or other system entities. The privacy of payments should be satisfied between an honest sender and an honest receiver in a transaction.

3.1 CBDC security requirements

In this section, we informally define security requirements that will be captured by our CBDC ideal functionality. Note that the CBDC system should be resilient against broad types of attacks (e.g., Sybil attacks, man-in-the-middle attacks, etc.); however, the focus of this section is on explaining requirements that are more specific to payment systems and CBDCs. These are as follows.

- *Financial and regulatory integrity.* No one should be able to update the account of another user. Furthermore, the currency in circulation, or the amount of CBDC used to

conduct transactions between consumers and businesses, does not change as the system evolves over time except when the central bank decides to create new money (digital currency). Double-spending prevention is a crucial requirement for any payment system. A specific balance of a user should not be used in two transactions without being updated each time. In addition, after a successful payment between two users, the accounts of both should be updated correctly, considering all parameters included in users' accounts for the purpose of checking financial and regulatory rules.

- *Comprehensive regulatory compliance.* This term means achieving all the following four items at the same time.
  - *Balance limit:* This limits the amount of funds that a particular user can possess in a specific period of time. The Bank of England [1] and a report from several central banks detailing the principles, motivations, and risks of CBDC [13] have mentioned that balance limits can help prevent bank runs and evasion of wealth tax. Moreover, the Bank of England [1] and the European Central Bank [12] have addressed that to manage the implications of a CBDC for financial stability, limits on how much CBDC any individual can hold are necessary.
  - *Receiving and sending limit:* This limits the amount of funds a particular user can receive or send in a specific period of time. The sent and received amounts should not exceed a predefined threshold. The European Central Bank [8] and several central banks [13] have mentioned that limiting receiving and sending values can help achieve AML and prevent tax evasion.
  - *Transaction value limit and KYT:* Reporting requirements and disclosure of the source of funds for large-value transactions are typically required (e.g., in the US, filing a report is required for transactions in cash exceeding $\$10,000$). To reflect this, we impose a limit on the value of each transaction. Furthermore, we discuss how it is possible to comply with more complex KYT policies, where users should disclose additional information for large-value transactions.
  - *Auditability:* In cases of suspicious activities, additional auditing actions are needed (e.g., filing suspicious activity reports called SARs [2]). The auditing functionality has two components:
    - *Privacy revocation:* Given an anonymous transaction, authorities can reveal the real-world identities of the involved parties and the transferred value of that transaction.[4]
    - *Tracing:* Given the real-world identity of a user, authorities can trace anonymous payments in which the user has engaged (as a sender or recipient).
- *Full privacy.* This property means achieving all the following three items at the same time.
  - *Identity privacy:* For any given transaction, the real-world identities of either the sender or the receiver cannot be revealed (except when auditing). Furthermore, given the identity of a specific user, no one can find the transactions in which the user has been involved as a sender or receiver.
  - *Transaction privacy:* The transferred value from the sender to the recipient cannot be revealed (except when auditing). Given a specific amount of transferred value, no one can find the transactions that match that same (or related) value. Only the sender and recipient should know the value of the transaction. Moreover, the account information of users (e.g., the sum of all sent and received values) is hidden from all network entities.
  - *Full unlinkability:* This consists of two parts, as follows.

[4] To improve regulatory compliance, we recognize the existence of more general rules that are subjective and cannot be formally captured. To address these, we encrypt transaction information, allowing decryptability and decision-making based on external information and additional evidence (*Privacy revocation*). This approach supports subjective decision-making. It is important to note that this encryption feature is part of a modular design. If deemed inappropriate, it can be removed without affecting other system functionalities.

  - ∗ *User unlinkability:* Given an anonymous payment, knowing the real-world identities of the sender or receiver should not make it possible to link the sender's or receiver's other transactions to the given transaction.
  - ∗ *Transaction unlinkability:* Given a transaction, it should not be possible to link any past transaction that resulted in the possession of the funds used by the current transaction.
- *Accountability.* When a user makes a payment, they should not be able to deny it later—there is an obligation to accept the responsibilities that come with a finalized transaction.

3.2 CBDC formal model

We formalize the objectives of a CBDC system as an ideal functionality in the Universal Composition framework [24]. The central bank digital currency scheme consists of six main sub-protocols: *user registration, currency issuance, payment, abort transaction, privacy revocation*, and *tracing*. The last two are collectively referred to as *auditing*. Valid transactions are recorded in the ledger $\mathcal{L}$ of each maintainer M. Hence, there is a history of all verified transactions accessible by anyone who is permissioned to audit private transactions. $\mathcal{F}_{\mathsf{CBDC}}$ is parameterized by $D, t, V_{\mathsf{max}}, B_{\mathsf{max}}, S_{\mathsf{max}}$, and R_{max}, where $D = 5t+1$ holds. The functionality $\mathcal{F}_{\mathsf{CBDC}}$ maintains the following tables and mappings:

- $T(\mathsf{U})$ outputs 0 if U has not been traced and 1 if it has been traced. Initially, $T(\mathsf{U}) \leftarrow \perp$, meaning that for non-registered users, $T(\mathsf{U})$ outputs $\perp$.
- Users to their accounts' state: $W = (B, S, R, x) \leftarrow \mathbb{K}(\mathsf{U})$. Initially, $\mathbb{K}(\mathsf{U}) \leftarrow \perp$.
- $U(\mathsf{U})$ outputs pid if the user U has an ongoing transaction with pid. Once the transaction is finalized (in the real world, the user receives α valid signature shares on its new account), $U(\mathsf{U})$ is set to $\perp$, meaning that the user is in the Idle state and can start a new transaction.
- Payment identifiers pid to transaction identifiers t_{id}: $t_{\mathsf{id}} \leftarrow P(\mathsf{pid})$.
- Set of maintainers who engage in a specific transaction whose identifier is t_{id}: $\mathcal{M}(t_{\mathsf{id}})$.
- Users to their most recent transaction metadata and transaction identifier: $(\mathsf{U}_s, \mathsf{U}_r, t_{\mathsf{id}}, v) \leftarrow \mathsf{Tid}(\mathsf{U})$, where $\mathsf{U} = \mathsf{U}_s$ or $\mathsf{U} = \mathsf{U}_r$, or $(\mathsf{B}, \mathsf{U}, t_{\mathsf{id}}, v) \leftarrow \mathsf{Tid}(\mathsf{U})$. Initially, $\mathsf{Tid}(\mathsf{U}) \leftarrow \perp$.
- Transaction identifiers to transaction metadata: $(\mathsf{U}_s, \mathsf{U}_r, v) \leftarrow \mathsf{Rvk}(t_{\mathsf{id}})$.
- Users to all their transaction identifiers and their role in each of them: $\{t^{\tau}_{\mathsf{id}}, \mathsf{role}^{\tau}\}^{x}_{\tau=1} \leftarrow \mathsf{Trc}(\mathsf{U})$.

We note that session identifiers are of the form $\mathsf{sid} = (\mathsf{B}, \mathbb{M}, \mathsf{sid}')$ such that $\mathbb{M} = \{\mathsf{M}_j\}^{D}_{j=1}$. Initially, $\mathsf{init} \leftarrow 0$, where $\mathsf{init} \in \{0, 1\}$. At the end of *initialization*, init is set to 1. Afterwards, at the beginning of all parts of the functionality (namely, *user registration, currency issuance, payment, abort transaction, privacy revocation*, and *tracing*), it is checked whether init has been set to 1. If it has not been set to 1, $\mathcal{F}_{\mathsf{CBDC}}$ ignores the received message. In $\mathcal{F}_{\mathsf{CBDC}}$, by sending a message m to $\mathbb{M}$ via delayed output, we mean the following: $\mathcal{F}_{\mathsf{CBDC}}$ provides m and the unique identifiers of all maintainers in the set $\mathbb{M}$ to the ideal-world adversary $\mathcal{A}$. $\mathcal{F}_{\mathsf{CBDC}}$ lets $\mathcal{A}$ decide the order of maintainers in the set $\mathbb{M}$ who receive the message m. Additionally, it can delay the message delivery or prevent the message from being delivered.

In more detail, the components of our functionality are as follows.

Initialization. This step ensures that the relevant parties (B and all D maintainers $\mathbb{M}$) have been activated. The functionality maintains a record of all parties that have been initialized.

User registration. During the registration phase, a user U must have their account ratified by the system. If the user U has already been registered by the maintainers $\mathbb{M}$, they cannot be registered again. As is common in the Universal Composition setting, we allow the adversary $\mathcal{A}$ to communicate and potentially block registration (i.e., we do not model denial-of-service attacks). The balance and regulation-related information of user U are initialized as follows: the balance B, the sum of all sent values S, the sum of all received

values R, and the transaction counter x are all set to zero. The maintainers are notified upon each successful user registration.

Currency issuance. In the *currency issuance* process, unlike *payment*, only the central bank B is permitted to act as the payer, and no limits are imposed on the funds the central bank possesses.[5] First, the functionality $\mathcal{F}_{\mathsf{CBDC}}$ verifies whether the recipient of the digital currency, U, is a valid registered user in the system. This means that if U has not been registered by the maintainers $\mathbb{M}$, it cannot obtain any digital currency. The functionality enforces regulatory restrictions: $B+v \leq B_{\mathsf{max}}$, and $R+v \leq R_{\mathsf{max}}$, where v is the amount of currency issued according to the central bank's instructions. We note that different jurisdictions may impose varying regulatory rules. For instance, restrictions such as capping the value issued by the central bank B ($v \leq V^*_{\mathsf{max}}$) can be easily incorporated. Similarly, the aforementioned constraints, $B+v \leq B_{\mathsf{max}}$ and $R+v \leq R_{\mathsf{max}}$, can be ignored for currency issuance transactions. Notably, since our construction is account-based rather than token-based, modifying or removing such regulatory compliance constraints is relatively straightforward. Currency issuance is not a unilateral action by the central bank B; it also requires activation by the recipient user U. This highlights a key distinction between our setting and blockchain systems: the recipient U must be online during the transaction, making the protocol interactive. The state of the receiver's account is updated after each currency issuance action. As before, the adversary $\mathcal{A}$ may attempt to block the currency issuance process. A successful currency issuance increases the balance of the receiver U by the specified amount v. The transaction value and the identity of the receiver remain hidden from the adversary. The adversary $\mathcal{A}$ must assign a unique transaction identifier t_{id}, and all transaction metadata are stored by the functionality in a table $\mathsf{Rvk}(t_{\mathsf{id}})$. The identifier t_{id} is also recorded in $\mathsf{Trc}(\mathsf{U})$, where U is the recipient.

Functionality $\mathcal{F}_{\mathsf{CBDC}}$, part I: *initialization, registration* and *issuance*

Initialization.

- Upon input $(\mathtt{Init}, \mathsf{sid})$ from party $P \in \{\mathsf{B}, \mathbb{M}\}$: Abort if $\mathsf{sid} \neq (\mathsf{B}, \mathbb{M}, \mathsf{sid}')$. Else, output $(\mathtt{InitEnd}, \mathsf{sid}, P)$ to $\mathcal{A}$. Once all parties have been initialized, set $\mathsf{init} \leftarrow 1$.

User registration.

- Upon receiving a message $(\mathtt{GenAcc}, \mathsf{sid})$ from U: If $\mathbb{K}(\mathsf{U}) = \perp$, output $(\mathtt{GenAcc}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{A}$. Else, ignore.
- Upon receiving $(\mathtt{Ok.GenAcc}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{A}$: Output $(\mathtt{AccGened}, \mathsf{sid}, \mathsf{U})$ to $\mathbb{M}$ via public-delayed output. Output $(\mathtt{AccGened}, \mathsf{sid})$ to U via public-delayed output and set $\mathbb{K}(\mathsf{U}) \leftarrow W = (0,0,0,0)$ and $T(\mathsf{U}) \leftarrow 0$ when delivered.

Currency issuance.

- Upon receiving a message $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ from B: Ignore if $\mathsf{B} \notin \mathsf{sid}$. Else, generate a new pid, and record the tuple $(\mathtt{Iss}, \mathsf{U}, v, \mathsf{pid}, 1)$. If U is corrupted, output $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}, v)$ to $\mathcal{A}$. Else, output $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, v)$ from U: If $\mathbb{K}(\mathsf{U}) = \perp$ or $U(\mathsf{U}) \neq \perp$ or the tuple $(\mathtt{Iss}, \mathsf{U}, v, \mathsf{pid}, 1)$ is not recorded, ignore. Else, retrieve $\mathbb{K}(\mathsf{U}) = W$. If $B+v > B_{\mathsf{max}}$ or $R+v > R_{\mathsf{max}}$, ignore. Else, set $U(\mathsf{U}) \leftarrow \mathsf{pid}$ and retrieve $T(\mathsf{U})$: (a) If $T(\mathsf{U}) = 0$, output $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{A}$. (b) Else, output $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ from $\mathcal{A}$: If already exits a $\mathsf{pid}' \neq \mathsf{pid}$ where $P(\mathsf{pid}') = t_{\mathsf{id}}$ or the tuple $(\mathtt{Iss}, \mathsf{U}, v, \mathsf{pid}, 1)$ is not recorded, ignore. Else, if $P(\mathsf{pid}) = \perp$, set $P(\mathsf{pid}) \leftarrow t_{\mathsf{id}}$. Else, retrieve $P(\mathsf{pid}) = t'_{\mathsf{id}}$, ignore if $t'_{\mathsf{id}} \neq t_{\mathsf{id}}$. Set $\mathsf{Tid}(\mathsf{U}) \leftarrow (\mathsf{B}, \mathsf{U}, t_{\mathsf{id}}, v)$.

[5] *Currency issuance* is one of the main differences between CBDCs and cryptocurrencies (e.g., Bitcoin [50]) or stablecoins (e.g., PARScoin [57]).

- Upon receiving $(\texttt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ from $\mathcal{A}$: Retrieve the tuple $(\texttt{Iss}, \mathsf{U}, v, \mathsf{pid}, b)$. Ignore if $b = 0$ or $P(\mathsf{pid}) = \perp$. Else, retrieve $P(\mathsf{pid}) = t_{\mathsf{id}}$. Set $\mathcal{M}(t_{\mathsf{id}}) \leftarrow \mathcal{M}(t_{\mathsf{id}}) \cup \mathsf{M}_k$ and output $(\texttt{Issued}, \mathsf{sid}, t_{\mathsf{id}})$ to M_k via public-delayed output. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq \beta$: Set $\mathbb{K}(\mathsf{U}) \leftarrow (B + v, S, R + v, x + 1)$, $\mathsf{Rvk}(t_{\mathsf{id}}) \leftarrow (\mathsf{B}, \mathsf{U}, v)$, $\mathsf{Trc}(\mathsf{U}) \leftarrow \mathsf{Trc}(\mathsf{U}) \cup (t_{\mathsf{id}}, \mathsf{receiver})$, $P(\mathsf{pid}) \leftarrow \perp$, and $b \leftarrow 0$ if $B + v \leq B_{\mathsf{max}}$ and $R + v \leq R_{\mathsf{max}}$ hold. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq \alpha$: Output $(\texttt{Issued}, \mathsf{sid}, v)$ to U via private-delayed output and set $U(\mathsf{U}) \leftarrow \perp$ when delivered.

Payment. As in the case of *currency issuance*, the *payment* process requires both the sender U_s and the receiver U_r to be activated. Unlike issuance transactions, the functionality verifies that the sender U_s has a sufficient balance to fund the payment, ensuring that $B_s - v \geq 0$. Interactive payments are necessary because $\mathcal{F}_{\mathsf{CBDC}}$ enforces regulatory compliance (e.g., AML, CFT) by considering both parties, meaning each party must be aware of their counterparty in the transaction. Thus, it is crucial for the receiver U_r to actively engage in every payment[6]. A successful payment protocol increases the balance of the receiver U_r by the specified amount v while subtracting the same amount from the sender U_s. Additionally, each user's account information is updated to reflect different regulatory policies. As in the case of issuance, a unique transaction identifier t_{id} is assigned by the ideal-world adversary $\mathcal{A}$, and the transaction metadata are stored in the table $\mathsf{Rvk}(t_{\mathsf{id}})$. The identifier t_{id} is also recorded in $\mathsf{Trc}(\mathsf{U}_r)$ and $\mathsf{Trc}(\mathsf{U}_s)$, where U_r and U_s are the recipient and sender of the payment, respectively. Note that the adversary $\mathcal{A}$ is not aware of the transaction value or the identities of the sender and receiver (unless one of them is malicious), and t_{id} is selected independently of them.

Abort transaction. A user initiates an abort transaction request to update the state of their account. The update the user receives depends on the number of maintainers who actively participated in the user's most recent transaction (either in *currency issuance* or *payment*). If at most $2t$ maintainers have participated in the user's most recent transaction, the state of the account remains unchanged, except that the transaction counter x is incremented by one. Otherwise, the functionality notifies the transaction counterparties and maintainers that the transaction has been finalized.

Functionality $\mathcal{F}_{\mathsf{CBDC}}$, part II: *payment* and *abort transaction*

Payment.

- Upon receiving a message $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ from U_s: If $\mathbb{K}(\mathsf{U}_s) = \perp$ or $U(\mathsf{U}_s) \neq \perp$ ignore. Else, retrieve $\mathbb{K}(\mathsf{U}_s) = W_s$. If $S_s + v > S_{\mathsf{max}}$, or $B_s - v < 0$, or $v > V_{\mathsf{max}}$, or $v < 0$ holds ignore. Else, generate a new pid, set $U(\mathsf{U}_s) \leftarrow \mathsf{pid}$, and record the tuple $(\texttt{Tnx}, \mathsf{U}_s, \mathsf{U}_r, v, \mathsf{pid}, 1)$. If U_r is corrupted, output $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s, \mathsf{U}_r, v)$ to $\mathcal{A}$. Else, retrieve $T(\mathsf{U}_s)$: (a) If $T(\mathsf{U}_s) = 0$, output $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{A}$. (b) Else, output $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{U}_s, v)$ from U_r: If $\mathbb{K}(\mathsf{U}_r) = \perp$ or $U(\mathsf{U}_r) \neq \perp$ or the tuple $(\texttt{Tnx}, \mathsf{U}_s, \mathsf{U}_r, v, \mathsf{pid}, 1)$ is not recorded, ignore. Else, retrieve $\mathbb{K}(\mathsf{U}_r) = W_r$. If $B_r + v > B_{\mathsf{max}}$, or $R_r + v > R_{\mathsf{max}}$, ignore. Else, set $U(\mathsf{U}_r) \leftarrow \mathsf{pid}$ and retrieve $T(\mathsf{U}_r)$: (a) If $T(\mathsf{U}_r) = 0$, output $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{A}$. (b) Else, output $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_r)$ to $\mathcal{A}$.

[6] Refer to PARScoin [57] for an alternative system design and modeling approach, where the receiver does not need to be online at the time of payment. In PARScoin, payments are non-interactive between the sender and receiver. When the receiver comes online, she scans the ledger, identifies transactions associated with her, and submits a zero-knowledge proof to claim the funds. Furthermore, the receiver still needs to prove compliance with the system's regulations to successfully claim the funds.

- Upon receiving $(\texttt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ from $\mathcal{A}$: If already exits a $\mathsf{pid}' \neq \mathsf{pid}$ where $P(\mathsf{pid}') = t_{\mathsf{id}}$ or the tuple $(\texttt{Tnx}, \mathsf{U}_s, \mathsf{U}_r, v, \mathsf{pid}, 1)$ is not recorded, ignore. Else, if $P(\mathsf{pid}) = \bot$, set $P(\mathsf{pid}) \leftarrow t_{\mathsf{id}}$. Else, retrieve $P(\mathsf{pid}) = t'_{\mathsf{id}}$, ignore if $t'_{\mathsf{id}} \neq t_{\mathsf{id}}$. Set $\mathsf{Tid}(\mathsf{U}_s) \leftarrow (\mathsf{U}_s, \mathsf{U}_r, t_{\mathsf{id}}, v)$ and $\mathsf{Tid}(\mathsf{U}_r) \leftarrow (\mathsf{U}_s, \mathsf{U}_r, t_{\mathsf{id}}, v)$.
- Upon receiving $(\texttt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ from $\mathcal{A}$: Retrieve the tuple $(\texttt{Tnx}, \mathsf{U}_s, \mathsf{U}_r, v, \mathsf{pid}, b)$. Ignore if $b = 0$ or $P(\mathsf{pid}) = \bot$. Else, retrieve $P(\mathsf{pid}) = t_{\mathsf{id}}$. Set $\mathcal{M}(t_{\mathsf{id}}) \leftarrow \mathcal{M}(t_{\mathsf{id}}) \cup \mathsf{M}_k$, and output $(\texttt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ to M_k via public-delayed output. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq \beta$: Set $\mathbb{K}(\mathsf{U}_s) \leftarrow (B_s - v, S_s + v, R_s, x_s + 1)$, $\mathbb{K}(\mathsf{U}_r) \leftarrow (B_r + v, S_r, R_r + v, x_r + 1)$, $\mathsf{Rvk}(t_{\mathsf{id}}) \leftarrow (\mathsf{U}_s, \mathsf{U}_r, v)$, $\mathsf{Trc}(\mathsf{U}_s) \leftarrow \mathsf{Trc}(\mathsf{U}_s) \cup (t_{\mathsf{id}}, \mathsf{sender})$, $\mathsf{Trc}(\mathsf{U}_r) \leftarrow \mathsf{Trc}(\mathsf{U}_r) \cup (t_{\mathsf{id}}, \mathsf{receiver})$, $P(\mathsf{pid}) \leftarrow \bot$, and $b \leftarrow 0$ if $S_s + v \leq S_{\mathsf{max}}$, and $v \leq B_s$, and $B_r + v \leq B_{\mathsf{max}}$, and $R_r + v \leq R_{\mathsf{max}}$ hold. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq \alpha$: Output $(\texttt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$ to U_r via private-delayed output and set $U(\mathsf{U}_r) \leftarrow \bot$ when delivered. Output $(\texttt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ to U_s via private-delayed output and set $U(\mathsf{U}_s) \leftarrow \bot$ when delivered.

Abort transaction.

- Upon receiving a message $(\texttt{AbrTnx}, \mathsf{sid})$ from U[a]: If $\mathbb{K}(\mathsf{U}) = \bot$ or $\mathsf{Tid}(\mathsf{U}) = \bot$, ignore. Else, retrieve $\mathsf{Tid}(\mathsf{U}) = (\mathsf{U}_s, \mathsf{U}_r, t_{\mathsf{id}}, v)$. Send $(\texttt{AbrTnx}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{A}$.[b]
- Upon receiving $(\texttt{AbrTnx.Ok}, \mathsf{sid}, t_{\mathsf{id}})$ from $\mathcal{A}$: Set $\mathsf{Tid}(\mathsf{U}) \leftarrow \bot$. (a) If $|\mathcal{M}(t_{\mathsf{id}})| < 2t + 1$: Set $\mathbb{K}(\mathsf{U}) \leftarrow (B, S, R, x + 1)$, $\mathsf{Trc}(\mathsf{U}) \leftarrow \mathsf{Trc}(\mathsf{U}) \cup (t_{\mathsf{id}}, \mathsf{Aborted})$. Output $(\texttt{TnxAborted}, \mathsf{sid})$ to U via public-delayed output and set $U(\mathsf{U}) \leftarrow \bot$ when delivered. Output $(\texttt{TnxAborted}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathbb{M}$ via public-delayed output. (b) Else, given the retrieved tuple $(\mathsf{U}_s, \mathsf{U}_r, t_{\mathsf{id}}, v)$: Output $(\texttt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$ to U_r via private-delayed output and set $U(\mathsf{U}_r) \leftarrow \bot$ when delivered. Output $(\texttt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ to U_s via private-delayed output and set $U(\mathsf{U}_s) \leftarrow \bot$ when delivered. Output $(\texttt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathbb{M}$ via public-delayed output.

[a] Either $\mathsf{U} = \mathsf{U}_s$ or $\mathsf{U} = \mathsf{U}_r$ holds.

[b] Depending on whether $\mathsf{U} = \mathsf{U}_s$ or $\mathsf{U} = \mathsf{U}_r$, the functionality marks the associated tag $\mathsf{T}_s \in t_{\mathsf{id}}$ or $\mathsf{T}_r \in t_{\mathsf{id}}$ to make it distinguishable to $\mathcal{A}$.

Privacy revocation. Privacy revocation is initiated by maintainers who submit the transaction identifier of a fully anonymous payment they wish to revoke. If a sufficient number of maintainers (set to β) agree on the revocation of a specific transaction, the functionality recovers the metadata of the transaction and returns it to the maintainers and the adversary.

Tracing. As in the case of revocation, the maintainers must agree to trace a specific user. If the quorum is reached (requiring β maintainers), the set of transaction identifiers and roles corresponding to the agreed-upon user is returned to the maintainers and the adversary.

Functionality $\mathcal{F}_{\mathsf{CBDC}}$, part III: *privacy revocation* and *tracing*

Privacy revocation.

- Upon receiving a message $(\texttt{RvkAnm}, \mathsf{sid}, t^j_{\mathsf{id}})$ from maintainer M_j: If $\mathsf{Rvk}(t^j_{\mathsf{id}}) = \bot$, ignore. Else, record $(\texttt{RvkAnm}, \mathsf{sid}, t^j_{\mathsf{id}}, \mathsf{M}_j)$ and output $(\texttt{RvkAnm}, \mathsf{sid}, t^j_{\mathsf{id}}, \mathsf{M}_j)$ to $\mathcal{A}$. Once $|\{j | t^j_{\mathsf{id}} = t_{\mathsf{id}}\}| \geq \beta$, set $X \leftarrow t_{\mathsf{id}}$.
- Upon receiving $(\texttt{RvkAnm.Ok}, \mathsf{sid}, t_{\mathsf{id}})$ from $\mathcal{A}$: If X has not already been set to t_{id}, ignore. Else, retrieve $\mathsf{Rvk}(X) = (\mathsf{U}_s, \mathsf{U}_r, v)$. Output $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$[a] to $\mathbb{M}$ via public-delayed output.

Tracing.

- Upon receiving a message $(\texttt{Trace}, \mathsf{sid}, \mathsf{U}_j)$ from maintainer M_j: If $\mathbb{K}(\mathsf{U}_j) = \perp$ ignore. Else, record $(\texttt{Trace}, \mathsf{sid}, \mathsf{U}_j, \mathsf{M}_j)$ and output $(\texttt{Trace}, \mathsf{sid}, \mathsf{U}_j, \mathsf{M}_j)$ to $\mathcal{A}$. Once $|\{j | \mathsf{U}_j = \mathsf{U}\}| \geq \beta$, set $Y \leftarrow \mathsf{U}$.
- Upon receiving $(\texttt{Trace.Ok}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{A}$: If Y has not already been set to U, ignore. Else, retrieve $\mathbb{K}(\mathsf{U}) = (B, S, R, x)$. Retrieve $\mathsf{Trc}(Y) = \{t^{\tau}_{\mathsf{id}}, \mathsf{role}^{\tau}\}^{x}_{\tau=1}$. Set $T(\mathsf{U}) \leftarrow 1$. Output $(\texttt{Traced}, \mathsf{sid}, \mathsf{U}, \{t^{\tau}_{\mathsf{id}}, \mathsf{role}^{\tau}\}^{x}_{\tau=1})$ to $\mathbb{M}$ via public-delayed output.

[a] For a currency issuance transaction $\mathsf{U}_s = \mathsf{B}$.

4 Our construction: PEReDi

4.1 High-level technical overview

In our construction, we aim to achieve all the financial, regulatory, and security properties described informally in Section 3.1 and formally in Section 3.2. We assume that the total number of maintainers is $D = 5t + 1$ and that up to t of them can be corrupted by the adversary. This assumption arises because the optimistic path eliminates communication between maintainers entirely to enable fast settlement of payments. Consequently, $D = 5t+1$ is necessary to ensure security in the pessimistic execution path (see Section 4.4 for more details). Thus, we set the thresholds of the blind signature scheme and auditing as $\alpha = 4t+1$ and $\beta = t + 1$, respectively.

Every user in the system has an account acc for storing the current balance B and other user-specific values related to the system's financial and regulatory considerations. Users update their accounts when transacting. For each new *currency issuance* or *payment* transaction, the parties involved in the transaction engage in a cryptographic protocol with all maintainers $\mathbb{M}$. To this end, users encode the values of accounts into cryptographic one-time objects that fix a unique tag T. When updating an account, a user discloses the tag associated with the previous account snapshot acc (which has been signed by at least α maintainers). A user also discloses $\sigma^{\mathsf{Rnd}}_{\mathbb{M}}$, which is a re-randomization of the consolidated signature $\sigma_{\mathbb{M}}$ on their previous account snapshot. The disclosed tags are stored by maintainers to enforce that users utilize their most updated accounts (similar to Chaum's double-spending prevention for online cash [29]). To support tracing, the protocol computes tags pseudo-randomly so that they can be recomputed by the *auditing* protocol using a special-purpose multi-party computation (MPC) protocol, as we will see.

The user creates their updated account $\mathsf{acc}^{\mathsf{new}}$ and blinds it to obtain $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$. The blinded account is then submitted to $\mathbb{M}$ for signing, along with an efficient zero-knowledge proof that demonstrates, for example, that: (i) $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ is consistent with the previous account $\mathsf{acc}^{\mathsf{old}}$; (ii) $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ is consistent with the transaction value v issued by B to U in the *currency issuance* protocol or transferred from the sender U_s to the receiver U_r in the *payment* protocol (e.g., the same value is deducted from U_s's account and added to U_r's account); and (iii) the account state transition complies with the system's regulatory requirements. The parties engaged in a transaction must acquire at least α maintainers' blind signature shares $\sigma^{\mathsf{new},\mathfrak{B}}$ on their new blinded accounts. They then locally unblind these signature shares σ^{new} and aggregate them to create a single consolidated signature $\sigma^{\mathsf{new}}_{\mathbb{M}}$ on their new account.

Furthermore, every transaction results in a unique transaction identifier t_{id} that is output to maintainers $\mathbb{M}$ and stored in each maintainer's ledger $\mathcal{L}$. This identifier contains cryptographic information concerning the transaction. To ensure privacy, we prove that the transaction identifier t_{id} does not leak any privacy-sensitive information, thereby achieving full privacy. It is only retrievable and reconstructable by an audit committee using the information stored in $\mathcal{L}$ for the purpose of privacy revocation and tracing. In other words, an

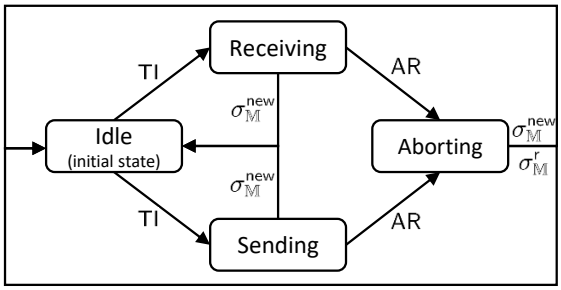

Fig. 1: User's state transition in PEReDi transactions. TI: transaction information. AR: abort request. $\sigma_{\mathbb{M}}^{\mathsf{new}}$: maintainers' signature on the user's new account. $\sigma_{\mathbb{M}}^{\mathsf{r}}$: maintainers' signature on the user's refreshed account.

audit can be conducted when the audit committee has been convinced that a specific transaction or user is sufficiently suspicious for anonymity to be revoked or for its counterparties to be traced, respectively. Note that tracing and revocation can be applied adaptively to reconstruct a set of counterparties across a sequence of payments.

In the following, we describe the user's state transition (in the *currency issuance* and *payment* protocols) in the PEReDi setting, as depicted in Figure 1. Upon receiving the environment's $\mathcal{Z}$ command of the form $(\texttt{AcceptIss}, \mathsf{sid}, v)$, $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{U}_s, v)$, or $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ to initiate a transaction:

- If in the Idle state, the user sends its transaction information TI (which includes U's new-blinded account $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$) to all maintainers $\mathbb{M}$. Upon sending TI, U's state changes to Receiving (if receiving from the central bank B or another user U_s) or to Sending (if sending to another user U_r).
- If in either the Receiving or Sending state (indicating that U's most recent transaction is still pending), U ignores $\mathcal{Z}$'s message.

When the state changes from Idle to Receiving or Sending, the transaction can either be successful or pending, as explained in the following cases:

- Successful (e.g., payment participants use their newly updated accounts, regulatory compliance is met, and maintainers have received valid transaction information for both payment participants). U receives at least α valid blind signature shares from maintainers on $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$. Upon generating the unblinded consolidated maintainers' signature on the new account $\sigma_{\mathbb{M}}^{\mathsf{new}}$, the state changes to Idle. Hence, U, who now has its new account signed, is ready to enter into the next transaction.
- Pending (e.g., the *sender-receiver* pair has not been generated on a sufficient number of maintainers' sides). U's state remains in Receiving or Sending until $\mathcal{Z}$ instructs U to send an abort request AR.

Upon $\mathcal{Z}$'s instruction of the form $(\texttt{AbrTnx}, \mathsf{sid})$ to send an abort request AR, which includes U's refreshed-blinded account $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$, U sends AR to $\mathbb{M}$. In this case, if U's state is not Sending or Receiving, it ignores $\mathcal{Z}$'s instruction. Doing so changes U's state from either Sending or Receiving to Aborting. The following two scenarios apply when U is in the Aborting state:

- If sufficient number of maintainers have saved a *sender-receiver* TI pair in their ledgers, they ignore $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ and send their signatures for $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ to U. Upon generating the unblinded consolidated maintainers' signature on the new account $\sigma_{\mathbb{M}}^{\mathsf{new}}$, the state changes to Idle.
- Otherwise, maintainers sign $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$, record the pending transaction as aborted, and ignore $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ included in TI. Upon generating the unblinded consolidated maintainers' signature on the refreshed account $\sigma_{\mathbb{M}}^{\mathsf{r}}$, the state changes to Idle.

A pictorial representation of all the sub-protocols of our construction can be found in Figure 2-23. Note that, for simplicity, the figures do not include the messages exchanged between the environment $\mathcal{Z}$ and the parties. For the same reason, AR messages are also omitted.

4.2 Details of the construction

In this section, we describe our CBDC construction Π_{PEReDi}. We will prove that Π_{PEReDi} securely realizes $\mathcal{F}_{\mathsf{CBDC}}$. Our construction uses several cryptographic components, including:

- Threshold ElGamal encryption (Definition 11).
- Shamir secret sharing (Appendix B.2).
- Pedersen commitment (Definition 19).
- Threshold blind signature TBS (Definition 4), which is based on the Pointcheval-Sanders signature [53] (Definition 3). Our scheme uses the Coconut threshold blind signature scheme [59,54] with small modifications. We reduce the unforgeability of the (modified) Coconut scheme to its underlying Pointcheval-Sanders signature component. Throughout this section, when we use TBS, we employ its algorithms as described in Definition 4. However, whenever possible, we merge its zero-knowledge proofs with those of the rest of the protocol to improve performance.

Π_{PEReDi} employs the following ideal functionalities:

- Key registration functionality $\mathcal{F}_{\mathsf{KR}}$ (Appendix C.1).
- Communication channels functionality $\mathcal{F}_{\mathsf{Ch}}$ (Appendix C.2), parameterized by different labels, e.g., "sa" for a sender-anonymous channel $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$. We assume that transacting parties communicate through variants of $\mathcal{F}_{\mathsf{Ch}}$ as specified. We note that some degree of sender anonymity is necessary for privacy; otherwise, network "leakage" will trivially reveal the counterparties of a transaction, regardless of the strength of cryptographic protections at the transactional level. We also note that in a real-world deployment, such network leakage may be considered tolerable. Our analysis applies directly to such a setting as well, with the unavoidable concession that the adversary may compromise privacy through traffic analysis.
- Asynchronous Byzantine agreement functionality $\mathcal{F}_{\mathsf{aBA}}$ (Appendix C.3).
- Broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ (Appendix C.4).
- Random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ (Appendix C.5).
- Non-interactive zero-knowledge (NIZK) functionality $\mathcal{F}_{\mathsf{NIZK}}$ (Appendix C.6).
- Signature of knowledge (SoK) functionality $\mathcal{F}_{\mathsf{SoK}}$ (Appendix C.7).

Each maintainer M has its own ledger $\mathcal{L}$ for storing registration and transaction information. In the *currency issuance* and *payment* protocols described below, the sender and receiver separately send their transaction information TI to all maintainers $\mathbb{M}$. However, an alternative plausible communication pattern could involve the sender first sending its transaction information TI to the receiver, who then forwards both the sender's TI and their own TI to $\mathbb{M}$.

The public key of the threshold encryption scheme, the ciphertexts, and the tracing tags all belong to $\mathbb{G}$ (see Definition 5), the first source group in the bilinear map.

4.2.1 Initialization The key generation algorithm takes the security parameter as input and generates the secret key sk and public key pk for the caller of the algorithm as outputs. Participants in the network independently invoke the key generation algorithm for each underlying cryptographic scheme to generate their keys (see Appendix B for key generation algorithms). The public keys of all parties are stored in a public-key directory and are assumed to be accessible on demand by calling $\mathcal{F}_{\mathsf{KR}}$ with input $(\texttt{RetrieveKey}, \mathsf{sid}, \mathsf{P})$ for party P.

4.2.2 User registration Upon receiving $(\texttt{GenAcc}, \mathsf{sid})$ from $\mathcal{Z}$, U, who is initially in the Idle state, initiates the *user registration* protocol (see Figure 2) to get their initial account signed by maintainers $\mathbb{M}$.

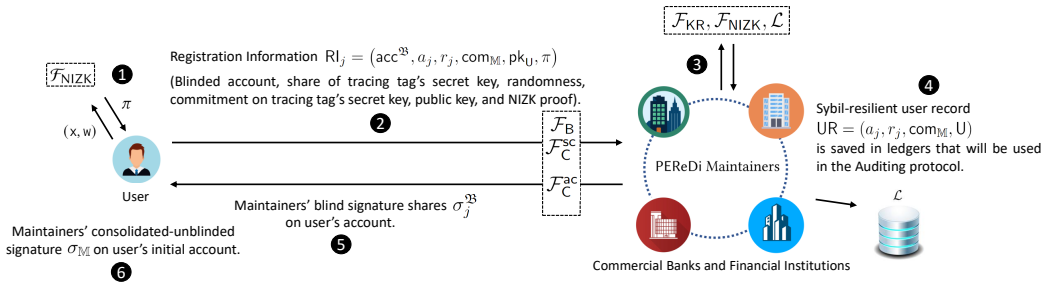

Fig. 2: *User registration* protocol

Maintainers $\mathbb{M}$ enroll a user U in the CBDC system by creating a signature on the user's initial account. Afterwards, U uses this signature to create transactions. For registration, U, with a pair of public-secret keys $(\mathsf{pk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}})$ and a secret key a (used in tag generation), engages in a threshold blind signature (TBS) protocol with $\mathbb{M}$, where U proves the well-formedness of their initial blinded account to $\mathbb{M}$.

The output of this protocol is a signed account $\sigma_{\mathbb{M}}$ for U (needed for their first transaction) and the user record UR, which is saved in the ledger $\mathcal{L}$ of each maintainer M (required for additional investigation during the *auditing* protocol, as we will see). Every user's account consists of a tuple of field elements

$$\mathsf{acc} = (B, S, R, \mathsf{sk}_{\mathsf{U}}, a^x, a)$$

where B represents the balance, S is the sum of all sent values, R is the sum of all received values, sk_{U} and a are secret keys, and x is the transaction counter. During registration, U sets B, S, R, and x to 0.

U generates its registration information RI, as described in Figure 3.

▷ Upon receiving $(\texttt{GenAcc}, \mathsf{sid})$ from $\mathcal{Z}$, act as follows.
▷ Set $\mathsf{acc} \leftarrow (0, 0, 0, \mathsf{sk}_{\mathsf{U}}, 1, a)$.
▷ Given acc, call $\mathsf{PrepareBlindSign}$ to generate the blinded account $\mathsf{acc}^{\mathfrak{B}}$.[a]
▷ Call $\{a_j\}_{j=1}^{D} \xleftarrow{\$} \mathsf{SSH.Share}^{D,\beta}(a)$ to secret share a and compute:

$$\tilde{\mathsf{com}}_j = g^{a_j} \cdot h^{r_j}, \quad \forall j \in \{1, \ldots, D\}, \quad \text{where } r_j \xleftarrow{\$} \mathbb{Z}_p^*, \quad \text{set } \mathsf{com}_{\mathbb{M}} \leftarrow \{\tilde{\mathsf{com}}_j\}_{j=1}^{D}.$$

▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$, and receive back $(\texttt{Proof}, \mathsf{sid}, \pi)$.
Denote the randomness used to create the blinded account $\mathsf{acc}^{\mathfrak{B}}$ and the commitment $\mathsf{com}_{\mathbb{M}}$ by $\mathsf{r}_{\mathsf{acc.com}}$. The following is the NIZK statement $\mathtt{x}$ and witness $\mathtt{w}$:

$$\mathtt{x} = (\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}), \qquad \mathtt{w} = (\mathsf{acc}, \{a_j\}_{j=1}^{D}, \mathsf{r}_{\mathsf{acc.com}}).$$

The definition of the relation $\mathcal{R}(\mathtt{x}, \mathtt{w})$ of NIZK is informally as follows (see below for the formal definition):

- The secret key sk_{U} in the blinded account $\mathsf{acc}^{\mathfrak{B}}$ is the secret key associated with the public key pk_{U}.

- The secret key a in $\mathsf{acc}^{\mathfrak{B}}$ is the same as the secret key that can be reconstructed from the shares $\{a_j\}_{j=1}^{D}$ committed in $\mathsf{com}_{\mathbb{M}}$.
- $\mathsf{acc}^{\mathfrak{B}}$ is generated such that $B = S = R = x = 0$ holds.
- The user U knows the randomness $\mathsf{r}_{\mathrm{acc.com}}$.

▷ Call $\mathcal{F}_{\mathsf{BC}}$ with $(\texttt{Broadcast}, \mathsf{sid}, \mathsf{com}_{\mathbb{M}})$.
▷ Set registration information:

$$\mathsf{RI}_j \leftarrow (\mathsf{acc}^{\mathfrak{B}}, a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}, \pi), \quad \forall j \in \{1, \ldots, D\}.$$

▷ For $k = 1$ to $D - 1$, perform the following steps:
- Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$ with the message $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{RI}_k)$.
- Wait for a response: $(\texttt{Continue}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$.

▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$ with the message $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_D, \mathsf{RI}_D)$.

[a] Given acc, it calls the $\mathsf{PrepareBlindSign}$ algorithm of the threshold blind signature TBS scheme to obtain a blinded account $\mathsf{acc}^{\mathfrak{B}}$. Here, as explained in the beginning of this section, in contrast to the original $\mathsf{PrepareBlindSign}$ algorithm of Coconut [59], the algorithm does not create a proof. All necessary ZK proofs are included in the user registration ZK relation (see below).

Fig. 3: User registration – U 1^{st} phase

Formal definition of user registration NIZK relation:

- $\mathtt{w} = \Big((0, 0, 0, \mathsf{sk}_{\mathsf{U}}, 1, a), \{a_j\}_{j=1}^{D}, o, \{o_\tau\}_{\tau=1}^{6}, \{r_j\}_{j=1}^{D}, \{\mathsf{coe}_j\}_{j=1}^{\beta-1}\Big)$ [7]
- $\mathtt{x} = \big(\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}\big)$
- The relation $\mathcal{R}(\mathtt{x}, \mathtt{w})$ is defined as:

$$\begin{aligned}\mathcal{R}(\mathtt{x}, \mathtt{w}) = \Big\{ & \mathsf{com} = g^{o} \cdot h_4^{\mathsf{sk}_{\mathsf{U}}} \cdot h_5 \cdot h_6^{a} \wedge \mathsf{com}_1 = g^{o_1} \wedge \mathsf{com}_2 = g^{o_2} \wedge \mathsf{com}_3 = g^{o_3} \\ & \wedge \mathsf{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}_{\mathsf{U}}} \wedge \mathsf{com}_5 = g^{o_5} \cdot h \wedge \mathsf{com}_6 = g^{o_6} \cdot h^{a} \\ & \wedge \Big\{ \tilde{\mathsf{com}}_j = g^{a_j} h^{r_j} \Big\}_{j=1}^{D} \wedge \mathsf{pk}_{\mathsf{U}} = g^{\mathsf{sk}_{\mathsf{U}}} \\ & \wedge \Big\{ \tilde{\mathsf{com}}_j = g^{a + \sum_{i=1}^{\beta-1} \mathsf{coe}_i j^i} \cdot h^{r_j} = g^{a} \cdot \prod_{i=1}^{\beta-1} g^{\mathsf{coe}_i j^i} \cdot h^{r_j} \Big\}_{j=1}^{D} \Big\}.\end{aligned}$$

See Appendix A.3 for the implementation details of the relation using Sigma protocols.

Each maintainer (M_j) acts as described in Figure 4.

▷ Upon receiving $(\texttt{Broadcasted}, \mathsf{sid}, \mathsf{U}, \mathsf{com}_{\mathbb{M}})$ from $\mathcal{F}_{\mathsf{BC}}$, and $(\texttt{Received}, \mathsf{sid}, \mathsf{U}, \mathsf{RI}_j)$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$, act as follows.
▷ Parse $\mathsf{RI}_j = (\mathsf{acc}^{\mathfrak{B}}, a_j^*, r_j^*, \mathsf{com}_{\mathbb{M}}^*, \mathsf{pk}_{\mathsf{U}}, \pi)$.
▷ Ignore if $\mathsf{com}_{\mathbb{M}} \neq \mathsf{com}_{\mathbb{M}}^*$. Otherwise, parse $\mathsf{com}_{\mathbb{M}} = \{\tilde{\mathsf{com}}_j\}_{j=1}^{D}$.
▷ Ignore if at least one of the following conditions holds:
- There already exists a user record UR' in $\mathcal{L}_j$ where $\mathsf{U}' = \mathsf{U}$.
- Upon calling $\mathcal{F}_{\mathsf{KR}}$ with $(\texttt{RetrieveKey}, \mathsf{sid}, \mathsf{U})$, $(\texttt{KeyRetrieved}, \mathsf{sid}, \mathsf{U}, \mathsf{pk}')$ is received such that $\mathsf{pk}_{\mathsf{U}} \neq \mathsf{pk}'$.
- Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$, $(\texttt{Verification}, \mathsf{sid}, 0)$ is received.

[7] See Definition 4 for more information.

- Given $(a_j^*, r_j^*) \in \mathsf{RI}_j$, compute $g^{a_j^*} \cdot h^{r_j^*}$ where it is not equal to $\tilde{\mathsf{com}}_j$ for $\tilde{\mathsf{com}}_j \in \mathsf{com}_{\mathbb{M}}$.
- Know your customer (KYC) guidelines for U are not verified.

▷ Otherwise, save the user record $\mathsf{UR} = (a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{U})$ in $\mathcal{L}_j$.
▷ Given $\mathsf{acc}^{\mathfrak{B}}$, call $\mathsf{BlindSign}$ to obtain $\sigma_j^{\mathfrak{B}}$.[a]
▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, \sigma_j^{\mathfrak{B}})$.
▷ Output $(\mathtt{AccGened}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{Z}$.

[a] Each maintainer signs associated information of $\mathsf{acc}^{\mathfrak{B}}$ using the $\mathsf{BlindSign}$ algorithm of the TBS scheme to obtain a blind signature share $\sigma_j^{\mathfrak{B}}$.

Fig. 4: User registration – M_j

The user U acts as described in Figure 5.

▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathfrak{B}})$ for different j from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$.
▷ Given $\sigma_j^{\mathfrak{B}}$ in the messages received from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, call $\mathsf{Unblind}$ to obtain σ_j.[a]
▷ Given the unblinded signatures $\{\sigma_j\}_{j=1}^{\alpha}$, call $\mathsf{TBS.Agg}$ to obtain $\sigma_{\mathbb{M}}$.[b]
▷ Output $(\mathtt{AccGened}, \mathsf{sid})$ to $\mathcal{Z}$.

[a] Unblinds at least α different signature shares $\{\sigma_j\}_{j=1}^{\alpha}$ using the $\mathsf{Unblind}$ algorithm of the TBS scheme.
[b] Aggregates unblinded signature shares using the $\mathsf{TBS.Agg}$ algorithm of the TBS scheme to form one consolidated signature $\sigma_{\mathbb{M}}$.

Fig. 5: User registration – U 2nd phase

4.2.3 Currency issuance Upon receiving $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ from $\mathcal{Z}$, B initiates the *currency issuance* protocol as shown in Figure 6. To issue digital currency worth v for U, B first sends v to U by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sra}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, v)$, so that U receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, v)$.

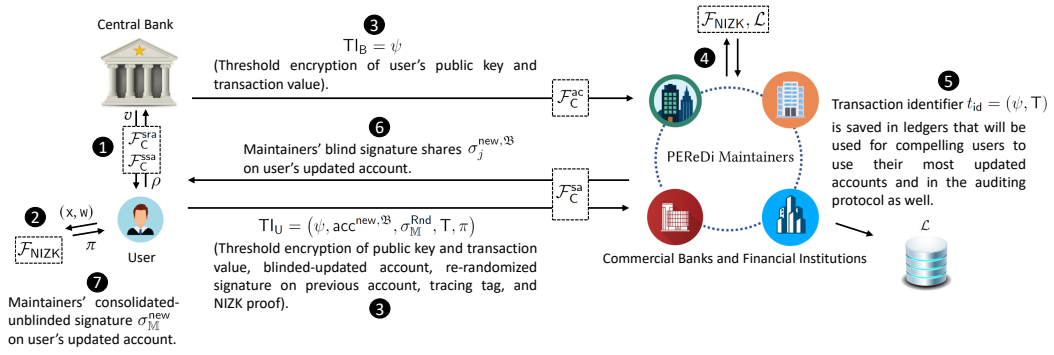

Fig. 6: *Currency issuance* protocol

Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, v)$ from $\mathcal{Z}$, if U is in the Idle state, it sends the fresh randomness ρ of ψ to B by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ssa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \rho)$.[8] ψ is a threshold ElGamal encryption of U's public key pk_{U} and g^v.

[8] Otherwise, if U is in the $\mathsf{Sending}$ or $\mathsf{Receiving}$ state, it ignores the message.

U must prove that it has a valid signature $\sigma_{\mathbb{M}}$ on its previous account $\mathsf{acc}^{\mathsf{old}}$ and request a new signature on its new account $\mathsf{acc}^{\mathsf{new}}$. $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ is computed for U's new account $\mathsf{acc}^{\mathsf{new}}$ using the $\mathsf{PrepareBlindSign}$ algorithm, and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is computed for U's previous account $\mathsf{acc}^{\mathsf{old}}$ (for which it has the consolidated signature $\sigma_{\mathbb{M}}$) using the $\mathsf{ProveSig}$ algorithm of the TBS scheme.
The tag is computed as $\mathsf{T} = g^{a^{x+1}}$, where x is an incrementing value per transaction. As we will see, the same value is used for tracing the user when necessary. U generates its transaction information TI_{U} and sends it to $\mathbb{M}$.

The algorithm executed by U is provided in Figure 7.

▷ Compute threshold ElGamal encryption as follows, setting the public key pk_{U} and g^v as plaintexts:

$$\psi = (\psi_1, \psi_2, \psi_3) \leftarrow (g^{\rho}, \mathsf{pk}_{1,\mathbb{M}}^{\rho} \cdot \mathsf{pk}_{\mathsf{U}}, \mathsf{pk}_{2,\mathbb{M}}^{\rho} \cdot g^{v})$$

▷ Given $\sigma_{\mathbb{M}}$, call $\mathsf{ProveSig}$ to obtain $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$.
▷ Given $\mathsf{acc}^{\mathsf{old}} = (B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}_{\mathsf{U}}, a^x, a)$ and v, set:

$$\mathsf{acc}^{\mathsf{new}} = (B^{\mathsf{new}}, S^{\mathsf{new}}, R^{\mathsf{new}}, \mathsf{sk}_{\mathsf{U}}, a^{x+1}, a) \leftarrow (B^{\mathsf{old}} + v, S^{\mathsf{old}}, R^{\mathsf{old}} + v, \mathsf{sk}_{\mathsf{U}}, a^x \cdot a, a).$$

▷ Given $\mathsf{acc}^{\mathsf{new}}$, call $\mathsf{PrepareBlindSign}$ to generate the blinded account $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$.
▷ Compute $\mathsf{T} = g^{a^{x+1}}$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$, and receive back $(\mathtt{Proof}, \mathsf{sid}, \pi)$.
Denote the randomness used to create $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}$, and threshold encryption ψ by $\mathsf{r}_{\mathsf{reg}}$. The following is the NIZK statement $\mathtt{x}$ and witness $\mathtt{w}$:

$$\mathtt{x} = (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}), \qquad \mathtt{w} = (\mathsf{acc}^{\mathsf{old}}, \mathsf{r}_{\mathsf{reg}}, v)$$

The definition of the relation $\mathcal{R}(\mathtt{x}, \mathtt{w})$ of NIZK is informally as follows (see below for formal definition):

- The secret key sk_{U} used in $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ is the secret key associated with the public key pk_{U} in the threshold encryption ψ generated under the public key of maintainers $\mathsf{pk}_{1,\mathbb{M}}$.
- T is well-formed; the exponent of g is the fifth element in $\mathsf{acc}^{\mathsf{new}}$.
- $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is a re-randomization of $\sigma_{\mathbb{M}}$, which is a signature generated by aggregating α different valid signature shares of maintainers on $\mathsf{acc}^{\mathsf{old}}$.
- $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ is generated considering $\mathsf{acc}^{\mathsf{old}}$ and v in ψ. Hence, the following hold for $\mathsf{acc}^{\mathsf{new}}$: $B^{\mathsf{new}} = B^{\mathsf{old}} + v$, $S^{\mathsf{new}} = S^{\mathsf{old}}$, $R^{\mathsf{new}} = R^{\mathsf{old}} + v$, $\mathsf{sk}_{\mathsf{U}}^{\mathsf{new}} = \mathsf{sk}_{\mathsf{U}}$, $a^{x+1} = a^x \cdot a$, $a^{\mathsf{new}} = a$. Additionally, the following constraints hold: $B^{\mathsf{new}} \leq B_{\mathsf{max}}$, $R^{\mathsf{new}} \leq R_{\mathsf{max}}$.[a]
- U knows the randomness $\mathsf{r}_{\mathsf{reg}}$.

▷ Set transaction information:

$$\mathsf{TI}_{\mathsf{U}} \leftarrow (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi).$$

▷ For $k = 1$ to $D - 1$, perform the following steps:

- Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_{\mathsf{U}})$.
- Wait for a response: $(\mathtt{Continue}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$.

▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_D, \mathsf{TI}_{\mathsf{U}})$.

[a] Different from the *payment* protocol, in which the transferred value is upper bounded, in this protocol, there is no upper bound on the transaction value v issued by B. However, as addressed before, it is straightforward to add such a constraint if desired.

Fig. 7: Currency issuance – U 1st phase

Formal definition of currency issuance NIZK relation:

- $\mathtt{w} = \Big((B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}, \varphi, a), \rho, o, \{o_\tau\}_{\tau=1}^{6}, r, r_1, r_2, v\Big)$ where $\varphi = a^x$.
- $\mathtt{x} = \big(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}\big)$
- The relation $\mathcal{R}(\mathtt{x}, \mathtt{w})$ is defined as:

$$\mathcal{R}(\mathtt{x},\mathtt{w}) = \Big\{ \psi_1 = g^\rho \wedge \psi_2 = \mathsf{pk}_{1,\mathbb{M}}^\rho \cdot g^{\mathsf{sk}} \wedge \psi_3 = \mathsf{pk}_{2,\mathbb{M}}^\rho \cdot g^v$$

$$\wedge\, \mathsf{com} = g^o \cdot h_1^{B^{\mathsf{old}}+v} \cdot h_2^{S^{\mathsf{old}}} \cdot h_3^{R^{\mathsf{old}}+v} \cdot h_4^{\mathsf{sk}} \cdot h_5^{\varphi^{\mathsf{old}}\cdot a} \cdot h_6^a$$
$$\wedge\, \mathsf{com}_1 = g^{o_1} \cdot h^{B^{\mathsf{old}}+v} \wedge \mathsf{com}_2 = g^{o_2} \cdot h^{S^{\mathsf{old}}} \wedge \mathsf{com}_3 = g^{o_3} \cdot h^{R^{\mathsf{old}}+v}$$
$$\wedge\, \mathsf{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}} \wedge \mathsf{com}_5 = g^{o_5} \cdot h^{\varphi^{\mathsf{old}}\cdot a} \wedge \mathsf{com}_6 = g^{o_6} \cdot h^a$$

$$\wedge\, \kappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{B^{\mathsf{old}}} \cdot \tilde{\beta}_2^{S^{\mathsf{old}}} \cdot \tilde{\beta}_3^{R^{\mathsf{old}}} \cdot \tilde{\beta}_4^{\mathsf{sk}} \cdot \tilde{\beta}_5^{\varphi^{\mathsf{old}}} \cdot \tilde{\beta}_6^a \cdot \tilde{g}^r$$
$$\wedge\, \mathsf{T} = g^{\varphi^{\mathsf{old}}\cdot a} \wedge N = g^{r_1} \cdot h^{\varphi^{\mathsf{old}}} \wedge \mathsf{com}_5 = N^a \cdot g^{r_2}$$
$$\wedge\, B^{\mathsf{new}} = B^{\mathsf{old}} + v \le B_{\mathsf{max}} \wedge R^{\mathsf{new}} = R^{\mathsf{old}} + v \le R_{\mathsf{max}} \Big\}.$$

See Appendix A.1 for the implementation details of the relation using Sigma protocols and bulletproofs (used only for range proofs).

B acts as described in Figure 8.

▷ Upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}, \rho)$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ssa}}$, act as follows.
▷ Set the transaction information $\mathsf{TI}_\mathsf{B} = \psi = (\psi_1, \psi_2, \psi_3) \leftarrow (g^\rho, \mathsf{pk}_{1,\mathbb{M}}^\rho \cdot \mathsf{pk}_\mathsf{U}, \mathsf{pk}_{2,\mathbb{M}}^\rho \cdot g^v)$.
▷ For $k = 1$ to $D-1$, perform the following steps:
- Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_\mathsf{B})$.
- Wait for a response: $(\mathtt{Continue}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$.

▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_D, \mathsf{TI}_\mathsf{B})$.

Fig. 8: Currency issuance – B

Each maintainer (M_j) acts as described in Figure 9.

▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_\mathsf{U}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ and parse $\mathsf{TI}_\mathsf{U} = (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$.
▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, \mathsf{TI}_\mathsf{B})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ and parse $\mathsf{TI}_\mathsf{B} = \psi$.[a]
▷ Ignore if at least one of the following conditions holds:
- There already exists a transaction identifier t'_{id} (either for an issuance transaction or an aborted transaction) in the ledger $\mathcal{L}_j$, where $\mathsf{T}' \in t'_{\mathsf{id}}$ and $\mathsf{T}' = \mathsf{T}$, or $\psi' \in t'_{\mathsf{id}}$ and $\psi' = \psi$. The latter condition applies only if t'_{id} corresponds to an issuance transaction.
- There already exists a transaction identifier t'_{id} (for a payment transaction) in $\mathcal{L}_j$, where $\mathsf{T}'_s \in t'_{\mathsf{id}}$ and $\mathsf{T}'_s = \mathsf{T}$, or $\mathsf{T}'_r \in t'_{\mathsf{id}}$ and $\mathsf{T}'_r = \mathsf{T}$.
- Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$, $(\mathtt{Verification}, \mathsf{sid}, 0)$ is received.
- Given $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$, upon calling $\mathsf{VerifySig}$, 0 is received.

▷ Otherwise, record a *sender-receiver* pair $((\mathsf{TI}_\mathsf{B}, \mathsf{TI}_\mathsf{U}), \mathsf{mid})$ in $\mathcal{L}_j$.
▷ Set the transaction identifier $t_{\mathsf{id}} \leftarrow (\psi, \mathsf{T})$ and record it in $\mathcal{L}_j$.
▷ Given $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$, call $\mathsf{BlindSign}$ to obtain $\sigma_j^{\mathsf{new},\mathfrak{B}}$.

▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}, \sigma_j^{\mathsf{new},\mathfrak{B}})$.
▷ Output $(\mathtt{Issued}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{Z}$.

[a] Note that it does not matter which transaction information TI_{U} or TI_{B} is received by M_j first.

Fig. 9: Currency issuance – M_j

The user U acts as described in Figure 10.

▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{new},\mathfrak{B}})$ for different j from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$.
▷ Given $\sigma_j^{\mathsf{new},\mathfrak{B}}$ received from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ for different j, call $\mathsf{Unblind}$ to obtain σ_j^{new}.
▷ Given $\{\sigma_j^{\mathsf{new}}\}_{j=1}^{\alpha}$ computed in the previous step, call $\mathsf{TBS.Agg}$ to obtain $\sigma_{\mathbb{M}}^{\mathsf{new}}$.
▷ Output $(\mathtt{Issued}, \mathsf{sid}, v)$ to $\mathcal{Z}$.

Fig. 10: Currency issuance – U 2^{nd} phase

4.2.4 Payment To make a payment, upon receiving $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ from $\mathcal{Z}$, if U_s is in the Idle state, it initiates the *payment* protocol as shown in Figure 11 by sending fresh randomness ρ_s of ψ_s and the transaction value v to the receiver U_r by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}_r, (\rho_s, v))$.[9] ψ_s is a threshold ElGamal encryption of U_s's public key pk_s and g^v.

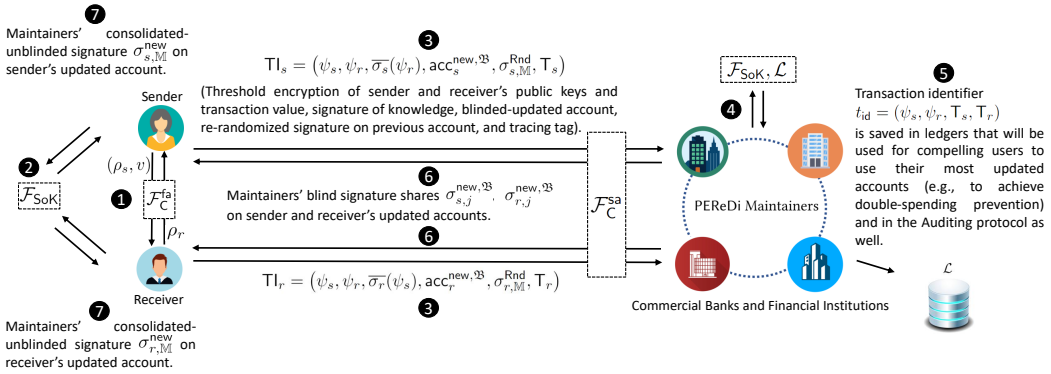

Fig. 11: *Payment* protocol

On receiving $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{U}_s, v)$ from $\mathcal{Z}$, if U_r is in the Idle state, it sends back fresh randomness ρ_r used in ψ_r to U_s by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}_s, \rho_r)$.[10] ψ_r is a threshold ElGamal encryption of U_r's public key pk_r. U_s and U_r generate their transaction information, denoted by TI_s and TI_r, respectively, and send it to maintainers.
The sender U_s executes the algorithm provided in Figure 12.

▷ Compute threshold ElGamal encryptions:

$$\psi_s = (\psi_{s,1}, \psi_{s,2}, \psi_{s,3}) = (g^{\rho_s}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_s} \cdot \mathsf{pk}_s, \mathsf{pk}_{2,\mathbb{M}}^{\rho_s} \cdot g^v), \quad \psi_r = (\psi_{r,1}, \psi_{r,2}) = (g^{\rho_r}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_r} \cdot \mathsf{pk}_r).$$

[9] Otherwise, if U_s is in the $\mathsf{Sending}$ or $\mathsf{Receiving}$ state, or if $v < 0$, it ignores the message.
[10] Otherwise, if U_r is in the $\mathsf{Sending}$ or $\mathsf{Receiving}$ state, or if $v < 0$, it ignores the message.

▷ Given $\mathsf{acc}_s^{\mathsf{old}} = (B_s^{\mathsf{old}}, S_s^{\mathsf{old}}, R_s^{\mathsf{old}}, \mathsf{sk}_s, a_s^{x_s}, a_s)$ and v, set:

$$\mathsf{acc}_s^{\mathsf{new}} = (B_s^{\mathsf{new}}, S_s^{\mathsf{new}}, R_s^{\mathsf{new}}, \mathsf{sk}_s, a_s^{x_s+1}, a_s) \leftarrow (B_s^{\mathsf{old}} - v, S_s^{\mathsf{old}} + v, R_s^{\mathsf{old}}, \mathsf{sk}_s, a_s^{x_s} \cdot a_s, a_s).$$

▷ Compute $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$, and T_s similar to the *currency issuance* protocol.
▷ Call $\mathcal{F}_{\mathsf{SoK}}$ on input $(\mathtt{Sign}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \mathtt{w}_s)$ and receive $(\mathtt{Signature}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$, where $\overline{\sigma_s}(\psi_r)$ is U_s's signature of knowledge (SoK) on ψ_r, binding the message ψ_r to the proof so that it proves knowledge of $\mathtt{w}_s$ satisfying the relation $\mathcal{R}(\mathtt{x}_s, \mathtt{w}_s)$.
Denote the set of all random values associated with $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$, and ψ_s by r_s. The following is the SoK statement $\mathtt{x}_s$ and witness $\mathtt{w}_s$:

$$\mathtt{x}_s = (\psi_s, \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s), \qquad \mathtt{w}_s = (\mathsf{acc}_s^{\mathsf{old}}, \mathsf{r}_s, v),$$

and the message of SoK, $\overline{\sigma_s}(\psi_r)$, is ψ_r.
The definition of the relation $\mathcal{R}(\mathtt{x}_s, \mathtt{w}_s)$ of SoK is informally as follows (see below for the formal definition):

- The secret key sk_s used in $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$ is the secret key associated with the public key pk_s in the threshold encryption ψ_s generated under the public key of maintainers $\mathsf{pk}_{1,\mathbb{M}}$.
- T_s is well-formed; the exponent of g is the fifth element in $\mathsf{acc}_s^{\mathsf{new}}$.
- $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$ is a re-randomization of $\sigma_{s,\mathbb{M}}$, which is a signature generated by aggregating α different valid signature shares of maintainers on $\mathsf{acc}_s^{\mathsf{old}}$.
- $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$ is generated considering $\mathsf{acc}_s^{\mathsf{old}}$ and v in ψ_s. Hence, the following hold for $\mathsf{acc}_s^{\mathsf{new}}$: $B_s^{\mathsf{new}} = B_s^{\mathsf{old}} - v$, $S_s^{\mathsf{new}} = S_s^{\mathsf{old}} + v$, $R_s^{\mathsf{new}} = R_s^{\mathsf{old}}$, $\mathsf{sk}_s^{\mathsf{new}} = \mathsf{sk}_s$, $a_s^{x_s+1} = a_s^x \cdot a_s$, $a_s^{\mathsf{new}} = a_s$. Additionally, the following constraints hold: $0 \le B_s^{\mathsf{new}}$, $S_s^{\mathsf{new}} \le S_{\mathsf{max}}$, $0 \le v \le V_{\mathsf{max}}$.
- U_s knows the randomness r_s.

▷ Set the sender transaction information:

$$\mathsf{TI}_s \leftarrow (\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s).$$

Fig. 12: Payment – U_s 1^{st} phase

Formal definition of payment (sender side) NIZK relation:

- $\mathtt{w}_s = \Big((B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}, \varphi^{\mathsf{old}}, a), \rho, o, \{o_\tau\}_{\tau=1}^6, r, r_1, r_2, v\Big)$ where $\varphi^{\mathsf{old}} = a^x$.
- $\mathtt{x}_s = \big(\psi_s, \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s\big)$
- The relation $\mathcal{R}(\mathtt{x}_s, \mathtt{w}_s)$ is defined as:

$$\begin{aligned}
\mathcal{R}(\mathtt{x}_s, \mathtt{w}_s) = \Big\{ & \psi_{s,1} = g^\rho \wedge \psi_{s,2} = \mathsf{pk}_{1,\mathbb{M}}^\rho \cdot g^{\mathsf{sk}} \wedge \psi_{s,3} = \mathsf{pk}_{2,\mathbb{M}}^\rho \cdot g^v \\
& \wedge \mathsf{com} = g^o \cdot h_1^{B^{\mathsf{old}}-v} \cdot h_2^{S^{\mathsf{old}}+v} \cdot h_3^{R^{\mathsf{old}}} \cdot h_4^{\mathsf{sk}} \cdot h_5^{\varphi^{\mathsf{old}} \cdot a} \cdot h_6^a \\
& \wedge \mathsf{com}_1 = g^{o_1} \cdot h^{B^{\mathsf{old}}-v} \wedge \mathsf{com}_2 = g^{o_2} \cdot h^{S^{\mathsf{old}}+v} \wedge \mathsf{com}_3 = g^{o_3} \cdot h^{R^{\mathsf{old}}} \\
& \wedge \mathsf{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}} \wedge \mathsf{com}_5 = g^{o_5} \cdot h^{\varphi^{\mathsf{old}} \cdot a} \wedge \mathsf{com}_6 = g^{o_6} \cdot h^a \\
& \wedge \kappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{B^{\mathsf{old}}} \cdot \tilde{\beta}_2^{S^{\mathsf{old}}} \cdot \tilde{\beta}_3^{R^{\mathsf{old}}} \cdot \tilde{\beta}_4^{\mathsf{sk}} \cdot \tilde{\beta}_5^{\varphi^{\mathsf{old}}} \cdot \tilde{\beta}_6^a \cdot \tilde{g}^r \\
& \wedge \mathsf{T} = g^{\varphi^{\mathsf{old}} \cdot a} \wedge N = g^{r_1} \cdot h^{\varphi^{\mathsf{old}}} \wedge \mathsf{com}_5 = N^a \cdot g^{r_2} \\
& \wedge 0 \le v \le V_{\mathsf{max}} \wedge B^{\mathsf{new}} = B^{\mathsf{old}} - v \ge 0 \wedge S^{\mathsf{new}} = S^{\mathsf{old}} + v \le S_{\mathsf{max}} \Big\}.
\end{aligned}$$

See Appendix A.2 for the implementation details of the relation using Sigma protocols and bulletproofs (used only for range proofs).

The receiver U_r acts as described in Figure 13.

This algorithm is similar to what has been described for U_s in Figure 12 except that $\mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}$ is generated considering $\mathsf{acc}_r^{\mathsf{old}}$ (for which user reveals $\sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}$) and v in $\psi_{s,3}$ (recall, U_r knows ρ_s).
Given $\mathsf{acc}_r^{\mathsf{old}} = (B_r^{\mathsf{old}}, S_r^{\mathsf{old}}, R_r^{\mathsf{old}}, \mathsf{sk}_r, a_r^{x_r}, a_r)$ and v, the receiver sets:

$$\mathsf{acc}_r^{\mathsf{new}} = (B_r^{\mathsf{new}}, S_r^{\mathsf{new}}, R_r^{\mathsf{new}}, \mathsf{sk}_r, a_r^{x_r+1}, a_r) \leftarrow (B_r^{\mathsf{old}} + v, S_r^{\mathsf{old}}, R_r^{\mathsf{old}} + v, \mathsf{sk}_r, a_r^{x_r} \cdot a_r, a_r)$$

Hence, the following hold for $\mathsf{acc}_r^{\mathsf{new}}$: $B_r^{\mathsf{new}} = B_r^{\mathsf{old}} + v$, $S_r^{\mathsf{new}} = S_r^{\mathsf{old}}$, $R_r^{\mathsf{new}} = R_r^{\mathsf{old}} + v$, $\mathsf{sk}_r^{\mathsf{new}} = \mathsf{sk}_r$, $a_r^{x_r+1} = a_r^{x_r} \cdot a_r$, $a_r^{\mathsf{new}} = a_r$. Additionally, the following constraints hold: $B_r^{\mathsf{new}} \leq B_{\mathsf{max}}$, $R_r^{\mathsf{new}} \leq R_{\mathsf{max}}$.[a]
The receiver transaction information, TI_r, is similar to TI_s with values associated to U_r's account:

$$\mathsf{TI}_r \leftarrow (\psi_s, \psi_r, \overline{\sigma_r}(\psi_s), \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}, \sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_r).$$

[a] Regulatory compliance $v \leq V_{\mathsf{max}}$ has already been considered in TI_s.

Fig. 13: Payment – U_r 1^{st} phase

The sender U_s (resp. receiver U_r) acts as described in Figure 14.

▷ For $k = 1$ to $D - 1$, perform the following steps:
- Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_s)$ (resp. $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_r)$).
- Wait for a response: $(\mathtt{Continue}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$.

▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_D, \mathsf{TI}_s)$ (resp. $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_D, \mathsf{TI}_r)$).

Fig. 14: Payment – U_s (resp. U_r) 1^{st} phase (continued)

Each maintainer (M_j) acts as described in Figure 15.

▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_s, \mathsf{mid}_s)$, and $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_r, \mathsf{mid}_r)$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ and parse

$$\mathsf{TI}_s = \big(\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s\big),$$
$$\mathsf{TI}_r = \big(\psi_s, \psi_r, \overline{\sigma_r}(\psi_s), \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}, \sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_r\big).$$

▷ Ignore TI_s (resp. TI_r) if at least one of the following conditions holds:
- There already exists a transaction identifier t'_{id} (either for an issuance transaction or an aborted transaction) in $\mathcal{L}_j$, where $\mathsf{T}' \in t'_{\mathsf{id}}$ and $\mathsf{T}' = \mathsf{T}_s$ (resp. $\mathsf{T}' \in t'_{\mathsf{id}}$ and $\mathsf{T}' = \mathsf{T}_r$).
- There already exists a transaction identifier t'_{id} (for a payment transaction) in $\mathcal{L}_j$, where $\mathsf{T}'_s \in t'_{\mathsf{id}}$ and $\mathsf{T}'_s = \mathsf{T}_s$ or $\mathsf{T}'_r \in t'_{\mathsf{id}}$ and $\mathsf{T}'_r = \mathsf{T}_s$ (resp. $\mathsf{T}'_s \in t'_{\mathsf{id}}$ and $\mathsf{T}'_s = \mathsf{T}_r$ or $\mathsf{T}'_r \in t'_{\mathsf{id}}$ and $\mathsf{T}'_r = \mathsf{T}_r$).
- Upon calling $\mathcal{F}_{\mathsf{SoK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$ (resp. $(\mathtt{Verify}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s))$), $(\mathtt{Verified}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r), 0)$ (resp. $(\mathtt{Verified}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s), 0)$) is received.
- Given $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$ (resp. $\sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}$), upon calling $\mathsf{VerifySig}$, 0 is received.

▷ Otherwise, record a *sender-receiver* pair $((\mathsf{TI}_s, \mathsf{TI}_r), (\mathsf{mid}_s, \mathsf{mid}_r))$ in $\mathcal{L}_j$.
▷ Set the transaction identifier $t_{\mathsf{id}} \leftarrow (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ and record it in $\mathcal{L}_j$.
▷ Given $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$ and $\mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}$, invoke $\mathsf{BlindSign}$ to obtain blind signature shares $\sigma_{s,j}^{\mathsf{new},\mathfrak{B}}$ and $\sigma_{r,j}^{\mathsf{new},\mathfrak{B}}$, which belong to U_s and U_r, respectively.
▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_s, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}})$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_r, \sigma_{r,j}^{\mathsf{new},\mathfrak{B}})$.
▷ Output $(\mathtt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{Z}$.

Fig. 15: Payment – M_j

The sender U_s (resp. receiver U_r) acts as described in Figure 16.

▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}})$ (resp. $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_{r,j}^{\mathsf{new},\mathfrak{B}})$) for different j from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$.
▷ Given $\sigma_{s,j}^{\mathsf{new},\mathfrak{B}}$, call $\mathsf{Unblind}$ to obtain $\sigma_{s,j}^{\mathsf{new}}$ (resp. given $\sigma_{r,j}^{\mathsf{new},\mathfrak{B}}$, call $\mathsf{Unblind}$ to obtain $\sigma_{r,j}^{\mathsf{new}}$).
▷ Given $\{\sigma_{s,j}^{\mathsf{new}}\}_{j=1}^{\alpha}$, call $\mathsf{TBS.Agg}$ to obtain $\sigma_{s,\mathbb{M}}^{\mathsf{new}}$ (resp. given $\{\sigma_{r,j}^{\mathsf{new}}\}_{j=1}^{\alpha}$, call $\mathsf{TBS.Agg}$ to obtain $\sigma_{r,\mathbb{M}}^{\mathsf{new}}$).
▷ Output $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ (resp. $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$) to $\mathcal{Z}$.

Fig. 16: Payment – U_s (resp. U_r) 2^{nd} phase

4.2.5 Abort transaction In the *currency issuance* and *payment* protocols, it may occur that a user's specific transaction is pending. This could mean that the transaction has passed the checks performed by maintainers; however, a sufficient number of maintainers have not yet received a valid TI for the user's counterparty. As a result, a *sender-receiver* pair has not been generated on a sufficient number of maintainers' sides,[11] which implies that the user has not yet received α valid signature shares on its new account.
Upon receiving the environment's instruction $(\mathtt{AbrTnx}, \mathsf{sid})$ to abort the most recent transaction, if the user is not in the $\mathsf{Receiving}$ or $\mathsf{Sending}$ state, it ignores the message. Otherwise, the user U sends an abort request AR to $\mathbb{M}$ by submitting its blinded refreshed account $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$, which maintains the same state as the current account, except that the transaction counter is incremented by one.

The user U executes the algorithm provided in Figure 17.

▷ Given the current (old) account state $\mathsf{acc}^{\mathsf{old}} = (B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}_{\mathsf{U}}, a^x, a)$, set the refreshed account:

$$\mathsf{acc}^{\mathsf{r}} = (B^{\mathsf{r}}, S^{\mathsf{r}}, R^{\mathsf{r}}, \mathsf{sk}_{\mathsf{U}}, a^{x+1}, a) \leftarrow (B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}_{\mathsf{U}}, a^x \cdot a, a)$$

▷ Given $\mathsf{acc}^{\mathsf{r}}$ and $\sigma_{\mathbb{M}}$, compute $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$, and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ calling $\mathsf{PrepareBlindSign}$ and $\mathsf{ProveSig}$ algorithms of the TBS scheme respectively.
▷ Set $\mathsf{T} \leftarrow g^{a^{x+1}}$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$, and receive back $(\mathtt{Proof}, \mathsf{sid}, \pi)$.

[11] Recall, e.g., in the payment protocol, the sender sends transaction information to the maintainers, and the receiver does the same. The maintainers then verify the information, and if everything is correct, they locally generate a *sender-receiver* pair associated with one transaction on their side.

Denote the randomness used to create $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ by $\mathsf{r}_{\mathsf{abr}}$. The NIZK statement and witness are as follows:

$$\mathtt{x} = (\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}), \qquad \mathtt{w} = (\mathsf{acc}, \mathsf{r}_{\mathsf{abr}})$$

The following is informal description of the NIZK relation $\mathcal{R}(\mathtt{x}, \mathtt{w})$ (see below for formal definition):

- T is well-formed; the exponent of g is the fifth element in $\mathsf{acc}^{\mathsf{r}}$.
- $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is a re-randomization of $\sigma_{\mathbb{M}}$, which is a signature generated by aggregating α different valid signature shares of maintainers on $\mathsf{acc}^{\mathsf{old}}$.
- $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ is generated considering $\mathsf{acc}^{\mathsf{old}}$. Hence, the following hold for $\mathsf{acc}^{\mathsf{r}}$: $B^{\mathsf{r}} = B^{\mathsf{old}}, \quad S^{\mathsf{r}} = S^{\mathsf{old}}, \quad R^{\mathsf{r}} = R^{\mathsf{old}}, \quad \mathsf{sk}_{\mathsf{U}}^{\mathsf{r}} = \mathsf{sk}_{\mathsf{U}}, \quad a^{x+1} = a^x \cdot a, \quad a^{\mathsf{r}} = a.$
- U knows the randomness $\mathsf{r}_{\mathsf{abr}}$.

▷ Set abort request:

$$\mathsf{AR} \leftarrow (\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi).$$

▷ For $k = 1$ to $D - 1$, perform the following steps:

- Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{AR})$.
- Wait for a response: $(\mathtt{Continue}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$.

▷ Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_D, \mathsf{AR})$.

Fig. 17: Abort transaction – U

Formal definition of abort transaction NIZK relation:

- $\mathtt{w} = \Big((B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}, \varphi, a), o, \{o_\tau\}_{\tau=1}^{6}, r, r_1, r_2\Big)$ where $\varphi = a^x$.
- $\mathtt{x} = \big(\mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}\big)$
- The relation $\mathcal{R}(\mathtt{x}, \mathtt{w})$ is defined as:

$$\begin{aligned}\mathcal{R}(\mathtt{x}, \mathtt{w}) = \Big\{ & \mathsf{com} = g^o \cdot h_1^{B^{\mathsf{old}}} \cdot h_2^{S^{\mathsf{old}}} \cdot h_3^{R^{\mathsf{old}}} \cdot h_4^{\mathsf{sk}} \cdot h_5^{\varphi^{\mathsf{old}} \cdot a} \cdot h_6^{a} \\ & \wedge \mathsf{com}_1 = g^{o_1} \cdot h^{B^{\mathsf{old}}} \wedge \mathsf{com}_2 = g^{o_2} \cdot h^{S^{\mathsf{old}}} \wedge \mathsf{com}_3 = g^{o_3} \cdot h^{R^{\mathsf{old}}} \\ & \wedge \mathsf{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}} \wedge \mathsf{com}_5 = g^{o_5} \cdot h^{\varphi^{\mathsf{old}} \cdot a} \wedge \mathsf{com}_6 = g^{o_6} \cdot h^{a} \\ & \\ & \wedge \kappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{B^{\mathsf{old}}} \cdot \tilde{\beta}_2^{S^{\mathsf{old}}} \cdot \tilde{\beta}_3^{R^{\mathsf{old}}} \cdot \tilde{\beta}_4^{\mathsf{sk}} \cdot \tilde{\beta}_5^{\varphi^{\mathsf{old}}} \cdot \tilde{\beta}_6^{a} \cdot \tilde{g}^{r} \\ & \wedge \mathsf{T} = g^{\varphi^{\mathsf{old}} \cdot a} \wedge N = g^{r_1} \cdot h^{\varphi^{\mathsf{old}}} \wedge \mathsf{com}_5 = N^{a} \cdot g^{r_2} \wedge r_2 = o_5 - a r_1 \Big\}.\end{aligned}$$

See Appendix A.3 for the implementation details of the relation using Sigma protocols.

Each maintainer M_j acts as described in Figure 18.

▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{AR}, \mathsf{mid})$ from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ and parse $\mathsf{AR} = (\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$.
▷ Ignore AR if at least one of the following conditions holds:

- There already exists a transaction identifier of the form $t'_{\mathsf{id}} = (\mathsf{Aborted}, \mathsf{T})$ in the ledger $\mathcal{L}_j$.
- Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$, $(\mathtt{Verification}, \mathsf{sid}, 0)$ is received.
- Given $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$, upon calling $\mathsf{VerifySig}$, 0 is received.
- There is no recorded *sender-receiver* pair $(\mathsf{TI}_s, \mathsf{TI}_r)$ with $\mathsf{T}' \in \mathsf{TI}_s$ or $\mathsf{T}' \in \mathsf{TI}_r$ such that $\mathsf{T}' = \mathsf{T}$.

▷ Otherwise, send $(\mathsf{TI}_s, \mathsf{TI}_r, \mathsf{mid}_s, \mathsf{mid}_r)$ to other maintainers[a] by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with:

$$(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_i, (\mathsf{TI}_s, \mathsf{TI}_r, \mathsf{mid}_s, \mathsf{mid}_r)) \quad \text{for } i = 1, \ldots, D \wedge i \neq j.$$

[a] $(\mathsf{mid}_s, \mathsf{mid}_r)$ have already been recorded—see the previous protocol.

Fig. 18: Abort transaction – M_j

Each maintainer M_i acts as described in Figure 19.

▷ Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, (\mathsf{TI}_s, \mathsf{TI}_r, \mathsf{mid}_s, \mathsf{mid}_r))$ sent by M_j.
▷ Verify $(\mathsf{TI}_s, \mathsf{TI}_r)$ by calling $\mathcal{F}_{\mathsf{NIZK}}$ and $\mathsf{VerifySig}$, and ignore the message on failure.[a] Otherwise, record $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, (\mathsf{TI}_s, \mathsf{TI}_r, \mathsf{mid}_s, \mathsf{mid}_r))$ and proceed.[b]
▷ Do not sign any account in any transaction that contains tag T' where $\mathsf{T}' = \mathsf{T}_s \in \mathsf{TI}_s$, $\mathsf{T}' = \mathsf{T}_r \in \mathsf{TI}_r$, or $\mathsf{T}' = \mathsf{T} \in \mathsf{TI}_{\mathsf{U}}$ until the decision about $(\mathsf{TI}_s, \mathsf{TI}_r)$ is made via the output of $\mathcal{F}_{\mathsf{aBA}}$, as described in the following.
▷ Check if there already exists an entry $(\mathsf{TI}_z, \mathsf{TI}_w)$ recorded in ledger $\mathcal{L}_i$ where at least one of the transaction information in the entry is different from TI_s or TI_r, and at least one of the tags used in $(\mathsf{TI}_z, \mathsf{TI}_w)$ equals one of the tags used in $(\mathsf{TI}_s, \mathsf{TI}_r)$. If so, send $(\mathsf{TI}_z, \mathsf{TI}_w, \mathsf{mid}_z, \mathsf{mid}_w)$ to other maintainers by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with:[c]

$$(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, (\mathsf{TI}_z, \mathsf{TI}_w, \mathsf{mid}_z, \mathsf{mid}_w)) \quad \text{for } j = 1, \ldots, D \wedge j \neq i.$$

▷ Otherwise, check if there already exists $(\mathsf{TI}_s, \mathsf{TI}_r)$ recorded. If so, send $(\mathsf{TI}_s, \mathsf{TI}_r, \mathsf{mid}'_s, \mathsf{mid}'_r)$[d] to other maintainers by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with:

$$(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, (\mathsf{TI}_s, \mathsf{TI}_r, \mathsf{mid}'_s, \mathsf{mid}'_r)) \quad \text{for } j = 1, \ldots, D \wedge j \neq i.$$

▷ Else, sign $(\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}})$ in $(\mathsf{TI}_s, \mathsf{TI}_r)$ and record them in $\mathcal{L}_i$ together with the associated t_{id}. Send $(\mathsf{TI}_s, \mathsf{TI}_r)$ to other maintainers by calling $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with:

$$(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, (\mathsf{TI}_s, \mathsf{TI}_r, \cdot, \cdot)) \quad \text{for } j = 1, \ldots, D \ \wedge \ j \neq i.$$

[a] M_j can be identified as malicious. M_i can submit the signature of M_j as proof of maliciousness to all other maintainers. Formally identifying malicious maintainers is out of the scope of this paper.
[b] If there already exists an entry recorded of the form $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, (\mathsf{TI}_z, \mathsf{TI}_w, \cdot, \cdot))$ where at least one of the transaction information in the entry is different from TI_s or TI_r, and at least one of the tags used in $(\mathsf{TI}_z, \mathsf{TI}_w)$ equals one of the tags used in $(\mathsf{TI}_s, \mathsf{TI}_r)$, this identifies M_j as malicious. M_i can submit the signatures of M_j as proof of maliciousness to all other maintainers. Additionally, the tuples $(\mathsf{TI}_z, \mathsf{TI}_w)$ and $(\mathsf{TI}_s, \mathsf{TI}_r)$ serve as proof of cheating by the transaction counterparty (whoever has engaged in both transactions as an honest user should finalize a transaction before engaging in another).
[c] The tuples $(\mathsf{TI}_z, \mathsf{TI}_w)$ and $(\mathsf{TI}_s, \mathsf{TI}_r)$ serve as proof of cheating by the transaction counterparty (whoever has engaged in both transactions as an honest user should finalize a transaction before engaging in another).
[d] mid'_s and mid'_r have already been recorded—see the payment protocol.

Fig. 19: Abort transaction (continued) – M_i

Users and maintainers act as described in Figure 20.

▷ Upon receiving messages from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, each maintainer acts as follows:
▷ Call $\mathcal{F}_{\mathsf{aBA}}$ with $(\mathtt{Agree}, \mathsf{sid} \mid\mid [(\mathsf{TI}_s, \mathsf{TI}_r), (\mathsf{mid}_s, \mathsf{mid}_r)], d_j)$.
- Set $d_j \leftarrow 1$ if M_j agrees to sign (based on checks in Figure 19)—or has already signed—$(\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}})$ in $(\mathsf{TI}_s, \mathsf{TI}_r)$.
- Otherwise, set $d_j \leftarrow 0$.

▷ Upon receiving the output of $\mathcal{F}_{\mathsf{aBA}}$, $(\mathtt{Agreed}, \mathsf{sid} \mid\mid [(\mathsf{TI}_s, \mathsf{TI}_r), (\mathsf{mid}_s, \mathsf{mid}_r)], Q)$, proceed as follows:
▷ If $Q = 1$:
- If accounts in $(\mathsf{TI}_z, \mathsf{TI}_w)$ have been signed, remove $(\mathsf{TI}_z, \mathsf{TI}_w)$ and the associated $t_{\mathsf{id}}^{\mathsf{zw}}$ from the ledger.
- Sign $(\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}})$ if they have not been signed yet to obtain $\sigma_s^{\mathsf{new},\mathfrak{B}}$ and $\sigma_r^{\mathsf{new},\mathfrak{B}}$.
- Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_s, \sigma_s^{\mathsf{new},\mathfrak{B}})$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_r, \sigma_r^{\mathsf{new},\mathfrak{B}})$.
- Record $(\mathsf{TI}_s, \mathsf{TI}_r)$ together with its associated $t_{\mathsf{id}}^{\mathsf{sr}}$.
- Output $(\mathtt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}}^{\mathsf{sr}})$ to $\mathcal{Z}$.

The user U_s (resp. receiver U_r) acts as follows:
– Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_k, \sigma_{s,k}^{\mathsf{new},\mathfrak{B}})$ (resp. $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_k, \sigma_{r,k}^{\mathsf{new},\mathfrak{B}})$) for different k from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$.
– Similar to payment, unblind and aggregate the received signatures.
– Output $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ to $\mathcal{Z}$ (resp. output $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$).

▷ Else $(Q = 0)$:

- If $(\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}})$ in $(\mathsf{TI}_s, \mathsf{TI}_r)$ have been signed and thus $(\mathsf{TI}_s, \mathsf{TI}_r)$ have been recorded, remove them together with the associated $t_{\mathsf{id}}^{\mathsf{sr}}$.
- Verify AR by calling $\mathcal{F}_{\mathsf{NIZK}}$ and $\mathsf{VerifySig}$. Ignore if it does not verify.
- Otherwise, call $\mathcal{F}_{\mathsf{aBA}}$ with $(\mathtt{Agree}, \mathsf{sid} \mid\mid [\mathsf{AR}, \mathsf{mid}], d_j)$.

– Set $d_j \leftarrow 1$ if M_j signs $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ (e.g., $\mathcal{F}_{\mathsf{NIZK}}$ and $\mathsf{VerifySig}$ checks are passed).
– Otherwise, set $d_j \leftarrow 0$.

- Upon receiving the output of $\mathcal{F}_{\mathsf{aBA}}$, $(\mathtt{Agree}, \mathsf{sid} \mid\mid [\mathsf{AR}, \mathsf{mid}], Q')$, proceed as follows:
- If $Q' = 1$:

– Save the aborted transaction identifier $t_{\mathsf{id}} = (\mathsf{Aborted}, \mathsf{T})$ in the ledger.
– Sign the refreshed-blinded account of the user $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ to obtain $\sigma^{\mathsf{r},\mathfrak{B}}$.
– Call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}, \sigma^{\mathsf{r},\mathfrak{B}})$.
– Output $(\mathtt{TnxAborted}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{Z}$.

- Else, ignore.

The user U acts as follows:
– Receive $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{r},\mathfrak{B}})$ for different j from $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$.
– Similar to payment, unblind and aggregate the received signatures.
– Output $(\mathtt{TnxAborted}, \mathsf{sid})$ to $\mathcal{Z}$.

Fig. 20: Abort transaction (continued) – maintainers and users

4.2.6 Auditing To achieve auditability, we make use of trust dispersal (cf. [1]). Users trust multiple independent authorities, ensuring that no single authority holds unlimited

power over any user. Hence, for privacy revocation and user tracing, we leverage threshold cryptography. Executing any type of auditing requires the participation of at least $\beta = t+1$ maintainers, where t is the maximum number of maintainers that can be corrupted by the adversary. Since we have set the threshold of the TBS scheme to $\alpha = D - t$, there always exist at least $D - 2t$ honest maintainers who have the transaction identifier t_{id} of a finalized transaction stored in their ledgers. The auditing protocol consists of two sub-protocols: *privacy revocation* and *tracing*, which are described in the following.

Privacy revocation: Given a privacy-preserved payment made by a specific *sender-receiver* pair, the audit committee revokes the privacy of the transaction by decrypting the ciphertexts and identifying the transaction participants and the transaction value (see Figure 21).

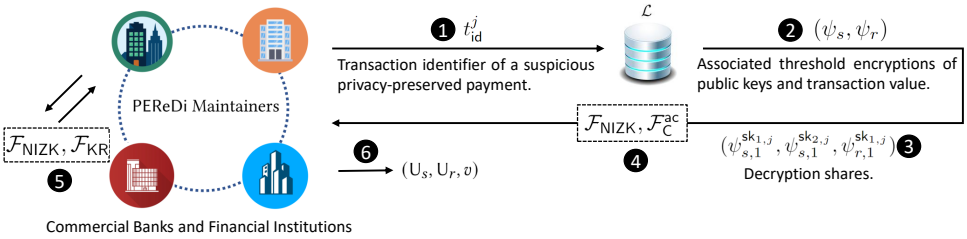

Fig. 21: *Privacy revocation* protocol

The algorithm executed by the j-th maintainer M_j is described in Figure 22.

▷ Upon receiving $(\texttt{RvkAnm}, \mathsf{sid}, t^j_{\mathsf{id}})$ from $\mathcal{Z}$, act as follows.
▷ Locate the associated (ψ_s, ψ_r) stored in the ledger[a] $\mathcal{L}_j$ for the given t^j_{id}.
▷ Parse

$$\psi_s = (\psi_{s,1}, \psi_{s,2}, \psi_{s,3}), \quad \psi_r = (\psi_{r,1}, \psi_{r,2}).$$

▷ Compute the decryption shares:

$$A_j \leftarrow \psi^{\mathsf{sk}_{1,j}}_{s,1}, \quad B_j \leftarrow \psi^{\mathsf{sk}_{2,j}}_{s,1}, \quad C_j \leftarrow \psi^{\mathsf{sk}_{1,j}}_{r,1}.$$

▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Prove}, \mathsf{sid}, \mathtt{x}_j, \mathtt{w}_j)$, and receive back $(\texttt{Proof}, \mathsf{sid}, \pi_j)$, where the NIZK statement and witness are:

$$\mathtt{x}_j = (\psi_{s,1}, \psi_{r,1}, A_j, B_j, C_j), \qquad \mathtt{w}_j = (\mathsf{sk}_{1,j}, \mathsf{sk}_{2,j})$$

for the following relation $\mathcal{R}(\mathtt{x}_j, \mathtt{w}_j)$:

$$\{\log_g \mathsf{pk}_{1,j} = \log_{\psi_{s,1}} A_j, \quad \log_g \mathsf{pk}_{2,j} = \log_{\psi_{s,1}} B_j, \quad \log_g \mathsf{pk}_{1,j} = \log_{\psi_{r,1}} C_j\}.$$

▷ Call $\mathcal{F}^{\mathsf{ac}}_{\mathsf{Ch}}$ with:

$$(\mathsf{Send}, \mathsf{sid}, \mathsf{M}_i, (\mathtt{x}_j, \pi_j)) \quad \text{for } i = 1, \ldots, D \wedge i \neq j.$$

▷ Upon receiving $(\texttt{Received}, \mathsf{sid}, \mathsf{M}_i, (\mathtt{x}_i, \pi_i))$ from $\mathcal{F}^{\mathsf{ac}}_{\mathsf{Ch}}$, proceed as follows.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \mathsf{sid}, \mathtt{x}_i, \pi_i)$, and ignore if $(\texttt{Verification}, \mathsf{sid}, 0)$ is received.
▷ Upon obtaining β valid decryption shares, compute pk_s, g^v, and pk_r as follows:

$$\mathsf{pk}_s = \frac{\psi_{s,2}}{\prod_{j \in I} A_j^{\lambda_{1,j}}}, \quad g^v = \frac{\psi_{s,3}}{\prod_{j \in I} B_j^{\lambda_{2,j}}}, \quad \mathsf{pk}_r = \frac{\psi_{r,2}}{\prod_{j \in I} C_j^{\lambda_{1,j}}}$$

such that $|I| = \beta$ and λ is the Lagrange coefficient. Compute v from g^v.[b]

▷ Call $\mathcal{F}_{\mathsf{KR}}$ with $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{pk}_s)$ and $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{pk}_r)$ to retrieve the unique identifiers of users by receiving $(\texttt{IDRetrieved}, \mathsf{sid}, \mathsf{U}_s, \mathsf{pk}_s)$ and $(\texttt{IDRetrieved}, \mathsf{sid}, \mathsf{U}_r, \mathsf{pk}_r)$ from $\mathcal{F}_{\mathsf{KR}}$.
▷ Output $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$ to $\mathcal{Z}$.

[a] For a currency issuance transaction, given that the sender is B, the cryptographic information saved for auditing contains only ψ. However, in this algorithm, we describe the steps of the *privacy revocation* protocol for a payment transaction; the process for a currency issuance transaction is similar.
[b] Note that to have efficient zero-knowledge and signature of knowledge proofs, the user sets g^v as one of the plaintexts in ψ. One of the system's regulatory compliance measures is enforcing a limit on transaction value $v < V_{\mathsf{max}}$, which makes extracting v from g^v efficient for $\mathbb{M}$ in this sub-protocol. Moreover, as proposed in [56], implementing a constant-time decryption mechanism can remove the need for brute-force attempts.

Fig. 22: Privacy revocation – M_j

Tracing: Given a suspicious user's unique identifier, the audit committee traces all transactions made by that user (see Figure 23). First, they locate the user's record generated during the *user registration* protocol. Using secret shares of a, maintainers compute all tracing tags of the user without revealing a. We will see that, to achieve simulatability, a must not be revealed.

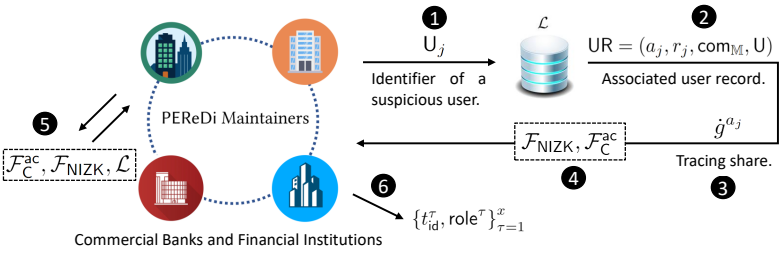

Fig. 23: *Tracing* protocol

The maintainers jointly compute tracing tags such that the last computation results in a tag that does not exist in their ledgers. In this way, tracing authorities can determine the most recent transaction of U. As described in the *currency issuance* and *payment* protocols, all transactions contain tracing tag values of the form g^{a^x}, where a is the user's (tracing tag) secret key and x is the transaction counter (recall that for aborted transactions, the user also increments x by one).
The threshold for TBS is α, ensuring that at least β honest maintainers always have the transaction identifier t_{id} of a specific transaction saved in their ledgers. However, the number of honest maintainers who possess the complete set of t_{id} values for all transactions of a specific user is not necessarily β, as we do not enforce any agreement in the *currency issuance* or *payment* protocols. Thus, we must ensure that, at each step of threshold tag computation, all maintainers can compute the tag T and then check their ledgers to determine whether such a tag already exists. They achieve this by sending their next tag-computation shares in a provable manner to others so that, once β shares are collected, the next tag can be computed. This process continues until maintainers do not find the computed tag T in their ledgers.

For a currency issuance transaction, t_{id} contains only the tracing tag of the receiver, whereas for a payment transaction, it contains the tracing tags of both the sender and the receiver. Based on the computed tracing tags, each maintainer determines whether the traced user was the sender or the receiver of the transaction for which t_{id} is retrieved (the sender's tag appears first in t_{id}). Hence, maintainers output $\{t^{\tau}_{\mathsf{id}}, \mathsf{role}^{\tau}\}^{x}_{\tau=1}$ to $\mathcal{Z}$, where role can be either sender or $\mathsf{receiver}$. Note that given the values $\{t^{\tau}_{\mathsf{id}}\}^{x}_{\tau=1}$, the counterparties of the suspicious user can be identified using the *privacy revocation* protocol. To ensure efficient tracing, in the *user registration* protocol, each user proves that x starts from 1 and increments by one for each transaction.

The algorithm executed by the j-th maintainer M_j is described in Figure 24.

▷ Upon receiving $(\texttt{Trace}, \mathsf{sid}, \mathsf{U}_j)$ from $\mathcal{Z}$, act as follows.
▷ Locate the associated user record $\mathsf{UR} = (a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{U})$ stored in $\mathcal{L}_j$, where $\mathsf{U} = \mathsf{U}_j$.
▷ Parse $\mathsf{com}_{\mathbb{M}} = \{\tilde{\mathsf{com}}_j\}^{D}_{j=1}$.
▷ (*) Set $\dot{g} \leftarrow g^{a_j{}^e}$ (with e initially set to 0). Set $E_j \leftarrow \dot{g}^{a_j}$. Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Prove}, \mathsf{sid}, \overline{\mathtt{x}}_j, \overline{\mathtt{w}}_j)$, and receive $(\texttt{Proof}, \mathsf{sid}, \overline{\pi}_j)$ from $\mathcal{F}_{\mathsf{NIZK}}$, where the NIZK statement and witness are:[a]

$$\overline{\mathtt{x}}_j = (\tilde{\mathsf{com}}_j, E_j, \dot{g}), \qquad \overline{\mathtt{w}}_j = (a_j, r_j)$$

The relation $\mathcal{R}(\overline{\mathtt{x}}_j, \overline{\mathtt{w}}_j)$ is defined as:

$$\{\tilde{\mathsf{com}}_j = g^{a_j} \cdot h^{r_j} \wedge E_j = \dot{g}^{a_j}\}.$$

▷ Call $\mathcal{F}^{\mathsf{ac}}_{\mathsf{Ch}}$ with:

$$(\texttt{Send}, \mathsf{sid}, \mathsf{M}_i, (\overline{\mathtt{x}}_j, \overline{\pi}_j)) \quad \text{for } i = 1, \ldots, D \wedge i \neq j.$$

▷ Upon receiving $(\texttt{Received}, \mathsf{sid}, \mathsf{M}_i, (\overline{\mathtt{x}}_i, \overline{\pi}_i))$ from $\mathcal{F}^{\mathsf{ac}}_{\mathsf{Ch}}$, proceed as follows.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \mathsf{sid}, \overline{\mathtt{x}}_i, \overline{\pi}_i)$, and ignore if $(\texttt{Verification}, \mathsf{sid}, 0)$ is received.
▷ Upon obtaining β valid tracing shares, compute $\dot{g}^a$ as follows:

$$\dot{g}^a = \prod_{j \in I} E_j^{\lambda_j}$$

where $|I| = \beta$ and λ is the Lagrange coefficient.
▷ If a transaction identifier t_{id} (for issuance, payment, or an aborted transaction) already exists in the ledger $\mathcal{L}_j$ that includes $\dot{g}^a$ as a tag T, proceed from (*) setting $\dot{g}$ step above with $e \leftarrow e + 1$ and record the associated t_{id} of the computed T along with role.
▷ Otherwise, send a message to all maintainers by calling $\mathcal{F}^{\mathsf{ac}}_{\mathsf{Ch}}$ with input:

$$(\texttt{Send}, \mathsf{sid}, \mathsf{M}_i, (0, \dot{g}^a)) \quad \text{for } i = 1, \ldots, D \wedge i \neq j.$$

indicating that M_j has not seen $\dot{g}^a$ in $\mathcal{L}_j$.
▷ Upon receiving $D - t$ messages of the form $(\texttt{Received}, \mathsf{sid}, \mathsf{M}_i, (0, \dot{g}^a))$, where $\dot{g} = g^{a^e}$ from $\mathcal{F}^{\mathsf{ac}}_{\mathsf{Ch}}$, output the recorded transaction identifiers and corresponding roles $(\texttt{Traced}, \mathsf{sid}, \mathsf{U}, \{t^{\tau}_{\mathsf{id}}, \mathsf{role}^{\tau}\}^{x}_{\tau=1})$ to $\mathcal{Z}$, and abort.

[a] The maintainer proves that the share contributed to the threshold tag computation is consistent with the j-th commitment $\tilde{\mathsf{com}}_j \in \mathsf{com}_{\mathbb{M}}$ (broadcasted during the *user registration* protocol).

Fig. 24: Tracing – M_j

4.3 Binding real-world identities to transactions

In our formal modeling, $\mathcal{F}_{\mathsf{CBDC}}$ captures well-known regulatory compliance rules such as balance limits and receiving and sending limits specified by various central banks. To address more general regulations, users can bind their unique real-world identities to their transactions. This enables them to prove different attributes associated with their real-world identities (without revealing them) in every transaction, thereby addressing general KYC regulations in a privacy-preserving manner.
For this purpose, anonymous credentials [30,31,22] can be used. However, due to the importance of efficiency in our setting, a recently introduced anonymous signature scheme called SyRA signatures [33] can be utilized. SyRA signatures incorporate the unique real-world identity of the signer and are efficient to generate and verify. This allows users to prove their adherence to regulations such as AML and KYC in an anonymous manner based on their real-world identities. Each PEReDi transaction information TI can be signed using SyRA signatures while proving attributes about identity. PEReDi maintainers can serve as distributed SyRA issuers.

4.4 On abort requests and a lower bound on D

Recall that in our construction, we neither use Byzantine broadcast nor Byzantine agreement in the optimistic execution path of a payment. However, in the pessimistic execution path, handling an abort request AR requires a supermajority of honest maintainers while maintaining asynchronous operation. In the following, we discuss the upper limit on the number of maintainers that can be corrupted by the adversary in our asynchronous setting such that the construction can handle the pessimistic execution path.
To understand the difficulty of handling abort requests, consider a malicious user that initiates multiple transactions with the same account state. Since the adversary controls the communication channel between users and maintainers, they can easily create conflicting views among honest maintainers, causing confusion about which transaction to confirm and which to discard in the case of an abort request AR. For example, a malicious user A, either as a sender or receiver, can engage in two different transactions with users B and C, generating two sets of transaction information objects and sending them to maintainers (B and C can be honest and/or malicious). Consequently, honest maintainers have differing local views about handling a potential abort request. Some maintainers may record $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$, while others may record $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$. Below, we argue that with fewer than $5t+1$ maintainers, it is impossible to correctly handle an abort request in an asynchronous setting.
We examine settings with $D \leq 5t$ maintainers. Without loss of generality, we focus on the $5t$ case. Consider a scenario in which user A has transactions $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ and $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$. Suppose $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ is signed by the majority of honest maintainers ($3t$) and all malicious maintainers (t), while $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$ is signed by a minority of honest maintainers (t) and all malicious maintainers (t). Recall that any agreement (e.g., by running asynchronous interactive consistency) among the maintainers should rely on messages from $D - t = 4t$ maintainers. Moreover, the adversary can delay a group of t honest maintainers and provide false claims on behalf of the t malicious maintainers. Consider now an adversary that falsely claims they have not signed $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ by showing their signature on $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$. This leads to a view for honest maintainers in which there are $2t$ votes for $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ and $2t$ votes for $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$, while the transaction containing $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ is already finalized, and the honest maintainers must make a decision about the abort request. A symmetric protocol configuration can be produced by reversing TI_B and TI_C. It follows that the following two configurations are hard to distinguish for honest maintainers:

(i) $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ is finalized and $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$ is pending.
(ii) $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$ is finalized and $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ is pending.

Observe that without privacy, a reasonable decision would be to reject both $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{B})$ and $(\mathsf{TI}_\mathsf{A}, \mathsf{TI}_\mathsf{C})$ once each receives $2t$ votes, provided no account state has been updated as a

result of these transactions. On the other hand, if a user has advanced their account state and created any of those transactions, the maintainers can run a protocol to check the (previous) state and finalize the transaction. As a result, the system can accurately update its state because the malicious user A cannot use an account generated from a transaction if it has already been decided to reject that transaction. Moreover, the transaction can never be rejected in the future if at least one transaction counterparty has already used the account created from the transaction. However, in our fully anonymous setting, where transactions are unlinkable to each other, it is not straightforward how to implement this procedure. Potentially, by using an MPC protocol, we could solve this issue in a privacy-preserving manner. However, this would require using MPC for every single submitted transaction and checking the incoming transaction against the whole state of the system, which would significantly reduce efficiency. It follows that $D \geq 5t+1$ is necessary for any construction that is both efficient and privacy-preserving. As we demonstrate in Section 5, choosing $D = 5t+1$ is also sufficient to realize $\mathcal{F}_{\mathsf{CBDC}}$ in the asynchronous setting.

4.5 Know your transaction for large payments

Enforcing limits on transaction value and the sum of all sent values are two general regulatory rules. The maximum allowed value for the former is denoted by V_{max}, and for the latter, it is denoted by S_{max}. While such limits serve a purpose, a user may need to exceed them when making a large payment. Even though we do not include this feature in our main functionality, we describe in this section how to realize it given our construction.
In such cases, regulatory compliance may require proving the source of funds. Under these circumstances, the user can exceed the specified thresholds up to the new limits V'_{max} and S'_{max}. The new limit is computed by adding all values whose sources are verified as legitimate to the predefined general limit. For instance, consider a scenario in which a user has accumulated funds over a long period of time and now wishes to spend them all at once (e.g., purchasing a property). This would result in a transaction value far exceeding V_{max} (note that we assume $B_{\mathsf{max}} \gg V_{\mathsf{max}}$; otherwise, this mechanism would not be necessary). The user saves the relevant information about transactions for which they will make a claim. Specifically, the user refers to transaction identifiers of past transactions in which they received funds from an acceptable source. The user can submit such a claim to $\mathbb{M}$, which will facilitate exceeding the predefined thresholds.
We denote the sum of all values for which the user makes a claim by δ. Following the explanation above, we obtain the updated limits: $V'_{\mathsf{max}} = V_{\mathsf{max}} + \delta$, and $S'_{\mathsf{max}} = S_{\mathsf{max}} + \delta$. The user references the transaction identifiers of l past transactions, $\{t^{\tau}_{\mathsf{id}}\}^{l}_{\tau=1}$, which contain the associated threshold encryptions $\{(\psi_s, \psi_r)^{\tau}\}^{l}_{\tau=1}$. Given that the user knows the randomness of these threshold encryptions, they provide proof of knowledge and demonstrate that the sum of all values in the threshold encryptions equals δ. Moreover, by using the corresponding random values, the user convinces $\mathbb{M}$ regarding the sender of the transactions. More generally, $\mathbb{M}$ can designate a third-party auditor to verify the user's claim. In this case, the user only needs to present a certification of this transaction issued by the auditor.

5 PEReDi security

Our main theorem is given below.

Theorem 1. *Assuming that ElGamal encryption (Definition 10) is (i) unconditionally correct (Definition 8), (ii) computationally* IND-CPA *secure (Definition 9), Pointcheval-Sanders signature (Definition 3) is (i) unconditionally correct (Definition 1) (ii) computationally* EU-CMA *secure (Definition 2) in the random oracle model, Pedersen commitment (Definition 19) is (i) unconditionally correct (Definition 16), (ii) unconditionally hiding (Definition 17), and (iii) computationally binding (Definition 18), and d-strong decisional Diffie-Hellman (d-sDDH) assumption (Definition 6), there exist two polynomials p_c and p_u such*

that no PPT environment $\mathcal{Z}$ can distinguish the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PEReDi}},\mathcal{A},\mathcal{Z}}$ from the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ in the $\{\mathcal{F}_{\mathsf{KR}},\mathcal{F}_{\mathsf{Ch}},\mathcal{F}_{\mathsf{aBA}},\mathcal{F}_{\mathsf{BC}},\mathcal{F}_{\mathsf{RO}},\mathcal{F}_{\mathsf{NIZK}},\mathcal{F}_{\mathsf{SoK}}\}$-hybrid model with static corruptions in the presence of arbitrary number of malicious users, up to t malicious maintainers out of $D = 5t+1$ total maintainers and a potentially malicious central bank with advantage better than:

$$p_c \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{EU\text{-}CMA}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}} + p_u \cdot \mathsf{Adv}^{d\text{-}\mathsf{sDDH}}_{\mathcal{A}}.$$

We denote the real-world protocol and adversary by Π_{PEReDi} and $\mathcal{A}$, respectively. The simulator $\mathcal{S}$, described in detail in Section 5.2, makes the view of the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PEReDi}},\mathcal{A},\mathcal{Z}}$ and the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ for any PPT environment $\mathcal{Z}$ indistinguishable. The session identifier, denoted by sid, is chosen by $\mathcal{Z}$. The simulator $\mathcal{S}$ internally runs a version of Π_{PEReDi} and ensures that the view of the dummy adversary $\mathcal{A}$ in the ideal world is indistinguishable from its view in the real world. At the inception of the execution, $\mathcal{Z}$ triggers $\mathcal{A}$ to corrupt parties with a message $(\mathtt{Corrupt}, \mathsf{sid}, \mathsf{P})$, where P denotes a party that can be any entity in the network. $\mathcal{S}$ reads these corruption messages and informs $\mathcal{F}_{\mathsf{CBDC}}$ which parties are corrupted by sending the message $(\mathtt{Corrupt}, \mathsf{sid}, \mathsf{P})$. The simulator $\mathcal{S}$ also stores the identifiers of the corrupted parties. $\mathcal{S}$ internally emulates the functionalities $\mathcal{F}_{\mathsf{KR}}, \mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{aBA}}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{BC}}$, and $\mathcal{F}_{\mathsf{SoK}}$. The adversary $\mathcal{A}$ instructs corrupted parties arbitrarily. The simulator $\mathcal{S}$ interacts with $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of the corrupt parties. In the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$, honest (dummy) parties forward their input from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{CBDC}}$.

5.1 Sequence of games and reductions

Through a sequence of games, we show that the random variables $\mathsf{EXEC}_{\Pi_{\mathsf{PEReDi}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ are statistically close. We denote by $\Pr[\mathsf{Game}^{\mathsf{i}}]$ the probability that the environment $\mathcal{Z}$ outputs 1 in $\mathsf{Game}^{\mathsf{i}}$. Each game $\mathsf{Game}^{\mathsf{i}}$ has its own $\mathcal{F}^{i}_{\mathsf{CBDC}}$ and $\mathcal{S}^{i}$. We start from the most leaky functionality $\mathcal{F}^{1}_{\mathsf{CBDC}}$ and the associated simulator $\mathcal{S}^{1}$ and gradually move toward the main functionality $\mathcal{F}_{\mathsf{CBDC}}$ and the simulator $\mathcal{S}$. For the security analysis, without loss of generality, we set the number of malicious maintainers equal to the maximum allowed number of malicious maintainers (wherever necessary).

Game^{0}: Initially, $\mathcal{F}^{0}_{\mathsf{CBDC}}$ forwards all communication with $\mathcal{Z}$, and the simulator $\mathcal{S}^{0}$ corresponds to the execution of the real-world protocol $\mathsf{EXEC}_{\Pi_{\mathsf{PEReDi}},\mathcal{A},\mathcal{Z}}$.

Game^{1}: Same as Game^{0} except that Game^{1} checks if $\mathcal{A}$ provides two commitments com and com' where $\mathsf{com} = \mathsf{com}'$ and $\mathsf{com} \in \mathsf{x}$, and $\mathsf{com}' \in \mathsf{x}'$ (with associated proofs π and π', respectively), but with different committed values.

More specifically, in Game^{1}, $\mathcal{F}^{1}_{\mathsf{CBDC}}$ prohibits $\mathcal{S}^{1}$ from submitting any message to $\mathcal{F}^{1}_{\mathsf{CBDC}}$ on behalf of the adversary $\mathcal{A}$ if $\mathcal{A}$ provides two different messages with the same associated commitment.

The simulator $\mathcal{S}^{(1)}$, which emulates $\mathcal{F}_{\mathsf{NIZK}}$, extracts witnesses by submitting $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}, \pi)$ and $(\mathtt{Verify}, \mathsf{sid}, \mathsf{x}', \pi')$ to $\mathcal{A}$. $\mathcal{S}^{(1)}$ receives $(\mathtt{Witness}, \mathsf{sid}, \mathsf{w})$ and $(\mathtt{Witness}, \mathsf{sid}, \mathsf{w}')$ from $\mathcal{A}$. If, with the extracted witnesses, the committed values are different, a flag is raised. Therefore, any difference between Game^{1} and Game^{0} is due to breaking the binding property of the underlying Pedersen commitment, which enables us to bound the probability that $\mathcal{Z}$ distinguishes Game^{1} from Game^{0} as follows[12].

$$|\Pr[\mathsf{Game}^{1}] - \Pr[\mathsf{Game}^{0}]| \leq \mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}}$$

Game^{2}: Same as Game^{1} except that in Game^{2} we change the w-th honest maintainer's blind signature share on U's account to $\sigma^{\mathfrak{B}}_{w}$, which is simulated by $\mathcal{S}^{2}$. To do so, in this game, $\mathcal{S}^{2}$ selects the secret signing key of the non-threshold Pointcheval-Sanders signature

[12] As this is straightforward, we refrain from providing a formal proof.

and then computes the non-threshold signature $\sigma_{\mathbb{M}} = (h, s)$. Note that after Game^2, the simulator $\mathcal{S}^i$ for $i \geq 2$ never uses the secret signing key of the non-threshold signature scheme (as we will see, it receives a non-threshold signature from the challenger of the non-threshold signature's unforgeability game—see Definition 2—in the associated reduction). Additionally, by selecting the secret key of malicious maintainers, $\mathcal{S}^2$ computes the partial blind signatures of malicious maintainers, denoted by $\sigma_t^{\mathfrak{B}}$ for $t \in \mathcal{C}$. As a result of having $\sigma_{\mathbb{M}}$ and $\sigma_t^{\mathfrak{B}}$ for $t \in \mathcal{C}$, the simulator $\mathcal{S}^2$ computes honest maintainers' signature shares $\sigma_w^{\mathfrak{B}}$ for $w \in \mathcal{H}$ as follows.

When $\mathcal{A}$ initiates the protocol by requesting a signature on the blinded account $\mathsf{acc}^{\mathfrak{B}}$, $\mathcal{S}^2$, which emulates $\mathcal{F}_{\mathsf{NIZK}}$ in *currency issuance* and $\mathcal{F}_{\mathsf{SoK}}$ in *payment* protocols, extracts the witness of $\mathcal{A}$'s (malicious user's) message, namely acc, and the associated randomness of $\mathsf{acc}^{\mathfrak{B}}$. Then, having the message acc, $\mathcal{S}^2$ computes $\sigma_{\mathbb{M}}$. $\mathcal{S}^2$ selects the secret keys of malicious maintainers and computes the associated public keys. $\mathcal{S}^2$ uses Lagrange interpolation to compute public keys of $\mathsf{M}_w \in \mathbf{H}$ using the computed public keys for $\mathsf{M}_t \in \mathbf{C}$ and the public key of the non-threshold signature. Hence, all public keys are consistent with the public key of the non-threshold signature.

First, $\mathcal{S}^2$ computes blind signature shares for $\forall \mathsf{M}_t \in \mathbf{C}$ using the selected $\mathsf{sk}_t = (x_t, \{y_{t,\tau}\}_{\tau=1}^q)$ to obtain

$$\sigma_t^{\mathfrak{B}} = (h, h^{x_t} \prod_{\tau=1}^{q} \mathsf{com}_\tau^{y_{t,\tau}})$$

As described above, $\mathcal{S}^2$ has extracted the witness of NIZK or SoK proofs; hence, it knows $\{o_\tau\}_{\tau=1}^q$, which allows it to compute unblinded signature shares in the following way:

$$\sigma_t = (h, c \prod_{\tau=1}^{q} \beta_{t,\tau}^{-o_\tau}) = (h, s_t)$$

where $s_t = h^{x_t} \prod_{\tau=1}^q h^{m_\tau y_{t,\tau}}$. Then, $\mathcal{S}^2$ computes unblinded signature shares for $\forall \mathsf{M}_w \in \mathbf{H}$ as follows (note that 0 does not exist in the corrupted maintainers' indexes $\mathcal{C}$):

$$\sigma_w = (h, s_w) = \left(h, s^{\prod_{k \in \mathcal{C}}((k-w)/k)} \prod_{t \in \mathcal{C}} s_t^{\prod_{k \in \mathcal{C} \cup \{0\}, k \neq t}((w-k)/(t-k))} \right)$$

Having extracted the witness $\{o_\tau\}_{\tau=1}^q$, the simulator computes blind signature shares for $\forall \mathsf{M}_w \in \mathbf{H}$ using the computed σ_w as follows:

$$\sigma_w^{\mathfrak{B}} = (h, \prod_{\tau=1}^{q} s_w \beta_{w,\tau}^{o_\tau})$$

As a result, in this game, we changed the w-th honest maintainer's blind signature share on U's account to $\sigma_w^{\mathfrak{B}}$, which is simulated by $\mathcal{S}^2$ as described above. Based on the $\mathsf{Unblind}$ algorithm, which is run by $\mathcal{A}$, the unblinded signature is computed as follows:

$$\sigma_w = (h, c \prod_{\tau=1}^{q} \beta_{w,\tau}^{-o_\tau})$$

for C simulated by the simulator as

$$c = \prod_{\tau=1}^{q} s_w \beta_{w,\tau}^{o_\tau}$$

As a result, we have $\sigma_w = (h, s_w)$, which passes the verification algorithm

$$e(h, \tilde{\alpha}_w \prod_{\tau=1}^{q} \tilde{\beta}_{w,\tau}^{m_\tau}) = e(s_w, \tilde{g})$$

which means that the following equation holds:

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

Game^3: Same as Game^2 except that in Game^3, $\mathcal{F}^3_{\mathsf{CBDC}}$ does not allow $\mathcal{S}^3$ to submit any message on behalf of the adversary $\mathcal{A}$ (malicious user) who forges the threshold blind signature TBS scheme to $\mathcal{F}^3_{\mathsf{CBDC}}$. Hence, Game^3 is equivalent to Game^2 except for the fact that it checks whether a flag is raised. If $\mathcal{A}$, who has not been issued at least $4t+1$ signature shares, submits a valid signature, the flag is raised. Hence, any difference between Game^3 and Game^2 is due to the forgery of the threshold blind signature TBS, which allows us to bound the probability that $\mathcal{Z}$ distinguishes Game^3 from Game^2 as follows.

Associated reduction (existential unforgeability of signature). If $\mathcal{A}$ forges the threshold blind signature TBS used in our construction, it can be used to construct another adversary $\mathcal{A}'$ who breaks the unforgeability property (see Definition 2) of the non-threshold Pointcheval-Sanders signature used in the threshold blind signature TBS scheme. The partial blind signatures of all honest maintainers $\sigma^{\mathfrak{B}}_w$ for $\forall w \in \mathcal{H}$ can be reconstructed from the partial blind signatures of malicious maintainers, which are $\sigma^{\mathfrak{B}}_t$ for $t \in \mathcal{C}$, and the non-threshold signature $\sigma_{\mathbb{M}} = (h, s)$ obtained from the challenger of the existential unforgeability game (different from Game^2, in which $\mathcal{S}^2$ selected the secret signing key of the non-threshold signature) using Lagrange interpolation for the other shares. We omit the details, as the algorithm is similar to what $\mathcal{S}^2$ does in Game^2, except that $\mathcal{A}'$ obtains the non-threshold signature $\sigma_{\mathbb{M}}$ from the challenger. Hence, given the non-threshold signature, $\mathcal{A}'$ simulates the entire view of $\mathcal{A}$, including the partial signatures contributed by the honest maintainers, which implies that $\mathcal{A}$ cannot forge messages in the threshold setting of our construction unless $\mathcal{A}$ forges them in the non-threshold one. In other words, for $\mathcal{Z}$, Game^3 is equivalent to running a threshold signature TBS with real-world maintainers rather than maintainers simulated by $\mathcal{A}'$. Thus, if $\mathcal{A}$ forges in the real world, it will also forge in this threshold setting, and $\mathcal{A}'$ will use this forgery as a forgery for the non-threshold scheme. As a result, TBS is simulatable, and together with the unforgeability property of the non-threshold Pointcheval-Sanders signature, this ensures that TBS is unforgeable in our construction.
Therefore, under the unforgeability property of the non-threshold Pointcheval-Sanders signature, the following inequality holds (see Definition 2):

$$|\Pr[\mathsf{Game}^3] - \Pr[\mathsf{Game}^2]| \leq \mathsf{Adv}^{\mathsf{EU\text{-}CMA}}_{\mathcal{A}}$$

Game^4: Same as Game^3 except that $\mathcal{S}^4$ computes $\sigma^{\mathsf{Rnd}}_{\mathbb{M}}$ for the honest user without knowing the account values of the user. In the real world, having the consolidated signature $\sigma_{\mathbb{M}} = (h, s)$, $\sigma^{\mathsf{Rnd}}_{\mathbb{M}}$ is computed as $(h', s') = (h^{r'}, s^{r'} h^{r'r})$ such that $r \xleftarrow{\$} \mathbb{Z}_p$ and $r' \xleftarrow{\$} \mathbb{Z}_p$. Assume a random value η, and set $h = g^{\eta}$ by programming the random oracle. Hence, we have

$$\sigma^{\mathsf{Rnd}}_{\mathbb{M}} = \left(h^{r'}, \left(\prod_{j \in E} \left(h^{x_j} \prod_{\tau=1}^{q} \mathsf{com}_{\tau}^{y_{j,\tau}} \beta_{j,\tau}^{-o_\tau} \right)^{l_j} \right)^{r'} h^{r'r} \right)$$

$$= \left(g^{\eta r'}, g^{\eta r' (x + \sum_{\tau=1}^{q} (m_\tau y_\tau) + r)} \right) \xrightarrow[\eta r' = d']{(x + \sum_{\tau=1}^{q} (m_\tau y_\tau) + r) = d} \sigma^{\mathsf{Rnd}}_{\mathbb{M}} = (g^{d'}, g^{dd'})$$

Also, in the real world, in the ProveSig algorithm, the user U computes κ as well, which is of the form:

$$\kappa = \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_{\tau}^{m} \tilde{g}^{r} = \tilde{g}^{(x + \sum_{\tau=1}^{q} (y_\tau m_\tau) + r)} \xrightarrow{(x + \sum_{\tau=1}^{q} (m_\tau y_\tau) + r) = d} \kappa = \tilde{g}^{d}$$

$\mathcal{S}^4$ randomly selects $u \xleftarrow{\$} \mathbb{Z}_p$ and $u' \xleftarrow{\$} \mathbb{Z}_p$. Then, it sets $h' \leftarrow g^{u'}$, $s' \leftarrow g^{uu'}$ and hence sets $\sigma^{\mathsf{Rnd}}_{\mathbb{M}} \leftarrow (g^{u'}, g^{uu'})$. Finally, it sets $\kappa \leftarrow \tilde{g}^{u}$. The computed values pass the verification

$e(h', \kappa) = e(s', \tilde{g})$, as we have $e(g^{u'}, \tilde{g}^u) = e(g^{uu'}, \tilde{g})$. Since $d = (x + \sum_{\tau=1}^{q}(m_\tau y_\tau) + r)$ and $d' = \eta r'$ are random values, they match the distribution of u and u', which concludes the fact that

$$\Pr[\mathsf{Game}^4] = \Pr[\mathsf{Game}^3]$$

Game^5: Same as Game^4 except that in Game^5, $\mathcal{S}^5$ simulates the decryption shares $\psi_{s,1}^{\mathsf{sk}_{1,w}}$ and $\psi_{s,1}^{\mathsf{sk}_{2,w}}$ (for ψ_s), and $\psi_{r,1}^{\mathsf{sk}_{1,w}}$ (for ψ_r) of honest maintainer M_w, where $w \in \mathcal{H}$, for the s-th and r-th honest users' threshold encryptions using the values of a non-threshold ElGamal encryption scheme. In this game, the plaintexts of ψ_s and ψ_r are the same as real-world values (in $\mathsf{Game}_{\mathsf{i}}^5$ for $i \geq 1$, we will change the plaintexts to dummy values selected by the simulator $\mathcal{S}_i^5$). To achieve this, $\mathcal{S}^5$ selects the secret decryption keys of the non-threshold ElGamal encryption $\mathsf{sk}_{1,\mathbb{M}}$ and $\mathsf{sk}_{2,\mathbb{M}}$. (Note that from Game^5 onward, the simulator does not use $\mathsf{sk}_{1,\mathbb{M}}$ and $\mathsf{sk}_{2,\mathbb{M}}$ directly, as the decryption shares are simulated using a non-threshold scheme. This allows for reductions to the non-threshold ElGamal IND-CPA security game; see Definition 9, associated with Game^5.) Then, $\mathcal{S}^5$ computes

$$c_s = (\psi_{s,1}, \psi_{s,2}, \psi_{s,3}) = (g^{\rho_s}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_s} \cdot \mathsf{pk}_s, \mathsf{pk}_{2,\mathbb{M}}^{\rho_s} g^v)$$

and

$$c_r = (\psi_{r,1}, \psi_{r,2}) = (g^{\rho_r}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_r} \cdot \mathsf{pk}_r)$$

The plaintexts of threshold encryptions are retrieved as

$$\mathsf{pk}_s = \psi_{s,2} / \prod_{j \in I} \psi_{s,1}^{\mathsf{sk}_{1,j} \lambda_{1,j}}, \quad \mathsf{pk}_r = \psi_{r,2} / \prod_{j \in I} \psi_{r,1}^{\mathsf{sk}_{1,j} \lambda_{1,j}}, \quad g^v = \psi_{s,3} / \prod_{j \in I} \psi_{s,1}^{\mathsf{sk}_{2,j} \lambda_{2,j}}$$

Now, $\mathcal{S}^5$ should simulate the honest maintainers' decryption shares such that the decrypted values become $\mathsf{pk}_s, \mathsf{pk}_r$, and g^v, respectively, which are consistent with $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$ received from the leakage of $\mathcal{F}_{\mathsf{CBDC}}^5$.
$\mathcal{S}^5$ computes $\psi_{s,2}/\mathsf{pk}_s$, which results in $\mathsf{pk}_{1,\mathbb{M}}^{\rho_s} = (g^{\mathsf{sk}_{1,\mathbb{M}}})^{\rho_s} = \psi_{s,1}^{sk_{1,\mathbb{M}}}$, which is used in the computation of the honest maintainers' shares in the following equation. $\mathcal{S}^5$ computes the w-th honest maintainer's decryption share as follows, knowing the malicious maintainers' shares $\psi_{s,1}^{\mathsf{sk}_{1,t}}$ and $\psi_{s,1}^{\mathsf{sk}_{2,t}}$ for ψ_s, and $\psi_{r,1}^{\mathsf{sk}_{1,t}}$ for ψ_r for $\forall t \in \mathcal{C}$ such that $|\mathcal{C}| \leq \beta - 1 = t$. (Note that 0 does not exist in the corrupted maintainers' indexes $\mathcal{C}$, similar to Game^2.)

$$\psi_{s,1}^{\mathsf{sk}_{1,w}} = (\psi_{s,2}/\mathsf{pk}_s)^{\prod_{k \in \mathcal{C}}(k-w)/k} \cdot \prod_{t \in \mathcal{C}} (\psi_{s,1}^{\mathsf{sk}_{1,t}})^{\prod_{k \in \mathcal{C} \cup \{0\}, k \neq t}(w-k)/(t-k)}$$

$\mathcal{S}^5$ computes $\psi_{r,2}/\mathsf{pk}_r$, which results in $\mathsf{pk}_{1,\mathbb{M}}^{\rho_r} = (g^{\mathsf{sk}_{1,\mathbb{M}}})^{\rho_r} = \psi_{r,1}^{sk_{1,\mathbb{M}}}$, then computes $\psi_{r,1}^{\mathsf{sk}_{1,w}}$ as follows:

$$\psi_{r,1}^{\mathsf{sk}_{1,w}} = (\psi_{r,2}/\mathsf{pk}_r)^{\prod_{k \in \mathcal{C}}(k-w)/k} \cdot \prod_{t \in \mathcal{C}} (\psi_{r,1}^{\mathsf{sk}_{1,t}})^{\prod_{k \in \mathcal{C} \cup \{0\}, k \neq t}(w-k)/(t-k)}$$

$\mathcal{S}^5$ computes $\psi_{s,3}/g^v$, which results in $\mathsf{pk}_{2,\mathbb{M}}^{\rho_s} = (g^{\mathsf{sk}_{2,\mathbb{M}}})^{\rho_s} = \psi_{s,1}^{sk_{2,\mathbb{M}}}$, then computes $\psi_{s,1}^{\mathsf{sk}_{2,w}}$ as follows:

$$\psi_{s,1}^{\mathsf{sk}_{2,w}} = (\psi_{s,3}/g^v)^{\prod_{k \in \mathcal{C}}(k-w)/k} \cdot \prod_{t \in \mathcal{C}} (\psi_{s,1}^{\mathsf{sk}_{2,t}})^{\prod_{k \in \mathcal{C} \cup \{0\}, k \neq t}(w-k)/(t-k)}$$

The $\mathcal{F}_{\mathsf{NIZK}}$ emulation allows $\mathcal{S}^5$ to provide fake proofs about the contribution of honest maintainers, which is unconditionally secure. Moreover, changing the honest maintainers' decryption shares is information-theoretically indistinguishable. Additionally, the simulated decryption shares work in the threshold decryption computation (as shown above), thus, we have the following equation:

$$\Pr[\mathsf{Game}^5] = \Pr[\mathsf{Game}^4]$$

Game^6: Same as Game^5 except that in Game^6, we change all plaintexts of threshold encryptions to dummy values selected by $\mathcal{S}^6$. Hence, Game^6 is equivalent to Game^5 except for the

fact that $\mathcal{S}^6$ computes encryptions for some dummy values as plaintexts (e.g., denoted by $\mathsf{pk}_s^*, \mathsf{pk}_r^*$, and g^{*v}) on behalf of an honest user. However, the decryption shares of honest maintainers are simulated in a way that the computation of

$$\psi_{s,2} / \prod_{j \in I} \psi_{s,1}^{\mathsf{sk}_{1,j} \lambda_{1,j}}, \quad \psi_{r,2} / \prod_{j \in I} \psi_{r,1}^{\mathsf{sk}_{1,j} \lambda_{1,j}}, \quad \psi_{s,3} / \prod_{j \in I} \psi_{s,1}^{\mathsf{sk}_{2,j} \lambda_{2,j}}$$

results in $\mathsf{pk}_s, \mathsf{pk}_r$, and g^v, respectively, which are consistent with ($\texttt{AnmRevoked}$, $\mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v$) received from $\mathcal{F}_{\mathsf{CBDC}}^6$ (rather than dummy values $\mathsf{pk}_s^*, \mathsf{pk}_r^*$, and g^{*v}). We omit writing the details as they are similar to Game^5. Hence, any difference between Game^6 and Game^5 is due to breaking the IND-CPA security of the threshold encryption used in our construction, which allows us to bound the probability that $\mathcal{Z}$ distinguishes Game^6 from Game^5 as follows.

We define $\mathsf{Game}_0^5 = \mathsf{Game}^5$ and let p_c be the upper bound on the number of all ciphertexts of honest users. Also, let us define Game_1^5 as a game similar to Game_0^5 except that in Game_1^5, we change the plaintext of the first ciphertext from the real-world value to the ideal-world dummy value. The reduction between Game_0^5 and Game_1^5 is similar to the described reduction below, so that any difference between Game_0^5 and Game_1^5 is upper bounded by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$ (see Definition 9). Similarly, we change the plaintexts of ciphertexts of the i-th honest user to dummy values and finally apply the same process for the last ciphertext of the last honest user such that in $\mathsf{Game}_{\mathsf{p_c}}^5$ (which equals Game^6), all ciphertexts are generated from dummy plaintexts. The reduction between $\mathsf{Game}_{\mathsf{p_c}-1}^5$ and $\mathsf{Game}_{\mathsf{p_c}}^5$ is similar to the reduction described below, so that any difference between $\mathsf{Game}_{\mathsf{p_c}-1}^5$ and $\mathsf{Game}_{\mathsf{p_c}}^5$ is upper bounded by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$.

Associated reduction between $\mathsf{Game}_{\mathsf{i}-1}^5$ and $\mathsf{Game}_{\mathsf{i}}^5$ (IND-CPA Security of Encryption). If $\mathcal{A}$ distinguishes $\mathsf{Game}_{\mathsf{i}-1}^5$ and $\mathsf{Game}_{\mathsf{i}}^5$, it can be used to construct another adversary $\mathcal{A}'$ who breaks the IND-CPA security of the non-threshold ElGamal encryption used in the threshold encryption scheme. The decryption shares of all honest maintainers $\psi_{s,1}^{\mathsf{sk}_{1,w}}$ and $\psi_{s,1}^{\mathsf{sk}_{2,w}}$ for ψ_s, and $\psi_{r,1}^{\mathsf{sk}_{1,w}}$ for ψ_r for $\forall w \in \mathcal{H}$ can be reconstructed from the decryption shares of malicious maintainers, which are $\psi_{s,1}^{\mathsf{sk}_{1,t}}$ and $\psi_{s,1}^{\mathsf{sk}_{2,t}}$ for ψ_s, and $\psi_{r,1}^{\mathsf{sk}_{1,t}}$ for ψ_r for $\forall t \in \mathcal{C}$, and the non-threshold encryption c_s and c_r obtained from the challenger of the IND-CPA security game of non-threshold encryption using Lagrange interpolation for the other shares. We omit writing the details, as the algorithm is similar to the one described in Game^5, except that $\mathcal{A}'$ obtains the non-threshold encryptions c_s and c_r from the challenger of the IND-CPA game. Hence, given the non-threshold ciphertexts, $\mathcal{A}'$ simulates the entire view of $\mathcal{A}$, which consists of the decryption shares contributed by the honest maintainers. This implies that $\mathcal{A}$ cannot distinguish $\mathsf{Game}_{\mathsf{i}-1}^5$ from $\mathsf{Game}_{\mathsf{i}}^5$ unless $\mathcal{A}$ distinguishes non-threshold ciphertexts c_s and c_r generated using real-world values as plaintexts from ciphertexts generated using ideal-world dummy values as plaintexts. In other words, for $\mathcal{Z}$, $\mathsf{Game}_{\mathsf{i}}^5$ is the same as running a threshold encryption scheme with real-world maintainers rather than simulated maintainers by $\mathcal{A}'$. Hence, if $\mathcal{Z}$ distinguishes $\mathsf{Game}_{\mathsf{i}-1}^5$ from $\mathsf{Game}_{\mathsf{i}}^5$, $\mathcal{A}'$ uses this to win the IND-CPA security game of the non-threshold encryption scheme. As a result, threshold encryption is simulatable, and together with the IND-CPA property of the non-threshold encryption scheme, this ensures that threshold encryption is IND-CPA secure in our construction.

Therefore, under the IND-CPA property of the non-threshold ElGamal encryption scheme, the following inequality holds:

$$\left|\Pr[\mathsf{Game}^6] - \Pr[\mathsf{Game}^5]\right| \leq p_c \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$$

Game^7: Same as Game^6, except that for the honest maintainer M_w, the simulator $\mathcal{S}^7$ computes the tracing tag share $\dot{g}^{a_w}$ for tracing the honest user U without directly knowing the shares a_w of the tracing key. Here, $\dot{g}$ is a group element computed in each step of the protocol. In this game, tracing tags are the same as real-world values (as we will see in $\mathsf{Game}_{\mathsf{i}}^7$ for $i \geq 1$, we will change these tags to dummy values selected by the simulator $\mathcal{S}_i^7$). $\mathcal{S}^7$ knows $\{g^{z_\tau}\}_{\tau=1}^{x}$ from the transaction identifiers leaked from $\mathcal{F}_{\mathsf{CBDC}}^7$, which are ($\texttt{Traced}, \mathsf{sid}, \mathsf{U}, \{t_{\mathsf{id}}^\tau, \mathsf{role}^\tau\}_{\tau=1}^{x}$),

and computes M_w's first share as follows ($\dot{g} = g$):

$$g^{a_w} = (g^{z_1})^{\prod_{k\in\mathcal{C},k\neq 0}(k-w)/k} \cdot \prod_{t\in\mathcal{C}} (g^{a_t})^{\prod_{k\in\mathcal{C}\cup\{0\},k\neq t}(w-k)/(t-k)}$$

Then, given the revealed $(\tau-1)$-th tracing tag ($g^{z_{\tau-1}}$), the w-th honest maintainer's share for the next computation is simulated as follows ($\dot{g} = g^{z_{\tau-1}}$):

$$(g^{z_{\tau-1}})^{a_w} = (g^{z_\tau})^{\prod_{k\in\mathcal{C},k\neq 0}(k-w)/k} \cdot \prod_{t\in\mathcal{C}} ((g^{z_{\tau-1}})^{a_t})^{\prod_{k\in\mathcal{C}\cup\{0\},k\neq t}(w-k)/(t-k)}$$

Changing the honest maintainers' shares is information-theoretically indistinguishable, and emulating $\mathcal{F}_{\mathsf{NIZK}}$ (which is unconditionally secure) allows $\mathcal{S}^7$ to provide faked proofs. As a result, we have

$$\Pr[\mathsf{Game}^7] = \Pr[\mathsf{Game}^6]$$

Game^8: Same as Game^7, except that in Game^8, we change the tracing tags of honest users to dummy values selected by $\mathcal{S}^8$. Hence, Game^8 is equivalent to Game^7 except for the fact that $\mathcal{S}^8$ computes tracing tags and submits them to $\mathcal{F}^8_{\mathsf{CBDC}}$ as part of transaction identifiers in the *currency issuance* and *payment* protocols. However, the tracing tag shares of honest maintainers are simulated in such a way that the computation of tags results in $\{g^{z_\tau}\}_{\tau=1}^{x}$, which are consistent with the transaction identifiers t_{id} leaked from $\mathcal{F}^8_{\mathsf{CBDC}}$. We omit writing the details as they are similar to Game^7. Hence, any difference between Game^8 and Game^7 is due to distinguishing g^{a^x} values from g^z values, which allows us to bound the probability that $\mathcal{Z}$ distinguishes Game^8 from Game^7 as follows.
We define $\mathsf{Game}^7_0 = \mathsf{Game}^7$ and let p_u be the upper bound on the number of all honest users. Also, let us define Game^7_1 as a game similar to Game^7_0, except that in Game^7_1, we change the tracing tags of the first honest user from real-world values to ideal-world dummy values. The reduction between Game^7_0 and Game^7_1 is described below, so that any difference between Game^7_0 and Game^7_1 is upper bounded by $\mathsf{Adv}^{d\text{-sDDH}}_{\mathcal{A}}$ (see Definition 6). Similarly, we change the tracing tags of the i-th honest user to dummy values, and finally, we apply the same process for the last honest user such that in $\mathsf{Game}^7_{\mathsf{p_u}}$ (which is equivalent to Game^8), all tracing tags are dummy values. The reduction between $\mathsf{Game}^7_{\mathsf{p_u}-1}$ and $\mathsf{Game}^7_{\mathsf{p_u}}$ is similar to the described reduction below, so that any difference between $\mathsf{Game}^7_{\mathsf{p_u}-1}$ and $\mathsf{Game}^7_{\mathsf{p_u}}$ is upper bounded by $\mathsf{Adv}^{d\text{-sDDH}}_{\mathcal{A}}$.
Associated reduction between $\mathsf{Game}^7_{\mathsf{i}-1}$ and $\mathsf{Game}^7_{\mathsf{i}}$ (Hardness of d-sDDH). If $\mathcal{A}$ distinguishes $\mathsf{Game}^7_{\mathsf{i}-1}$ from $\mathsf{Game}^7_{\mathsf{i}}$, it can be used to construct another adversary $\mathcal{A}'$ who breaks the hardness of the d-strong Diffie-Hellman problem. The tracing tag shares of all honest maintainers $\dot{g}^{a_w}$ for $\forall w \in \mathcal{H}$ can be reconstructed from the tracing tag shares of malicious maintainers, which are $\dot{g}^{a_t}$ for $\forall t \in \mathcal{C}$, and the tracing tags $\{g^{z_\tau}\}_{\tau=1}^{x}$ received from the leakage of functionality, using Lagrange interpolation for the other shares. We omit writing the details, as the algorithm is similar to the one described in Game^7. Hence, $\mathcal{A}'$ simulates the entire view of $\mathcal{A}$, which consists of the tracing tag computation shares contributed by the honest maintainers. This implies that $\mathcal{A}$ cannot distinguish $\mathsf{Game}^7_{\mathsf{i}-1}$ from $\mathsf{Game}^7_{\mathsf{i}}$ unless $\mathcal{A}$ breaks the hardness of the d-strong Diffie-Hellman problem. Thus, if $\mathcal{Z}$ distinguishes $\mathsf{Game}^7_{\mathsf{i}-1}$ from $\mathsf{Game}^7_{\mathsf{i}}$, $\mathcal{A}'$ uses this to win the indistinguishability game of the d-strong Diffie-Hellman problem. As a result, under the hardness of the d-strong Diffie-Hellman problem, the following inequality holds:

$$\left|\Pr[\mathsf{Game}^8] - \Pr[\mathsf{Game}^7]\right| \leq p_u \cdot \mathsf{Adv}^{d\text{-sDDH}}_{\mathcal{A}}$$

As $\mathcal{F}^8_{\mathsf{CBDC}} = \mathcal{F}_{\mathsf{CBDC}}$ and $\mathcal{S}^8 = \mathcal{S}$, which means that Game^8 corresponds to the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$, we argue that the random variables $\mathsf{EXEC}_{\Pi_{\mathsf{PEReDi}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ are statistically close. In other words, the probability for any PPT environment $\mathcal{Z}$ to distinguish $\mathsf{EXEC}_{\Pi_{\mathsf{PEReDi}},\mathcal{A},\mathcal{Z}}$ from $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ is upper bounded by

$$p_c \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{EU\text{-}CMA}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}} + p_u \cdot \mathsf{Adv}^{d\text{-sDDH}}_{\mathcal{A}}$$

which, together with the simulator description, concludes the security proof.

5.2 Simulation

We describe a simulator $\mathcal{S}$ that reproduces the real-world view of $\mathcal{A}$ and emulates the execution of honest parties. The simulator internally emulates the functionalities $\mathcal{F}_{\mathsf{KR}}, \mathcal{F}_{\mathsf{Ch}}, \mathcal{F}_{\mathsf{aBA}}, \mathcal{F}_{\mathsf{RO}}, \mathcal{F}_{\mathsf{BC}}, \mathcal{F}_{\mathsf{NIZK}}$, and $\mathcal{F}_{\mathsf{SoK}}$. To achieve this, it maintains specific lists associated with each functionality. However, without loss of generality, we assume that $\mathcal{S}$ internally tracks the states of functionalities and omit explicit discussion of these state management processes. $\mathcal{S}$ interacts with the dummy adversary $\mathcal{A}$ and the CBDC functionality $\mathcal{F}_{\mathsf{CBDC}}$. Similar to the functionality $\mathcal{F}_{\mathsf{CBDC}}$ and our construction Π_{PEReDi}, the simulator is described in six parts: *user registration, currency issuance, payment, abort transaction, privacy revocation*, and *tracing*. Following the notations described in Section 2.1, we use the subscript w for an honest maintainer (e.g., σ_w denotes a signature generated by an honest maintainer). Similarly, we use the subscript t for a malicious maintainer.

Remark 1. In the following, we assume that the simulator $\mathcal{S}$, whenever emulating an honest entity, follows the associated real-world algorithms provided in Section 4.2 without explicitly mentioning all the details (e.g., updates local ledgers emulating honest maintainers). However, whenever the simulator needs to deviate from the real-world algorithms (e.g., by simulating a cryptographic object using information leaked via the ideal functionality $\mathcal{F}_{\mathsf{CBDC}}$, or extracting NIZK witness from the adversary's proof), we explicitly state it.

Remark 2. In any protocol where there is communication between a malicious user and maintainers, the adversary controlling the communication channel may, for example, choose to block t honest maintainers and, on behalf of t malicious maintainers, refrain from transferring blind signature shares via the ideal channel functionality emulated by the simulator. In this case, the simulator does not know whether the malicious user's account state transition has been finalized in that protocol, as $4t + 1$ valid shares are required for state transition. However, since the simulator always extracts the witness whenever a transaction including a NIZK proof is submitted by the adversary, it is aware of the account state of malicious users. Therefore, the simulator checks its own state to determine whether the adversary has decided to finalize any previous account state transitions (for malicious users). If the account state has advanced, the simulator invokes $\mathcal{F}_{\mathsf{CBDC}}$ with the appropriate message to finalize the previous protocol. Henceforth, we omit explicit mention of this process for clarity and conciseness.

The following is the simulator $\mathcal{S}$ description.

Initialization. Emulating $\mathcal{F}_{\mathsf{KR}}$, internally update the state of $\mathcal{F}_{\mathsf{KR}}$ for all honest users and maintainers. Also, upon receiving calls from the adversary (on behalf of malicious entities), follow the functionality description of $\mathcal{F}_{\mathsf{KR}}$ and update its state accordingly. Provide all the associated leakages of $\mathcal{F}_{\mathsf{KR}}$ to $\mathcal{A}$ accordingly.

User registration. We note that in order to keep the functionality as simple as possible we leave it to the adversary to determine the outcome of the KYC process in the ideal world. Our construction on the other hand does capture it.

(i) **Honest U and at most t malicious maintainers:**

- Upon receiving $(\mathtt{GenAcc}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{F}_{\mathsf{CBDC}}$, initiate the honest user emulation, for the user with the leaked unique identifier U, as follows.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, output $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$ where $\mathtt{x} = (\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}})$ is computed following the real-world algorithm provided in Figure 3.
- Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$ update the state of $\mathcal{F}_{\mathsf{NIZK}}$ and proceed as follows.
- Emulating $\mathcal{F}_{\mathsf{BC}}$, send $(\mathtt{Broadcasted}, \mathsf{sid}, \mathsf{U}, \mathsf{com}_{\mathbb{M}})$ to $\mathcal{A}$ (both as the leakage of $\mathcal{F}_{\mathsf{BC}}$ and the message malicious maintainers receive).
- Set $\mathsf{RI}_j \leftarrow (\mathsf{acc}^{\mathfrak{B}}, a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}, \pi), \quad \forall j \in \{1, \ldots, D\}$.

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$ and the honest user provide all the channel leakages to $\mathcal{A}$.
- Send $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}, \mathsf{RI}_t)$ to $\mathcal{A}$ (for malicious maintainers) depending on $\mathcal{A}$'s choice on $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$'s message delivery.
- Emulating $\mathcal{F}_{\mathsf{KR}}$ provide all the leakages of the form $(\mathtt{RetrieveKey}, \mathsf{sid}, \mathsf{U}, \mathsf{M})$ to $\mathcal{A}$ (both as honest maintainers emulation and whenever the adversary on behalf of malicious maintainers calls $\mathcal{F}_{\mathsf{KR}}$ with $(\mathtt{RetrieveKey}, \mathsf{sid}, \mathsf{U})$).
- Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \mathsf{U}, \mathsf{M})$ from $\mathcal{A}$, submit the information malicious maintainers receive from $\mathcal{F}_{\mathsf{KR}}$ to $\mathcal{A}$: $(\mathtt{KeyRetrieved}, \mathsf{sid}, \mathsf{U}, \mathsf{pk}_{\mathsf{U}})$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, whenever $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ for $\mathtt{x} = (\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}})$ output $(\mathtt{Verification}, \mathsf{sid}, 1)$.[13]
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, provide the channel leakages of the form $(\mathtt{Send}, \mathsf{sid}, (\mathsf{M}_j, \mathsf{U}, \sigma_j^{\mathfrak{B}}), \mathsf{mid})$ to $\mathcal{A}$ on behalf of honest maintainers and malicious ones who call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, \sigma_t^{\mathfrak{B}})$. For honest maintainers (e.g., M_w) generate $\sigma_w^{\mathfrak{B}}$ following the algorithm provided in Figure 3.
- Depending on $\mathcal{A}$'s choice on message delivery in $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, let $\mathcal{F}_{\mathsf{CBDC}}$ deliver $(\mathtt{AccGened}, \mathsf{sid}, \mathsf{U})$ to maintainers.
- Submit $(\mathtt{Ok.GenAcc}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{F}_{\mathsf{CBDC}}$.
- Depending on $\mathcal{A}$'s choice on $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ blockage, let $\mathcal{F}_{\mathsf{CBDC}}$ output $(\mathtt{AccGened}, \mathsf{sid})$ to U.

(ii) **Malicious U and at most t malicious maintainers:** (We avoid addressing simulations that are similar to honest U case described above.)

- Emulating $\mathcal{F}_{\mathsf{BC}}$, once adversary $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{BC}}$ with $(\mathtt{Broadcast}, \mathsf{sid}, \mathsf{com}_{\mathbb{M}})$, submit $(\mathtt{GenAcc}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of malicious U.[14]
- Upon receiving $(\mathtt{GenAcc}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{F}_{\mathsf{CBDC}}$ proceed as follows.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$, receive RI_j values submitted by the adversary.
- Emulating honest maintainers parse $\mathsf{RI}_j \leftarrow (\mathsf{acc}^{\mathfrak{B}}, a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}, \pi), \quad \forall j \in \{1, \ldots, D\}$, and execute the algorithm described in Figure 4.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$ and honest maintainers, submit $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$ where $\mathtt{x} = (\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}})$, $\mathtt{x} \in \mathsf{RI}_w$ and $\pi \in \mathsf{RI}_w$ for RI_w values received from $\mathcal{A}$.
- Upon receiving $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, check if $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ holds. If holds, store $(\mathtt{x}, \pi)$. Else, ignore.

Currency issuance.

(i) **Honest U, honest B and at most t malicious maintainers:**

- Upon receiving $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$, initiate *currency issuance* protocol by emulating honest B.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sra}}$, provide the leakage of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sra}}$ to $\mathcal{A}$ where $|v|$ in the leakage is for a random v.
- Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$, emulate an honest user.[15]
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ssa}}$, submit the leakage of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ssa}}$ to $\mathcal{A}$ for a random ρ.
- Based on the algorithm provided in Figure 7 compute threshold ElGamal encryption ψ on dummy values as plaintexts, and sample random values for $\mathsf{acc}^{\mathsf{old}}$. Select a random value as v.
- Emulating honest maintainers, generate $\sigma_{\mathbb{M}}$ for $\mathsf{acc}^{\mathsf{old}}$.
- Given $\mathsf{acc}^{\mathsf{old}}, v$ and $\sigma_{\mathbb{M}}$, compute $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ following TBS algorithms.
- Sample $z \xleftarrow{\$} \mathbb{Z}_p$, and set $\mathsf{T} \leftarrow g^z$.

[13] Since $\mathcal{S}$ (emulating $\mathcal{F}_{\mathsf{NIZK}}$) has already recorded $(\mathtt{x}, \pi)$, we will no longer mention the $\mathtt{Verify}$ command of $\mathcal{F}_{\mathsf{NIZK}}$ whenever the adversary invokes it in our simulator description across all protocols.

[14] Emulating $\mathcal{F}_{\mathsf{BC}}$, $\mathcal{S}$ knows U.

[15] Note that if the user has already been traced $\mathcal{S}$ receives $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U})$ from $\mathcal{F}_{\mathsf{CBDC}}$ so that it is able to use the same tag in this protocol as it had generated for the user U who did not have any transactions in the time of executing the *tracing* protocol.

- Set $\mathtt{x} \leftarrow (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T})$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$. Upon receiving back $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, record $(\mathtt{x}, \pi)$ and proceed.
- Set $\mathsf{TI}_{\mathsf{U}} \leftarrow (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$, and $\mathsf{TI}_{\mathsf{B}} \leftarrow \psi$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ and an honest user provide all the channel leakages to $\mathcal{A}$.
- Send $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid})$ to $\mathcal{A}$ (for malicious maintainers) depending on $\mathcal{A}$'s choice on $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$'s message delivery.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ and the honest bank B provide all the channel leakages to $\mathcal{A}$.
- Send $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, \mathsf{TI}_{\mathsf{B}})$ to $\mathcal{A}$ (for malicious maintainers) depending on $\mathcal{A}$'s choice on $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$'s message delivery.
- Set $t_{\mathsf{id}} \leftarrow (\psi, \mathsf{T})$, and submit $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ to $\mathcal{F}_{\mathsf{CBDC}}$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, provide the channel leakages of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{new},\mathfrak{B}}, \mathsf{mid})$ to $\mathcal{A}$ on behalf of honest maintainers and malicious ones who call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}, \sigma_t^{\mathsf{new},\mathfrak{B}})$. For honest maintainers (e.g., M_w) generate $\sigma_w^{\mathsf{new},\mathfrak{B}}$ following the algorithm provided in Figure 9.
- Depending on $\mathcal{A}$'s choice on message delivery in $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ and $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ for maintainer M_k, submit $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ to $\mathcal{F}_{\mathsf{CBDC}}$.

(ii) **Malicious U, honest B and at most t malicious maintainers:** (We avoid addressing simulations that are similar to honest U and honest B case described above.)

- Upon receiving $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}, v)$ from $\mathcal{F}_{\mathsf{CBDC}}$, start emulating the honest B by initiating the *currency issuance* protocol to issue a digital currency worth of v for the adversary $\mathcal{A}$ (malicious user U).
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sra}}$, submit $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, v)$ to $\mathcal{A}$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ssa}}$, receive the message of $\mathcal{A}$ $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \rho)$ and leak the leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ssa}}$.
- Emulating $\mathcal{F}_{\mathsf{KR}}$, retrieve pk_{U} and compute $\psi \leftarrow (g^{\rho}, \mathsf{pk}_{1,\mathbb{M}}^{\rho} \cdot \mathsf{pk}_{\mathsf{U}}, \mathsf{pk}_{2,\mathbb{M}}^{\rho} \cdot g^{v})$.
- Upon receiving, the call of $\mathcal{A}$ to $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{U}})$ provide the necessary leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ to $\mathcal{A}$ and deliver associated messages to $\mathcal{A}$ (on behalf of malicious maintainers).
- Parse $\mathsf{TI}_{\mathsf{U}} = (\psi^*, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$ and ignore if $\psi^* \neq \psi$.
- Else, submit $(\mathtt{AcceptIss}, \mathsf{sid}, v)$ on behalf of malicious user U to $\mathcal{F}_{\mathsf{CBDC}}$ and proceed upon receiving back $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid})$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$ and honest maintainers, submit $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$ where $\mathtt{x} = (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T})$, $\mathtt{x} \in \mathsf{TI}_{\mathsf{U}}$ and $\pi \in \mathsf{TI}_{\mathsf{U}}$ received from $\mathcal{A}$.
- Upon receiving $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, check if $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ holds. If holds, store $(\mathtt{x}, \pi)$ and set $t_{\mathsf{id}} \leftarrow (\psi, \mathsf{T})$ for $\mathsf{T} \in \mathsf{TI}_{\mathsf{U}}$. Else, ignore.[16]
- Depending on $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ blockage by $\mathcal{A}$, send $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{new},\mathfrak{B}})$ to $\mathcal{A}$ (malicious U).

(iii) **Honest U, malicious B and at most t malicious maintainers:** (We avoid addressing simulations that are similar to previous cases e.g., in honest U and honest B case described above.)

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sra}}$, receive $\mathcal{A}$'s call of the form: $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, v)$.
- Submit $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ to $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of malicious B.
- Upon receiving back $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$, proceed as follows.
- Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$, start emulating the honest user.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, receive $\mathcal{A}$'s call of the form: $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}})$.
- Parse $\mathsf{TI}_{\mathsf{B}} = \psi^*$ and ignore if $\psi^* \neq \psi$ (where ψ is generated by $\mathcal{S}$ using randomness ρ).

(iv) **Malicious U, malicious B and at most t malicious maintainers:**

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, receive $\mathcal{A}$'s call of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{U}})$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, receive $\mathcal{A}$'s call of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}'_j, \mathsf{TI}_{\mathsf{B}})$.

[16] Recall Remark 1 that we do not explicitly mention the steps of the real-world algorithm executed by an honest maintainer simulated by $\mathcal{S}$, e.g., verifying the signature by running the $\mathsf{VerifySig}$ algorithm of the TBS scheme.

- Ignore if $\psi \neq \psi^*$ for $\psi \in \mathsf{TI}_\mathsf{B}$ equals to $\psi^* \in \mathsf{TI}_\mathsf{U}$.
- Else, extract the witness of NIZK proof $\pi \in \mathsf{TI}_\mathsf{U}$ similar to the malicious U and honest B case described above and ignore if NIZK relation does not hold.

Payment.

(i) **Honest U_s, honest U_r and at most t malicious maintainers:**

- Upon receiving $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$, initiate the *payment* protocol by emulating an honest sender U_s.[17]
- Choose a random value as v.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ and an honest U_s, provide the associated leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$, emulate an honest receiver U_r, and provide associated leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ to $\mathcal{A}$.[18]
- Based on the algorithms provided in Figure 12 and Figure 13 compute threshold ElGamal encryptions ψ_s and ψ_r on dummy values as plaintexts, and sample random values for $\mathsf{acc}_s^{\mathsf{old}}$ and $\mathsf{acc}_r^{\mathsf{old}}$.
- Emulating honest maintainers, generate $\sigma_{s,\mathbb{M}}$ and $\sigma_{r,\mathbb{M}}$ for $\mathsf{acc}_s^{\mathsf{old}}$ and $\mathsf{acc}_r^{\mathsf{old}}$ respectively.
- Given $\mathsf{acc}_s^{\mathsf{old}}, \mathsf{acc}_r^{\mathsf{old}}, v, \sigma_{s,\mathbb{M}}$, and $\sigma_{r,\mathbb{M}}$ compute $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}$ and $\sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}$ following TBS algorithms.
- Sample $z_s \xleftarrow{\$} \mathbb{Z}_p$ and $z_r \xleftarrow{\$} \mathbb{Z}_p$. Set $\mathsf{T}_s \leftarrow g^{z_s}$ and $\mathsf{T}_r \leftarrow g^{z_r}$.
- Set $\mathtt{x}_s \leftarrow (\psi_s, \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s)$ and $\mathtt{x}_r \leftarrow (\psi_r, \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}, \sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_r)$.
- Emulating $\mathcal{F}_{\mathsf{SoK}}$, compute $\overline{\sigma_s}(\psi_r) \leftarrow \mathsf{Simsign}(\psi_r, \mathtt{x}_s)$ and $\overline{\sigma_r}(\psi_s) \leftarrow \mathsf{Simsign}(\psi_s, \mathtt{x}_r)$.
- Emulating $\mathcal{F}_{\mathsf{SoK}}$, record the entries $(\psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$, and $(\psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s))$.
- Set $\mathsf{TI}_s \leftarrow (\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s)$ and $\mathsf{TI}_r \leftarrow (\psi_s, \psi_r, \overline{\sigma_r}(\psi_s), \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}, \sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_r)$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ provide all the associated leakages to $\mathcal{A}$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, send $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_s, \mathsf{mid}_s)$, and $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_r, \mathsf{mid}_r)$ to $\mathcal{A}$ (for malicious maintainers) depending on $\mathcal{A}$'s choice on $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$'s message delivery.[19]
- Set $t_{\mathsf{id}} \leftarrow (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$, and record t_{id}.
- Submit $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ to $\mathcal{F}_{\mathsf{CBDC}}$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, provide the channel leakages of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}_s)$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}'_j, \sigma_{r,j}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}_r)$ to $\mathcal{A}$ on behalf of honest maintainers and malicious ones who call $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_s, \sigma_{s,t}^{\mathsf{new},\mathfrak{B}})$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_r, \sigma_{r,t}^{\mathsf{new},\mathfrak{B}})$ respectively. Emulating honest maintainers (e.g., M_w) generate $\sigma_{s,w}^{\mathsf{new},\mathfrak{B}}$ and $\sigma_{r,w}^{\mathsf{new},\mathfrak{B}}$ following the algorithm provided in Figure 15.
- Depending on $\mathcal{A}$'s choice on message delivery in $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ for maintainer M_k, submit $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ to $\mathcal{F}_{\mathsf{CBDC}}$.

(ii) **Malicious U_s, honest U_r and at most t malicious maintainers:** (We avoid addressing simulations that are similar to honest U_s and honest U_r case described above.)

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$, receive $\mathcal{A}$'s call to $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}_r, (\rho_s, v))$. Send the leakage of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ to $\mathcal{A}$.
- Call $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of malicious U_s with $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ and proceed as follows upon receiving back $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid})$ (or $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s)$).

[17] Note that if the sender U_s has already been traced, $\mathcal{S}$ receives $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s)$ from $\mathcal{F}_{\mathsf{CBDC}}$ so that it is able to use the same tag in this protocol as it had generated for U_s who did not have any transactions in the time of executing the *tracing* protocol.

[18] Similar to the case where U_s is traced, if the receiver U_r has already been traced, $\mathcal{S}$ receives $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_r)$ from $\mathcal{F}_{\mathsf{CBDC}}$.

[19] Emulating $\mathcal{F}_{\mathsf{SoK}}$, whenever $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{SoK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$ and $(\mathtt{Verify}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s))$ (where all the values are simulated by $\mathcal{S}$) output $(\mathtt{Verified}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r), 1)$ and $(\mathtt{Verified}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s), 1)$ respectively to $\mathcal{A}$.

- Upon receiving $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid})$ (or $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_r)$) from $\mathcal{F}_{\mathsf{CBDC}}$ start emulating the honest receiver U_r.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ and U_r, send $(\texttt{Received}, \mathsf{sid}, \mathsf{U}_r, \rho_r)$ to $\mathcal{A}$ for a randomly chosen ρ_r.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, receive $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_s)$ from $\mathcal{A}$ once it calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ and act accordingly (as described for the previous case).
- Emulating $\mathcal{F}_{\mathsf{KR}}$, retrieve pk_s and pk_r.
- Compute $\psi_s = (g^{\rho_s}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_s} \cdot \mathsf{pk}_s, \mathsf{pk}_{2,\mathbb{M}}^{\rho_s} \cdot g^v)$ and $\psi_r = (g^{\rho_r}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_r} \cdot \mathsf{pk}_r)$.
- Parse $\mathsf{TI}_s = (\psi_s^*, \psi_r^*, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s)$. Ignore if $\psi_s^* \neq \psi_s$ or $\psi_r^* \neq \psi_s$.
- Emulating $\mathcal{F}_{\mathsf{SoK}}$, let $\mathtt{w}_s \leftarrow \mathsf{Extract}(\psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$. If $(\mathtt{x}_s, \mathtt{w}_s) \in \mathcal{R}$ holds proceed as follows. Else ignore.
- Set $t_{\mathsf{id}} \leftarrow (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ where T_r is computed in a similar way as described in honest U_s and honest U_r case above.

(iii) **Honest U_s, malicious U_r and at most t malicious maintainers:** (We avoid addressing simulations that are similar to malicious U_s and honest U_r case described above.)

- Upon receiving $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s, \mathsf{U}_r, v)$ from $\mathcal{F}_{\mathsf{CBDC}}$, start emulating the honest U_s, who initiates the *payment* protocol.
- Send $(\texttt{Received}, \mathsf{sid}, \mathsf{U}_s, \rho_s, v)$ to $\mathcal{A}$ (malicious U_r) for a randomly chosen ρ_s.
  (this case is similar to malicious U_s and honest U_r case described above, however, e.g., witness extraction is for TI_r.)

(iv) **Malicious U_s, malicious U_r, and at most t malicious maintainers:** (We avoid addressing simulations that are similar to previous cases.)

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, receive $\mathcal{A}$'s calls to $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ with the messages $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_s)$ and $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_r)$.
- Emulating honest maintainers execute the algorithm described in Figure 15, however, emulating $\mathcal{F}_{\mathsf{SoK}}$, extract the witness for both TI_s and TI_r (e.g., similar to malicious U_s and honest U_r case described above.). If the relation holds submit $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ and $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{U}_s, v)$ on behalf of malicious U_s and U_r to $\mathcal{F}_{\mathsf{CBDC}}$.

Abort transaction.
(i) **Honest U, and at most t malicious maintainers:**

- Upon receiving $(\texttt{AbrTnx}, \mathsf{sid}, t_{\mathsf{id}})$ from $\mathcal{F}_{\mathsf{CBDC}}$, start emulating an honest user.
- Sample random values for $\mathsf{acc}^{\mathsf{old}}$.
- Emulating honest maintainers, generate $\sigma_{\mathbb{M}}$ for $\mathsf{acc}^{\mathsf{old}}$.
- Given $\mathsf{acc}^{\mathsf{old}}$ and $\sigma_{\mathbb{M}}$, compute $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ following TBS algorithms.
- Retrieve the marked tag leaked from $\mathcal{F}_{\mathsf{CBDC}}$: $\mathsf{T} \in t_{\mathsf{id}}$ (see abort transaction leakage in $\mathcal{F}_{\mathsf{CBDC}}$).
- Set $\mathtt{x} \leftarrow (\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T})$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\texttt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$. Upon receiving back $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, record $(\mathtt{x}, \pi)$ and proceed.
- Set $\mathsf{AR} \leftarrow (\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ provide all the associated leakages to $\mathcal{A}$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, send $(\texttt{Received}, \mathsf{sid}, \mathsf{AR}, \mathsf{mid})$ to $\mathcal{A}$ (for malicious maintainers) depending on $\mathcal{A}$'s choice on $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$'s message delivery.
- Following the algorithm described in Figure 18, leak the leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ to $\mathcal{A}$ (for both honest maintainers being emulated and malicious maintainers on behalf of whom $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ with $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_i, (\mathsf{TI}_s, \mathsf{TI}_r, \mathsf{mid}_s, \mathsf{mid}_r))$), and submit $(\texttt{AbrTnx.Ok}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{F}_{\mathsf{CBDC}}$.
- Following the algorithm described in Figure 19 when emulating honest maintainers for messages sent by malicious maintainers, act as follows.
  - For $(j = s \wedge i = r)$ and $(j = r \wedge i = s)$:

  - ∗ check whether there exists some σ' such that $(\psi_i, \mathtt{x}_j, \sigma')$ where $\mathtt{x}_j = (\psi_j, \mathsf{acc}_j^{\mathsf{new},\mathfrak{B}}, \sigma_{j,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_j)$ is stored.
  - ∗ if not, emulate $\mathcal{F}_{\mathsf{SoK}}$ and compute: $\mathtt{w}_j \leftarrow \mathsf{Extract}(\psi_i, \mathtt{x}_j, \overline{\sigma_j}(\psi_i))$.
  - ∗ check whether the relation holds: $(\mathtt{x}_j, \mathtt{w}_j) \in \mathcal{R}$. If it does not hold, ignore. Otherwise, proceed as follows.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, leak all associated leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ to $\mathcal{A}$ based on the algorithm in Figure 19. Deliver messages to $\mathcal{A}$ (malicious maintainers) depending on $\mathcal{A}$'s choice for message blockage.
- Emulating $\mathcal{F}_{\mathsf{aBA}}$, leak the associated leakages of $\mathcal{F}_{\mathsf{aBA}}$ to $\mathcal{A}$ for honest maintainers (following the algorithm described in Figure 20) and malicious ones who call $\mathcal{F}_{\mathsf{aBA}}$ with $(\mathtt{Agree}, \mathsf{sid} \mid\mid [(\mathsf{TI}_s, \mathsf{TI}_r), (\mathsf{mid}_s, \mathsf{mid}_r)], d_j)$.
- Emulating $\mathcal{F}_{\mathsf{aBA}}$, based on $\mathcal{A}$'s actions, terminate the Byzantine agreement upon receiving the call from $4t+1$ maintainers (including honest ones that are being emulated). Set Q to the bit that has the majority among the $4t+1$ (e.g., in the worst-case scenario, messages are split into two $2t$ groups, and the $4t+1$-th message will terminate the agreement). *(Note that if a transaction is already finalized (i.e., there are at least* $4t+1$ *valid maintainer signature shares for it), the adversary can never prevent the* abort transaction *protocol from finalizing the transaction. Even if the adversary blocks* t *honest maintainers and provides incorrect input on behalf of* t *malicious ones, there are still* $2t+1$ *(*$= 4t+1-t-t$*) honest maintainers emulated by* $\mathcal{S}$ *who can safely finalize the transaction and unstuck the user similar to the real-world protocol.)*
- If $Q = 1$, follow the algorithm described in Figure 20 and leak the $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$'s leakages to $\mathcal{A}$ accordingly. Let $\mathcal{F}_{\mathsf{CBDC}}$ send $(\mathtt{TnxAborted}, \mathsf{sid})$ and $(\mathtt{TnxAborted}, \mathsf{sid}, t_{\mathsf{id}})$ (to associated maintainers), then terminate.
- If $Q = 0$, emulating $\mathcal{F}_{\mathsf{aBA}}$, leak the associated leakages of $\mathcal{F}_{\mathsf{aBA}}$ to $\mathcal{A}$ for honest maintainers (following algorithm described in Figure 20) and malicious ones who call $\mathcal{F}_{\mathsf{aBA}}$ with $(\mathtt{Agree}, \mathsf{sid} \mid\mid [\mathsf{AR.mid}], d_j)$. Follow the algorithm described in Figure 20 and leak the $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$'s leakages to $\mathcal{A}$ accordingly. Let $\mathcal{F}_{\mathsf{CBDC}}$ send $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$, $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$, and $(\mathtt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ (to associated maintainers), then terminate.

(ii) **Malicious** U**, and at most** t **malicious maintainers:** (We avoid addressing simulations that are similar to the honest U case described above.)

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$, upon receiving messages of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{AR})$ from $\mathcal{A}$, leak associated leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ to $\mathcal{A}$.
- Parse $\mathsf{AR} = (\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$ and honest maintainers, submit $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$ where $\mathtt{x} = (\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T})$.
- Upon receiving $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, check if $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ holds. If holds, store $(\mathtt{x}, \pi)$ and submit $(\mathtt{AbrTnx}, \mathsf{sid})$ on behalf of malicious U to $\mathcal{F}_{\mathsf{CBDC}}$. Else, ignore.
- Upon receiving $(\mathtt{AbrTnx}, \mathsf{sid}, t_{\mathsf{id}})$ from $\mathcal{F}_{\mathsf{CBDC}}$, retrieve the marked tag leaked from $\mathcal{F}_{\mathsf{CBDC}}$: $\mathsf{T}^* \in t_{\mathsf{id}}$ (see abort transaction leakage in $\mathcal{F}_{\mathsf{CBDC}}$).
- Ignore if $\mathsf{T}^* \neq \mathsf{T}$. Else, proceed similar to the honest U case described above.

Privacy revocation.
(i) **Honest** U_s**, honest** U_r**, and at most** t **malicious maintainers for both *currency issuance* and *payment* protocols:**[20]

- Upon receiving $(\mathtt{RvkAnm}, \mathsf{sid}, t_{\mathsf{id}}^w, \mathsf{M}_w)$ from $\mathcal{F}_{\mathsf{CBDC}}$, start emulating honest maintainers (recall w and t are used for information associated to honest and malicious maintainers respectively).
- Emulating honest maintainers if there exists at least one t_{id}^w recorded, submit $(\mathtt{RvkAnm.Ok}, \mathsf{sid}, t_{\mathsf{id}}^w)$ to $\mathcal{F}_{\mathsf{CBDC}}$.

[20] In the following, for simplicity we describe $\mathcal{S}$ for payment transactions (issuance transactions are similar and more straightforward).

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, receive $\mathcal{A}$'s call (e.g., on behalf of M_t) of the form: $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_i, (\mathtt{x}_t, \pi_t))$ and submit leakages accordingly to $\mathcal{A}$.
- Parse $\mathtt{x}_t = (\psi_{s,1}, \psi_{r,1}, A_t, B_t, C_t)$.
- Emulating honest maintainers, check if there exists recorded t^*_{id} with ψ_s and ψ_r where $\psi_s = (\psi_{s,1}, \cdot, \cdot)$ and $\psi_r = (\psi_{r,1}, \cdot)$ for $\psi_{s,1} \in \mathtt{x}_t$ and $\psi_{r,1} \in \mathtt{x}_t$ hold. If hold, proceed.
- Emulating honest maintainer, upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_t, (\mathtt{x}_t, \pi_t))$ (e.g., if $\mathcal{A}$ does not block $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$), submit $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}_t, \pi_t)$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w}_t)$ from $\mathcal{A}$, check if $(\mathtt{x}_t, \mathtt{w}_t) \in \mathcal{R}$ holds. If holds, store $(\mathtt{x}_t, \pi_t)$ and submit $(\mathtt{RvkAnm}, \mathsf{sid}, t^*_{\mathsf{id}})$ on behalf of malicious maintainer M_t to $\mathcal{F}_{\mathsf{CBDC}}$.
- Upon receiving $(\mathtt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$ (for a maintainer M_j) from $\mathcal{F}_{\mathsf{CBDC}}$, proceed as follows.
- Emulating $\mathcal{F}_{\mathsf{KR}}$, retrieve $(\mathsf{U}_s, \mathsf{pk}_s)$ and $(\mathsf{U}_r, \mathsf{pk}_r)$.
- Parse $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$, $\psi_s = (\psi_{s,1}, \psi_{s,2}, \psi_{s,3})$ and $\psi_r = (\psi_{r,1}, \psi_{r,2})$.
- Set $\mathtt{x}_w \leftarrow (\psi_{s,1}, \psi_{r,1}, A_w, B_w, C_w)$ where A_w, B_w and C_w are simulated using Lagrange interpolation in a way that threshold decryption of (ψ_s, ψ_r) result in pk_s, g^v, and pk_r as described in details in Section 5.1.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}_w)$ to $\mathcal{A}$. Upon receiving back $(\mathtt{Proof}, \mathsf{sid}, \pi_w)$ from $\mathcal{A}$, record $(\mathtt{x}_w, \pi_w)$ and proceed.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, output the associated leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ to $\mathcal{A}$.
- Emulating $\mathcal{F}_{\mathsf{KR}}$, output the associated leakages of $\mathcal{F}_{\mathsf{KR}}$ to $\mathcal{A}$.

(ii) **Honest (resp. malicious) U_s and malicious (resp. honest) U_r, or malicious U_s and malicious U_r; and at most t malicious maintainers for both *currency issuance* and *payment* protocols:** The simulation of this case is similar to the case of honest U_s and honest U_r, except that there is no need to simulate the threshold decryption shares of honest maintainers. The reason is that, in this case, $\mathcal{S}$ has already known the identities of the participants U_s and U_r, as well as the transaction value v. Hence, $\mathcal{S}$, on behalf of honest maintainers, computes the decryption shares by following the real-world algorithm described in Figure 22.

Tracing.

Remark 3. Our construction achieves a stronger form of post-tracing privacy. However, for the sake of keeping the functionality concise we exclude that property in the functionality.
(i) **Honest U and at most t malicious maintainers for both *currency issuance* and *payment* protocols:**

- Upon receiving $(\mathtt{Trace}, \mathsf{sid}, \mathsf{U}_j, \mathsf{M}_w)$ from $\mathcal{F}_{\mathsf{CBDC}}$, start emulating honest maintainer.
- Emulating honest maintainers if there exists a user record UR recorded for U_j, submit $(\mathtt{Trace.0k}, \mathsf{sid}, \mathsf{U}_j)$ to $\mathcal{F}_{\mathsf{CBDC}}$.
- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, receive $\mathcal{A}$'s call (e.g., on behalf of M_t) of the form: $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_i, (\overline{\mathtt{x}}_t, \overline{\pi}_t))$ and submit leakages accordingly to $\mathcal{A}$.
- Parse $\overline{\mathtt{x}}_t = (\tilde{\mathsf{com}}_t, \dot{g}^{a_t}, \dot{g})$.
- Emulating honest maintainers, check if there exists a user record $\mathsf{UR} = (\cdot, \cdot, \mathsf{com}_{\mathbb{M}}, \mathsf{U}^*)$ where parsing $\mathsf{com}_{\mathbb{M}} = \{\tilde{\mathsf{com}}_j\}_{j=1}^{D}$, $\tilde{\mathsf{com}}_t \in \mathsf{com}_{\mathbb{M}}$ holds. If holds, proceed.
- Emulating honest maintainer, upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_t, (\overline{\mathtt{x}}_t, \overline{\pi}_t))$ (e.g., if $\mathcal{A}$ does not block $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$), submit $(\mathtt{Verify}, \mathsf{sid}, \overline{\mathtt{x}}_t, \overline{\pi}_t)$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Witness}, \mathsf{sid}, \overline{\mathtt{w}}_t)$ from $\mathcal{A}$, check if $(\overline{\mathtt{x}}_t, \overline{\mathtt{w}}_t) \in \mathcal{R}$ holds. If holds, store $(\overline{\mathtt{x}}_t, \overline{\pi}_t)$ and submit $(\mathtt{Trace}, \mathsf{sid}, \mathsf{U}^*)$ on behalf of malicious maintainer M_t to $\mathcal{F}_{\mathsf{CBDC}}$.
- Upon receiving $(\mathtt{Traced}, \mathsf{sid}, \mathsf{U}, \{t^{\tau}_{\mathsf{id}}, \mathsf{role}^{\tau}\}_{\tau=1}^{x})$ (for a maintainer M_j) from $\mathcal{F}_{\mathsf{CBDC}}$, proceed as follows.
- Depending on role^{τ} retrieve the associated tag $\mathsf{T}^{\tau} \in t^{\tau}_{\mathsf{id}}$.
- Given U, retrieve $\tilde{\mathsf{com}}_w$.
- Set $\overline{\mathtt{x}}_w \leftarrow (\tilde{\mathsf{com}}_w, E_w, \dot{g})$ where E_w is simulated using Lagrange interpolation in a way that threshold tracing tag computation result in t^{τ}_{id} as described in details in Section 5.1.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\mathtt{Prove}, \mathsf{sid}, \overline{\mathtt{x}}_w)$ to $\mathcal{A}$. Upon receiving back $(\mathtt{Proof}, \mathsf{sid}, \overline{\pi}_w)$ from $\mathcal{A}$, record $(\overline{\mathtt{x}}_w, \overline{\pi}_w)$ and proceed.

- Emulating $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$, output the associated leakages of $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ to $\mathcal{A}$.

(ii) **Malicious U and at most t malicious maintainers for both *currency issuance* and *payment* protocols:** The simulation of this case is similar to the honest U case above except that there is no need to simulate the tracing tag shares of honest maintainers. $\mathcal{S}$ on behalf of honest maintainers participate at computing the tracing tags as described in Figure 24.

6 Implementation details and PEReDi performance

We measured the performance of PEReDi transactions on an Intel Core i7-9850H CPU @ 2.60 GHz with 16 GB of RAM using Ubuntu 20.04.2 LTS. Table 1 lists the size of the field and group elements, as well as the exponentiation running time and pairing cost, using the Charm-Crypto framework [4], a Python library for pairing-based cryptography. $\mathsf{E}, \tilde{\mathsf{E}}, \mathsf{E}_t$, and P denote exponentiation in $\mathbb{G}$, $\tilde{\mathbb{G}}$, and $\mathbb{G}_t$, and pairing, respectively. We applied the Barreto-Naehrig (BN) curve, type F, defined by the equation $y^2 = x^3 + b$ over a field of order p, with an embedding curve degree of $k = 12$ and a 1920-bit discrete logarithm (DLog) security level. For simplicity, computations over $\mathbb{Z}_p$ and hash functions are not taken into account.

Table 1: Size of parameters and cost of operations

Parameter:	$\lvert\mathbb{Z}_p\rvert$	$\lvert\mathbb{G}\rvert$	$\lvert\tilde{\mathbb{G}}\rvert$	$\lvert\mathbb{G}_t\rvert$
Size (bytes):	46	46	90	514

Parameter: field operations		E	$\tilde{\mathsf{E}}$	E_t	P
Time (ms):	*negligible*	0.89	1.58	5.36	23.32

The summary of time complexity and communication costs with respect to the number of maintainers and the ranges required for regulatory compliance are shown in Table 2 and Table 3, respectively[21].

Table 2: Time complexity of PEReDi transactions considering *all regulatory compliance-related information* in a user's account. We assume that $B_{\mathsf{max}} = 2^{n_b} - 1, S_{\mathsf{max}} = 2^{n_s} - 1, R_{\mathsf{max}} = 2^{n_r} - 1$, and $V_{\mathsf{max}} = 2^{n_v} - 1$. t is the maximum number of malicious maintainers, where the total number of maintainers is $D = 5t + 1$.

Transaction	**Sender (ms)**	**Receiver (ms)**
Issuance	3.56 (for B)	$7.12(n_b + n_r) + 310.86t + 192.21$
Payment	$7.12(n_v + n_s + n_b) + 310.86t + 193.99$	$7.12(n_b + n_r) + 310.86t + 193.99$
	Maintainer (ms)	
Issuance	$3.56(n_b + n_r) + 81.15$	
Payment	$3.56(n_v + 2n_b + n_r + n_s) + 162.3$	

[21] For the sake of completeness and to provide some examples of concrete values, one may set n_v to 32, and other values can be set accordingly.

Table 3: Communication cost of PEReDi transactions considering *all regulatory compliance-related information* in user's account. We assume that $B_{\mathsf{max}} = 2^{n_b} - 1, S_{\mathsf{max}} = 2^{n_s} - 1, R_{\mathsf{max}} = 2^{n_r} - 1$ and $V_{\mathsf{max}} = 2^{n_v} - 1$. $\mathbb{G}$ and $\tilde{\mathbb{G}}$ means group elements, and $\mathbb{F}$ means field elements.

Transactions	**Sender**
Issuance	$3\ \mathbb{G}$ (for B)
Payment	$48\ \mathbb{G} + 2\log_2(n_v)\ \mathbb{G} + 2\log_2(n_b)\ \mathbb{G} + 2\log_2(n_s)\ \mathbb{G} + 2\ \tilde{\mathbb{G}} + 19\ \mathbb{F}$
	Receiver
Issuance	$40\ \mathbb{G} + 2\log_2(n_b)\ \mathbb{G} + 2\log_2(n_r)\ \mathbb{G} + 2\ \tilde{\mathbb{G}} + 19\ \mathbb{F}$
Payment	$42\ \mathbb{G} + 2\log_2(n_b)\ \mathbb{G} + 2\log_2(n_r)\ \mathbb{G} + 2\ \tilde{\mathbb{G}} + 19\ \mathbb{F}$
	Maintainer
Issuance	$2\ \mathbb{G}$
Payment	$2\ \mathbb{G}$

See Appendix A for more details on implementation and performance. Note that, as proposed in [57,56], implementing a constant-time decryption mechanism can remove the need for brute-force attempts to retrieve v from the ElGamal ciphertext, where g^v is encrypted.
To instantiate $\mathcal{F}_{\mathsf{NIZK}}$ and $\mathcal{F}_{\mathsf{SoK}}$ efficiently, one can either follow the approach of [46], which is based on Fischlin's transform [44], or construct a simulation-extractable NIZK/SoK from a simulation-sound NIZK and a CPA-secure encryption scheme, as described by [34]. Instead, to allow for an apple-to-apple comparison with existing schemes, we analyze performance in the more commonly used stand-alone setting, e.g., [60], in which one can employ the plain Fiat-Shamir transform.

6.1 Fiat-Shamir transform

All the proofs presented in this section as an interactive protocol with a logarithmic number of rounds can be converted into a non-interactive protocol that is secure and zero-knowledge in the random oracle model using the Fiat-Shamir transform [10]. All random challenges are replaced by hashes of the transcript up to that point, including the statement itself.

6.2 Range proofs

For the range proofs of PEReDi, we use Bulletproofs [18]. The range proof relation is defined as follows:

$$\{(\text{Public input: } h, g \in \mathbb{G}, A, n; \text{witness: } v, \gamma \in \mathbb{Z}_p) : A = g^{\gamma} \cdot h^{v} \wedge v \in [0, 2^n - 1]\}$$

According to comparisons made in [47], the computational complexity of one range proof of the form above is $8n$ group exponentiations on the prover's side and $4n$ group exponentiations on the verifier's side. We note that using other techniques for range proofs (rather than Bulletproofs) can result in better efficiency.

Acknowledgements

This work was supported by Input Output (iohk.io) through their funding of the Edinburgh Blockchain Technology Lab.

References

1. Central bank digital currency—opportunities, challenges and design. Bank of England, Discussion Paper, March (2020)

2. SAR Online User Guidance. National Crime Agency (2021), Available in this Link
3. Know Your Transaction (KYT) – The Key to Combating Transaction Laundering. Insights into Payments and Beyond (December 2020), Available in this Link
4. Akinyele, J.A., Garman, C., Miers, I., Pagano, M.W., Rushanan, M., Green, M., Rubin, A.D.: Charm: a framework for rapidly prototyping cryptosystems. Journal of Cryptographic Engineering **3**(2), 111–128 (Jun 2013). https://doi.org/10.1007/s13389-013-0057-3
5. Allen, S., Čapkun, S., Eyal, I., Fanti, G., Ford, B.A., Grimmelmann, J., Juels, A., Kostiainen, K., Meiklejohn, S., Miller, A., et al.: Design choices for central bank digital currency: Policy and technical considerations. Tech. rep., National Bureau of Economic Research (2020)
6. Androulaki, E., Camenisch, J., De Caro, A., Dubovitskaya, M., Elkhiyaoui, K., Tackmann, B.: Privacy-preserving auditable token payments in a permissioned blockchain system. Cryptology ePrint Archive, Report 2019/1058 (2019), `https://eprint.iacr.org/2019/1058`
7. Baldimtsi, F., Chase, M., Fuchsbauer, G., Kohlweiss, M.: Anonymous transferable E-cash. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 101–124. Springer, Heidelberg (Mar / Apr 2015). https://doi.org/10.1007/978-3-662-46447-2_5
8. Bank, E.C.: Exploring anonymity in central bank digital currencies (2019)
9. Baudet, M., Danezis, G., Sonnino, A.: Fastpay: High-performance byzantine fault tolerant settlement. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 163–177 (2020)
10. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Proceedings of the 1st ACM Conference on Computer and Communications Security. pp. 62–73 (1993)
11. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE Computer Society Press (May 2014). https://doi.org/10.1109/SP.2014.36
12. Bindseil, U.: Tiered CBDC and the Financial System (2020), Available in this Link
13. BIS: Bank of canada, european central bank, bank of japan, sveriges riksbank, swiss national bank, bank of england, board of governors of the federal reserve, and bank for international settlements. central bank digital currencies: foundational principles and core features, (2020), Available in this Link
14. Bjerg, O.: Designing new money-the policy trilemma of central bank digital currency (2017)
15. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications (extended abstract). In: 20th ACM STOC. pp. 103–112. ACM Press (May 1988). https://doi.org/10.1145/62212.62222
16. Boyle, E., Cohen, R., Goel, A.: Breaking the $o(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 319–330 (2021)
17. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
18. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE symposium on security and privacy (SP). pp. 315–334. IEEE (2018)
19. Camenisch, J., Drijvers, M., Gagliardoni, T., Lehmann, A., Neven, G.: The wonderful world of global random oracles. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 280–312. Springer (2018)
20. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R. (ed.) Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3494, pp. 302–321. Springer (2005). https://doi.org/10.1007/11426639_18
21. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Balancing accountability and privacy using e-cash (extended abstract). In: Prisco, R.D., Yung, M. (eds.) SCN 06. LNCS, vol. 4116, pp. 141–155. Springer, Heidelberg (Sep 2006). https://doi.org/10.1007/11832072_10
22. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 93–118. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_7

23. Canard, S., Gouget, A.: Anonymity in transferable e-cash. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 08. LNCS, vol. 5037, pp. 207–223. Springer, Heidelberg (Jun 2008). https://doi.org/10.1007/978-3-540-68914-0_13
24. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
25. Canetti, R.: Security and composition of cryptographic protocols: A tutorial. Cryptology ePrint Archive, Report 2006/465 (2006), https://eprint.iacr.org/2006/465
26. Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., Peled, U.: UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1769–1787. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3423367
27. Chan, A.H., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. In: Nyberg, K. (ed.) Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding. Lecture Notes in Computer Science, vol. 1403, pp. 561–575. Springer (1998). https://doi.org/10.1007/BFb0054154
28. Chase, M., Lysyanskaya, A.: On signatures of knowledge. Cryptology ePrint Archive, Report 2006/184 (2006), https://eprint.iacr.org/2006/184
29. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) CRYPTO'82. pp. 199–203. Plenum Press, New York, USA (1982)
30. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM **28**(10), 1030–1044 (1985). https://doi.org/10.1145/4372.4373, https://doi.org/10.1145/4372.4373
31. Chaum, D., Evertse, J.: A secure and privacy-protecting protocol for transmitting personal information between organizations. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 118–167. Springer (1986). https://doi.org/10.1007/3-540-47721-7_10, https://doi.org/10.1007/3-540-47721-7_10
32. Chaum, D., Grothoff, C., Moser, T.: How to issue a central bank digital currency. arXiv preprint arXiv:2103.00254 (2021)
33. Crites, E., Kiayias, A., Kohlweiss, M., Sarencheh, A.: SyRA: Sybil-resilient anonymous signatures with applications to decentralized identity. Cryptology ePrint Archive, Paper 2024/379 (2024), https://eprint.iacr.org/2024/379
34. Damgård, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., Siniscalchi, L.: Balancing privacy and accountability in blockchain identity management. In: Paterson, K.G. (ed.) CT-RSA 2021. LNCS, vol. 12704, pp. 552–576. Springer, Heidelberg (May 2021). https://doi.org/10.1007/978-3-030-75539-3_23
35. Danezis, G., Meiklejohn, S.: Centrally banked cryptocurrencies. In: NDSS 2016. The Internet Society (Feb 2016)
36. Doerner, J., Kondi, Y., Lee, E., Shelat, A., Tyner, L.: Threshold bbs+ signatures for distributed anonymous credential issuance. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 773–789. IEEE (2023)
37. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
38. Garay, J.A., Katz, J., Kumaresan, R., Zhou, H.S.: Adaptively secure broadcast, revisited. In: Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing. pp. 179–186 (2011)
39. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 81–98. Springer, Heidelberg (Feb 2016)
40. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology **20**(1), 51–83 (Jan 2007). https://doi.org/10.1007/s00145-006-0347-3
41. Groth, J., Ostrovsky, R., Sahai, A.: Perfect non-interactive zero knowledge for np. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 339–358. Springer (2006)
42. Katz, J.: Round-optimal, fully secure distributed key generation. In: Annual International Cryptology Conference. pp. 285–316. Springer (2024)

43. Kiayias, A., Kohlweiss, M., Sarencheh, A.: Peredi: Privacy-enhanced, regulated and distributed central bank digital currencies. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22). pp. 1739–1752. ACM (2022). https://doi.org/10.1145/3548606.3560707
44. Kondi, Y., abhi shelat: Improved straight-line extraction in the random oracle model with applications to signature aggregation. Cryptology ePrint Archive, Paper 2022/393 (2022), https://eprint.iacr.org/2022/393
45. Kumhof, M., Noone, C.: Central bank digital currencies-design principles and balance sheet implications (2018)
46. Lysyanskaya, A., Rosenbloom, L.N.: Universally composable sigma-protocols in the global random-oracle model. Cryptology ePrint Archive, Paper 2022/290 (2022), https://eprint.iacr.org/2022/290
47. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM Press (Nov 2019). https://doi.org/10.1145/3319535.3339817
48. Metere, R., Dong, C.: Automated cryptographic analysis of the pedersen commitment scheme. In: Computer Network Security: 7th International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2017, Warsaw, Poland, August 28-30, 2017, Proceedings 7. pp. 275–287. Springer (2017)
49. Micali, S., Rabin, M., Kilian, J.: Zero-knowledge sets. In: 44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings. pp. 80–91. IEEE (2003)
50. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized business review (2008)
51. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
52. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
53. Pointcheval, D., Sanders, O.: Short randomizable signatures. Cryptology ePrint Archive, Paper 2015/525 (2015), https://eprint.iacr.org/2015/525
54. Rial, A., Piotrowska, A.M.: Security analysis of coconut, an attribute-based credential scheme with threshold issuance. Cryptology ePrint Archive (2022)
55. Salvo, M.D.: Tornado Cash User 'Dusts' Hundreds of Public Wallets (Aug 9, 2022), Available in this Link
56. Sarencheh, A., Khoshakhlagh, H., Kavousi, A., Kiayias, A.: Dart: Decentralized, anonymous, and regulation-friendly tokenization. Cryptology ePrint Archive (2025)
57. Sarencheh, A., Kiayias, A., Kohlweiss, M.: Parscoin: A privacy-preserving, auditable, and regulation-friendly stablecoin. Cryptology ePrint Archive (2023)
58. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
59. Sonnino, A., Al-Bassam, M., Bano, S., Meiklejohn, S., Danezis, G.: Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In: NDSS 2019. The Internet Society (Feb 2019)
60. Tomescu, A., Bhat, A., Applebaum, B., Abraham, I., Gueta, G., Pinkas, B., Yanai, A.: Utt: Decentralized ecash with accountable privacy. Cryptology ePrint Archive (2022)
61. Wüst, K., Kostiainen, K., Capkun, S.: Platypus: A central bank digital currency with unlinkable transactions and privacy preserving regulation. Cryptology ePrint Archive, Report 2021/1443 (2021), https://eprint.iacr.org/2021/1443
62. Wüst, K., Kostiainen, K., Capkun, V., Capkun, S.: PRCash: Fast, private and regulated transactions for digital currencies. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 158–178. Springer, Heidelberg (Feb 2019). https://doi.org/10.1007/978-3-030-32101-7_11
63. Yao, Q.: A systematic framework to understand central bank digital currency. Science China Information Sciences **61**(3), 1–8 (2018)

A Implementation details and PEReDi performance (continued)

A.1 Performance details of currency issuance

1. Central bank needs to compute $\psi = (\psi_1, \psi_2, \psi_3) = (g^\rho, \mathsf{pk}_{1,\mathbb{M}}^\rho \cdot \mathsf{pk}_{\mathsf{U}}, \mathsf{pk}_{2,\mathbb{M}}^\rho \cdot g^v)$ which requires 4 exponentiation in $\mathbb{G}$ and 2 multiplication in $\mathbb{G}$.

2. User needs to compute $\mathsf{TI}_{\mathsf{U}} = (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$. Associated computation complexities to compute each element of TI_{U} are as follows.
   (a) ψ: Similar to central bank user needs to perform 4 exponentiation in $\mathbb{G}$ and 2 multiplication in $\mathbb{G}$ to compute ψ.
   (b) $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$: For $\mathsf{acc}^{\mathsf{new},\mathfrak{B}} = (\mathsf{com}, \{\mathsf{com}_\tau\}_{\tau=1}^{6}, h)$ user performs 19 exponentiation in $\mathbb{G}$, 12 multiplication in $\mathbb{G}$ and 1 hash.
   (c) $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$: For re-randomizing signature user parses $\sigma_{\mathbb{M}}$ as (h, s), picks $r \xleftarrow{\$} \mathbb{Z}_p$ and sets $r' \xleftarrow{\$} \mathbb{Z}_p$. Then, it computes $\sigma_{\mathbb{M}}^{\mathsf{int}} = (h', s') \leftarrow (h^{r'}, s^{r'}(h')^r)$. It computes $\kappa \leftarrow \tilde{\alpha} \prod_{\tau=1}^{6} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^r$. Sets $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}} = (\sigma_{\mathbb{M}}^{\mathsf{int}}, \kappa) = (\sigma_{\mathbb{M}}^{\mathsf{int}}, \tilde{\alpha} \prod_{\tau=1}^{6} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^r)$. Hence, computing $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ requires 3 exponentiation in $\mathbb{G}$, 1 multiplication in $\mathbb{G}$, 7 exponentiation in $\tilde{\mathbb{G}}$, and 7 multiplication in $\tilde{\mathbb{G}}$.
   (d) $\mathsf{T} = g^{a^{x+1}}$: It requires 1 exponentiation in field, and 1 exponentiation in $\mathbb{G}$.
   (e) π: To compute computation complexity of proof π we describe the details of Sigma protocol between the prover (user) and the verifier (each maintainer).
   The witness of the user is $\mathtt{w} = ((B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}, \varphi = a^x, a), \rho, o, \{o_\tau\}_{\tau=1}^{6}, r, r_1, r_2, v)$, the statement is $\mathtt{x} = (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T})$, and the relation is

$$
\begin{aligned}
&\{\psi_1 = g^\rho \wedge \psi_2 = \mathsf{pk}_{1,\mathbb{M}}^\rho \cdot g^{\mathsf{sk}} \wedge \psi_3 = \mathsf{pk}_{2,\mathbb{M}}^\rho \cdot g^v \wedge \\
&\mathsf{com} = g^o \cdot h_1^{B^{\mathsf{old}}+v} \cdot h_2^{S^{\mathsf{old}}} \cdot h_3^{R^{\mathsf{old}}+v} \cdot h_4^{\mathsf{sk}} \cdot h_5^{\varphi^{\mathsf{old}} \cdot a} \cdot h_6^a \wedge \\
&\mathsf{com}_1 = g^{o_1} \cdot h^{B^{\mathsf{old}}+v} \wedge \mathsf{com}_2 = g^{o_2} \cdot h^{S^{\mathsf{old}}} \wedge \mathsf{com}_3 = g^{o_3} \cdot h^{R^{\mathsf{old}}+v} \wedge \\
&\mathsf{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}} \wedge \mathsf{com}_5 = g^{o_5} \cdot h^{\varphi^{\mathsf{old}} \cdot a} \wedge \mathsf{com}_6 = g^{o_6} \cdot h^a \wedge \\
&\kappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{B^{\mathsf{old}}} \cdot \tilde{\beta}_2^{S^{\mathsf{old}}} \cdot \tilde{\beta}_3^{R^{\mathsf{old}}} \cdot \tilde{\beta}_4^{\mathsf{sk}} \cdot \tilde{\beta}_5^{\varphi^{\mathsf{old}}} \cdot \tilde{\beta}_6^a \cdot \tilde{g}^r \wedge \\
&\mathsf{T} = g^{\varphi^{\mathsf{old}} \cdot a} \wedge N = g^{r_1} \cdot h^{\varphi^{\mathsf{old}}} \wedge \mathsf{com}_5 = N^a \cdot g^{r_2} \wedge \\
&B^{\mathsf{new}} = B^{\mathsf{old}} + v \leq B_{\mathsf{max}} \wedge R^{\mathsf{new}} = R^{\mathsf{old}} + v \leq R_{\mathsf{max}}\}
\end{aligned}
$$

   We stress that prover sets $r_2 \leftarrow o_5 - ar_1$. Also, all values are included in the (defined) statement instead of N which is sent by the prover to the verifier as part of the proof. First of all, the prover and verifier execute 2 range proofs using bulletproofs (as defined in 6.2) where the relations are as follows: $\{(g, h \in \mathbb{G}, \mathsf{com}_1, B_{\mathsf{max}}; o_1, B^{\mathsf{new}} \in \mathbb{Z}_p) : \mathsf{com}_1 = g^{o_1} \cdot h^{B^{\mathsf{new}}} \wedge B^{\mathsf{new}} \in [0, B_{\mathsf{max}}]\}$ and $\{(g, h \in \mathbb{G}, \mathsf{com}_3, R_{\mathsf{max}}; o_3, S^{\mathsf{new}} \in \mathbb{Z}_p) : \mathsf{com}_3 = g^{o_3} \cdot h^{R^{\mathsf{new}}} \wedge R^{\mathsf{new}} \in [0, R_{\mathsf{max}}]\}$. Then the prover and verifier run a sigma protocol (where in the non-interactive version all random challenges are replaced by hashes of the transcript up to that point, including the statement itself, so the hash in the following Sigma protocol contains bulletproof's transcripts as well). The commitments used in the range proof relations are exactly the commitments used in the sigma protocol explained in the following. The prover and verifier execute the following (interactive) Sigma protocol:
      i. Prover computes:

$$
\begin{aligned}
\psi_1' = g^{\eta_1}, \quad \psi_2' = \mathsf{pk}_{1,\mathbb{M}}^{\eta_1} \cdot g^{\eta_2}, \quad \psi_3' = \mathsf{pk}_{2,\mathbb{M}}^{\eta_1} \cdot g^{\eta_3}, \\
\mathsf{com}' = g^{\eta_4} \cdot h_1^{\eta_5+\eta_3} \cdot h_2^{\eta_6} \cdot h_3^{\eta_7+\eta_3} \cdot h_4^{\eta_2} \cdot h_5^{\eta_{17}} \cdot h_6^{\eta_8}, \\
\mathsf{com}_1' = g^{\eta_{10}} \cdot h^{\eta_5+\eta_3}, \quad \mathsf{com}_2' = g^{\eta_{11}} \cdot h^{\eta_6}, \quad \mathsf{com}_3' = g^{\eta_{12}} \cdot h^{\eta_7+\eta_3}, \\
\mathsf{com}_4' = g^{\eta_{13}} \cdot h^{\eta_2}, \quad \mathsf{com}_5' = g^{\eta_{14}} \cdot h^{\eta_{17}}, \quad \mathsf{com}_6' = g^{\eta_{15}} \cdot h^{\eta_8}, \\
\kappa' = \tilde{\alpha} \cdot \tilde{\beta}_1^{\eta_5} \cdot \tilde{\beta}_2^{\eta_6} \cdot \tilde{\beta}_3^{\eta_7} \cdot \tilde{\beta}_4^{\eta_2} \cdot \tilde{\beta}_5^{\eta_9} \cdot \tilde{\beta}_6^{\eta_8} \cdot \tilde{g}^{\eta_{16}}, \\
\mathsf{T}' = g^{\eta_{17}}, \quad N' = g^{\eta_{18}} \cdot h^{\eta_9}, \quad \mathsf{com}_5'' = N^{\eta_8} \cdot g^{\eta_{19}}.
\end{aligned}
$$

Prover sends $\psi'_1, \psi'_2, \psi'_3, \mathsf{com}', \mathsf{com}'_1, \mathsf{com}'_2, \mathsf{com}'_3, \mathsf{com}'_4, \mathsf{com}'_5, \mathsf{com}'_6, \kappa', \mathsf{T}', N', \mathsf{com}''_5$ to the verifier.

ii. Verifier sends back challenge C.

iii. Prover computes:

$$\begin{aligned}
&\omega_1 = \eta_1 - \rho c, \quad \omega_2 = \eta_2 - \mathsf{sk} c, \quad \omega_3 = \eta_3 - v c, \quad \omega_4 = \eta_4 - o c,\\
&\omega_5 = \eta_5 - B^{\mathsf{old}} c, \quad \omega_6 = \eta_6 - S^{\mathsf{old}} c, \quad \omega_7 = \eta_7 - R^{\mathsf{old}} c, \quad \omega_8 = \eta_8 - a c,\\
&\omega_9 = \eta_9 - \varphi^{\mathsf{old}} c, \quad \omega_{10} = \eta_{10} - o_1 c, \quad \omega_{11} = \eta_{11} - o_2 c, \quad \omega_{12} = \eta_{12} - o_3 c,\\
&\omega_{13} = \eta_{13} - o_4 c, \quad \omega_{14} = \eta_{14} - o_5 c, \quad \omega_{15} = \eta_{15} - o_6 c, \quad \omega_{16} = \eta_{16} - r c,\\
&\omega_{17} = \eta_{17} - \varphi^{\mathsf{new}} c, \quad \omega_{18} = \eta_{18} - r_1 c, \quad \omega_{19} = \eta_{19} - r_2 c.
\end{aligned}$$

where $\varphi^{\mathsf{new}} = \varphi^{\mathsf{old}} \cdot a$. Prover sends $\omega_1, \ldots, \omega_{19}$ to the verifier.

iv. Verifier checks if:

$$\begin{aligned}
&\psi'_1 = \psi_1^c \cdot g^{\omega_1}, \quad \psi'_2 = \psi_2^c \cdot \mathsf{pk}_{1,\mathbb{M}}^{\omega_1} \cdot g^{\omega_2}, \quad \psi'_3 = \psi_3^c \cdot \mathsf{pk}_{2,\mathbb{M}}^{\omega_1} \cdot g^{\omega_3},\\
&\mathsf{com}' = \mathsf{com}^c \cdot g^{\omega_4} \cdot h_1^{\omega_5+\omega_3} \cdot h_2^{\omega_6} \cdot h_3^{\omega_7+\omega_3} \cdot h_4^{\omega_2} \cdot h_5^{\omega_{17}} \cdot h_6^{\omega_8},\\
&\mathsf{com}'_1 = \mathsf{com}_1^c \cdot g^{\omega_{10}} \cdot h^{\omega_5+\omega_3}, \quad \mathsf{com}'_2 = \mathsf{com}_2^c \cdot g^{\omega_{11}} \cdot h^{\omega_6}, \quad \mathsf{com}'_3 = \mathsf{com}_3^c \cdot g^{\omega_{12}} \cdot h^{\omega_7+\omega_3},\\
&\mathsf{com}'_4 = \mathsf{com}_4^c \cdot g^{\omega_{13}} \cdot h^{\omega_2}, \quad \mathsf{com}'_5 = \mathsf{com}_5^c \cdot g^{\omega_{14}} \cdot h^{\omega_{17}}, \quad \mathsf{com}'_6 = \mathsf{com}_6^c \cdot g^{\omega_{15}} \cdot h^{\omega_8},\\
&\kappa' = \kappa^c \cdot \tilde{\alpha}^{1-c} \cdot \tilde{\beta}_1^{\omega_5} \cdot \tilde{\beta}_2^{\omega_6} \cdot \tilde{\beta}_3^{\omega_7} \cdot \tilde{\beta}_4^{\omega_2} \cdot \tilde{\beta}_5^{\omega_9} \cdot \tilde{\beta}_6^{\omega_8} \cdot \tilde{g}^{\omega_{16}},\\
&\mathsf{T}' = \mathsf{T}^c \cdot g^{\omega_{17}}, \quad N' = N^c \cdot g^{\omega_{18}} \cdot h^{\omega_9}, \quad \mathsf{com}''_5 = \mathsf{com}_5^c \cdot N^{\omega_8} \cdot g^{\omega_{19}}.
\end{aligned}$$

The interactive protocol explained above is converted to non-interactive version using Fiat-Shamir transform. The computation complexity on the prover's side for non-interactive version is 1 hash, 29 exponentiation in $\mathbb{G}$, 7 exponentiation in $\tilde{\mathbb{G}}$, 16 multiplication in $\mathbb{G}$, 7 multiplication in $\tilde{\mathbb{G}}$, 23 field addition, 19 field multiplication, and $16n$ exponentiation in $\mathbb{G}$. Moreover, as explained above the prover computes N which is sent to the verifier as well that needs 2 exponentiation in $\mathbb{G}$ and 1 multiplication in $\mathbb{G}$.

The user also needs to unblind and aggregate the maintainers' signatures. Hence, we address each of them in the following.

(a) For unblinding signature: the user parses $\sigma_j^{\mathfrak{B}}$ as (h', c). Aborts if $h \neq h'$. Then, computes $\sigma_j = (h, s_j) \leftarrow (h, c \prod_{\tau=1}^{6} \beta_{j,\tau}^{-o_\tau})$. Hence, this step requires at maximum $6D$ exponentiation in $\mathbb{G}$ and $6D$ multiplication in $\mathbb{G}$. Afterwards, aborts if $e(h, \tilde{\alpha}_j \prod_{\tau=1}^{6} \tilde{\beta}_{j,\tau}^{m_\tau}) = e(s_j, \tilde{g})$ does not hold. Hence, at maximum this step requires $2D$ pairings, $6D$ exponentiation in $\tilde{\mathbb{G}}$ and $6D$ multiplication in $\tilde{\mathbb{G}}$.

(b) For aggregating signature: the user parses $\sigma_j = (h, s_j)$ and computes the signature $\sigma_{\mathbb{M}} = (h, s) \leftarrow (h, \prod_{j \in E} s_j^{l_j})$ which requires $D - t$ exponentiation in $\mathbb{G}$ and $D - t - 1$ multiplication in $\mathbb{G}$. Afterwards, aborts if $e(h, \tilde{\alpha} \prod_{\tau=1}^{6} \tilde{\beta}_\tau^{m_\tau}) = e(s, \tilde{g})$ does not hold which requires 2 pairings, 6 exponentiation in $\tilde{\mathbb{G}}$ and 6 multiplication in $\tilde{\mathbb{G}}$.

3. Each maintainer verifies the proof and re-randomized signature and generates a blind signature.

(a) For verification, each maintainer does the following:

i. Parses $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ as $(\sigma_{\mathbb{M}}^{\mathsf{int}} = (h', s'), \kappa)$ and aborts if $h' = 1$ or if $e(h', \kappa) = e(s', \tilde{g})$ does not hold which requires 1 pairing.

ii. Verifies π and aborts if the proof is not correct which means that the maintainer acts as what explained above for the proof verification as a result, considering the presented details above this step requires 1 hash, 42 exponentiation in $\mathbb{G}$, 9 exponentiation in $\tilde{\mathbb{G}}$, 29 multiplication in $\mathbb{G}$, 8 multiplication in $\tilde{\mathbb{G}}$, 4 field addition, and $8n$ exponentiation in $\mathbb{G}$.

(b) For blind signature, each maintainer does the following:

i. Sends com to $\mathcal{F}_{\mathrm{RO}}$ and receive h' from $\mathcal{F}_{\mathrm{RO}}$. Aborts if $h \neq h'$ which requires 1 hash.

ii. Computes $c = h^{x_j} \prod_{\tau=1}^{6} \mathsf{com}_\tau^{y_{j,\tau}}$ and sets the blind signature share $\sigma_j^{\mathfrak{B}} = (h, h^{x_j} \prod_{\tau=1}^{6} \mathsf{com}_\tau^{y_{j,\tau}})$ which requires 7 exponentiation in $\mathbb{G}$, and 6 multiplication in $\mathbb{G}$.

A.2 Performance details of payment

In the following, we present the details for the sender's side. The computation complexity on the receiver's side is similar to the sender's side except the fact that the receiver performs one range proof less than the sender.

1. The sender needs to compute $\mathsf{TI}_s = (\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s)$. Associated computation complexities to compute each element of TI_s are as follows.
   (a) ψ_s: The sender needs to perform 4 exponentiation in $\mathbb{G}$ and 2 multiplication in $\mathbb{G}$ to compute $\psi_s = (\psi_{s,1}, \psi_{s,2}, \psi_{s,3}) = (g^{\rho_s}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_s} \cdot \mathsf{pk}_s, \mathsf{pk}_{2,\mathbb{M}}^{\rho_s} \cdot g^v)$.
   (b) ψ_r: Computing $\psi_r = (\psi_{r,1}, \psi_{r,2}) = (g^{\rho_r}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_r} \cdot \mathsf{pk}_r)$ requires 2 exponentiation in $\mathbb{G}$ and 1 multiplication in $\mathbb{G}$.
   (c) $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$: Computing $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}} = (\mathsf{com}, \{\mathsf{com}_\tau\}_{\tau=1}^{6}, h)$ requires 19 exponentiation in $\mathbb{G}$, 12 multiplication in $\mathbb{G}$ and 1 hash.
   (d) $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$: For re-randomizing signature user parses $\sigma_{\mathbb{M}}$ as (h, s), picks $r \xleftarrow{\$} \mathbb{Z}_p$ and sets $r' \xleftarrow{\$} \mathbb{Z}_p$. Then, it computes $\sigma_{\mathbb{M}}^{\mathsf{int}} = (h', s') \leftarrow (h^{r'}, s^{r'}(h')^r)$. It computes $\kappa \leftarrow \tilde{\alpha} \prod_{\tau=1}^{6} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^r$. Sets $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}} = (\sigma_{\mathbb{M}}^{\mathsf{int}}, \kappa) = (\sigma_{\mathbb{M}}^{\mathsf{int}}, \tilde{\alpha} \prod_{\tau=1}^{6} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^r)$. Hence, computing $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ requires 3 exponentiation in $\mathbb{G}$, 1 multiplication in $\mathbb{G}$, 7 exponentiation in $\tilde{\mathbb{G}}$, and 7 multiplication in $\tilde{\mathbb{G}}$.
   (e) $\mathsf{T}_s = g^{a_s{}^{x_s+1}}$: It requires 1 exponentiation in field, and 1 exponentiation in $\mathbb{G}$.
   (f) $\overline{\sigma_s}(\psi_r)$: To compute computation complexity of signature of knowledge $\overline{\sigma_s}(\psi_r)$ we describe the details of Sigma protocol between the prover (sender) and the verifier (each maintainer).
   The witness of the sender is $\mathtt{w}_s = ((B^{\mathsf{old}}, S^{\mathsf{old}}, R^{\mathsf{old}}, \mathsf{sk}, \varphi^{\mathsf{old}} = a^x, a), \rho, o, \{o_\tau\}_{\tau=1}^{6}, r, r_1, r_2, v)$, the statement is $\mathtt{x}_s = (\psi_s, \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s)$, and the relation is

$$
\begin{aligned}
&\{\psi_{s,1} = g^{\rho} \wedge \psi_{s,2} = \mathsf{pk}_{1,\mathbb{M}}^{\rho} \cdot g^{\mathsf{sk}} \wedge \psi_{s,3} = \mathsf{pk}_{2,\mathbb{M}}^{\rho} \cdot g^{v} \wedge \\
&\mathsf{com} = g^{o} \cdot h_1^{B^{\mathsf{old}}-v} \cdot h_2^{S^{\mathsf{old}}+v} \cdot h_3^{R^{\mathsf{old}}} \cdot h_4^{\mathsf{sk}} \cdot h_5^{\varphi^{\mathsf{old}} \cdot a} \cdot h_6^{a} \wedge \\
&\mathsf{com}_1 = g^{o_1} \cdot h^{B^{\mathsf{old}}-v} \wedge \mathsf{com}_2 = g^{o_2} \cdot h^{S^{\mathsf{old}}+v} \wedge \mathsf{com}_3 = g^{o_3} \cdot h^{R^{\mathsf{old}}} \wedge \\
&\mathsf{com}_4 = g^{o_4} \cdot h^{\mathsf{sk}} \wedge \mathsf{com}_5 = g^{o_5} \cdot h^{\varphi^{\mathsf{old}} \cdot a} \wedge \mathsf{com}_6 = g^{o_6} \cdot h^{a} \wedge \\
&\kappa = \tilde{\alpha} \cdot \tilde{\beta}_1^{B^{\mathsf{old}}} \cdot \tilde{\beta}_2^{S^{\mathsf{old}}} \cdot \tilde{\beta}_3^{R^{\mathsf{old}}} \cdot \tilde{\beta}_4^{\mathsf{sk}} \cdot \tilde{\beta}_5^{\varphi^{\mathsf{old}}} \cdot \tilde{\beta}_6^{a} \cdot \tilde{g}^{r} \wedge \\
&\mathsf{T} = g^{\varphi^{\mathsf{old}} \cdot a} \wedge N = g^{r_1} \cdot h^{\varphi^{\mathsf{old}}} \wedge \mathsf{com}_5 = N^{a} \cdot g^{r_2} \wedge \\
&0 \leq v \leq V_{\mathsf{max}} \wedge B^{\mathsf{new}} = B^{\mathsf{old}} - v \geq 0 \wedge S^{\mathsf{new}} = S^{\mathsf{old}} + v \leq S_{\mathsf{max}}\}
\end{aligned}
$$

   Similar to currency issuance protocol, we stress that prover sets $r_2 \leftarrow o_5 - ar_1$. Also, all values are included in the (defined) statement instead of N which is sent by the prover to the verifier as part of the proof. First of all, the prover and verifier execute 3 range proofs using bulletproofs (as defined in 6.2) where the relations are as follows: $\{(\mathsf{pk}_{2,\mathbb{M}}, g \in \mathbb{G}, \psi_{s,3}, V_{\mathsf{max}}; \rho, v \in \mathbb{Z}_p) : \psi_{s,3} = \mathsf{pk}_{2,\mathbb{M}}^{\rho} \cdot g^{v} \wedge v \in [0, V_{\mathsf{max}}]\}$, $\{(g, h \in \mathbb{G}, \mathsf{com}_1, B_{\mathsf{max}}; o_1, B^{\mathsf{new}} \in \mathbb{Z}_p) : \mathsf{com}_1 = g^{o_1} \cdot h^{B^{\mathsf{new}}} \wedge B^{\mathsf{new}} \in [0, B_{\mathsf{max}}]\}$, and $\{(g, h \in \mathbb{G}, \mathsf{com}_2, S_{\mathsf{max}}; o_2, S^{\mathsf{new}} \in \mathbb{Z}_p) : \mathsf{com}_2 = g^{o_2} \cdot h^{S^{\mathsf{new}}} \wedge S^{\mathsf{new}} \in [0, S_{\mathsf{max}}]\}$. Then the prover and verifier run a sigma protocol (where in the non-interactive version

all random challenges are replaced by hashes of the transcript up to that point, including the statement itself and the message of signature of knowledge ψ_r, so the hash in the following Sigma protocol contains bulletproof's transcripts as well). The commitments used in the range proof relations are exactly the commitments used in the sigma protocol explained in the following. The prover and verifier execute the following (interactive) Sigma protocol:

i. Prover computes:

$$\begin{aligned}
\psi'_{s,1} &= g^{\eta_1}, \quad \psi'_{s,2} = \mathsf{pk}_{1,\mathbb{M}}^{\eta_1} \cdot g^{\eta_2}, \quad \psi'_{s,3} = \mathsf{pk}_{2,\mathbb{M}}^{\eta_1} \cdot g^{\eta_3}, \\
\mathsf{com}' &= g^{\eta_4} \cdot h_1^{\eta_5-\eta_3} \cdot h_2^{\eta_6+\eta_3} \cdot h_3^{\eta_7} \cdot h_4^{\eta_2} \cdot h_5^{\eta_{17}} \cdot h_6^{\eta_8}, \\
\mathsf{com}'_1 &= g^{\eta_{10}} \cdot h^{\eta_5-\eta_3}, \quad \mathsf{com}'_2 = g^{\eta_{11}} \cdot h^{\eta_6+\eta_3}, \quad \mathsf{com}'_3 = g^{\eta_{12}} \cdot h^{\eta_7}, \\
\mathsf{com}'_4 &= g^{\eta_{13}} \cdot h^{\eta_2}, \quad \mathsf{com}'_5 = g^{\eta_{14}} \cdot h^{\eta_{17}}, \quad \mathsf{com}'_6 = g^{\eta_{15}} \cdot h^{\eta_8}, \\
\kappa' &= \tilde{\alpha} \cdot \tilde{\beta}_1^{\eta_5} \cdot \tilde{\beta}_2^{\eta_6} \cdot \tilde{\beta}_3^{\eta_7} \cdot \tilde{\beta}_4^{\eta_2} \cdot \tilde{\beta}_5^{\eta_9} \cdot \tilde{\beta}_6^{\eta_8} \cdot \tilde{g}^{\eta_{16}}, \\
\mathsf{T}' &= g^{\eta_{17}}, \quad N' = g^{\eta_{18}} \cdot h^{\eta_9}, \quad \mathsf{com}''_5 = N^{\eta_8} \cdot g^{\eta_{19}}.
\end{aligned}$$

Prover sends $\psi'_{s,1}, \psi'_{s,2}, \psi'_{s,3}, \mathsf{com}', \mathsf{com}'_1, \mathsf{com}'_2, \mathsf{com}'_3, \mathsf{com}'_4, \mathsf{com}'_5, \mathsf{com}'_6, \kappa', \mathsf{T}', N', \mathsf{com}''_5$ to the verifier.

ii. Verifier sends back challenge C.

iii. Prover computes:

$$\begin{aligned}
\omega_1 &= \eta_1 - \rho c, \quad \omega_2 = \eta_2 - \mathsf{sk} c, \quad \omega_3 = \eta_3 - v c, \quad \omega_4 = \eta_4 - o c, \\
\omega_5 &= \eta_5 - B^{\mathsf{old}} c, \quad \omega_6 = \eta_6 - S^{\mathsf{old}} c, \quad \omega_7 = \eta_7 - R^{\mathsf{old}} c, \quad \omega_8 = \eta_8 - a c, \\
\omega_9 &= \eta_9 - \varphi^{\mathsf{old}} c, \quad \omega_{10} = \eta_{10} - o_1 c, \quad \omega_{11} = \eta_{11} - o_2 c, \quad \omega_{12} = \eta_{12} - o_3 c, \\
\omega_{13} &= \eta_{13} - o_4 c, \quad \omega_{14} = \eta_{14} - o_5 c, \quad \omega_{15} = \eta_{15} - o_6 c, \quad \omega_{16} = \eta_{16} - r c, \\
\omega_{17} &= \eta_{17} - \varphi^{\mathsf{new}} c, \quad \omega_{18} = \eta_{18} - r_1 c, \quad \omega_{19} = \eta_{19} - r_2 c.
\end{aligned}$$

Prover sends $\omega_1, \ldots, \omega_{19}$ to the verifier.

iv. Verifier checks if:

$$\begin{aligned}
\psi'_{s,1} &= \psi_{s,1}^c \cdot g^{\omega_1}, \quad \psi'_{s,2} = \psi_{s,2}^c \cdot \mathsf{pk}_{1,\mathbb{M}}^{\omega_1} \cdot g^{\omega_2}, \quad \psi'_{s,3} = \psi_{s,3}^c \cdot \mathsf{pk}_{2,\mathbb{M}}^{\omega_1} \cdot g^{\omega_3}, \\
\mathsf{com}' &= \mathsf{com}^c \cdot g^{\omega_4} \cdot h_1^{\omega_5-\omega_3} \cdot h_2^{\omega_6+\omega_3} \cdot h_3^{\omega_7} \cdot h_4^{\omega_2} \cdot h_5^{\omega_{17}} \cdot h_6^{\omega_8}, \\
\mathsf{com}'_1 &= \mathsf{com}_1^c \cdot g^{\omega_{10}} \cdot h^{\omega_5-\omega_3}, \quad \mathsf{com}'_2 = \mathsf{com}_2^c \cdot g^{\omega_{11}} \cdot h^{\omega_6+\omega_3}, \quad \mathsf{com}'_3 = \mathsf{com}_3^c \cdot g^{\omega_{12}} \cdot h^{\omega_7}, \\
\mathsf{com}'_4 &= \mathsf{com}_4^c \cdot g^{\omega_{13}} \cdot h^{\omega_2}, \quad \mathsf{com}'_5 = \mathsf{com}_5^c \cdot g^{\omega_{14}} \cdot h^{\omega_{17}}, \quad \mathsf{com}'_6 = \mathsf{com}_6^c \cdot g^{\omega_{15}} \cdot h^{\omega_8}, \\
\kappa' &= \kappa^c \cdot \tilde{\alpha}^{1-c} \cdot \tilde{\beta}_1^{\omega_5} \cdot \tilde{\beta}_2^{\omega_6} \cdot \tilde{\beta}_3^{\omega_7} \cdot \tilde{\beta}_4^{\omega_2} \cdot \tilde{\beta}_5^{\omega_9} \cdot \tilde{\beta}_6^{\omega_8} \cdot \tilde{g}^{\omega_{16}}, \\
\mathsf{T}' &= \mathsf{T}^c \cdot g^{\omega_{17}}, \quad N' = N^c \cdot g^{\omega_{18}} \cdot h^{\omega_9}, \quad \mathsf{com}''_5 = \mathsf{com}_5^c \cdot N^{\omega_8} \cdot g^{\omega_{19}}.
\end{aligned}$$

The interactive protocol explained above is converted to non-interactive version using Fiat-Shamir transform. The computation complexity on the prover's side for non-interactive version is 1 hash, 29 exponentiation in $\mathbb{G}$, 7 exponentiation in $\tilde{\mathbb{G}}$, 16 multiplication in $\mathbb{G}$, 7 multiplication in $\tilde{\mathbb{G}}$, 23 field addition, 19 field multiplication, and $24n$ exponentiation in $\mathbb{G}$. Moreover, as explained above the prover computes N which is sent to the verifier as well that needs 2 exponentiation in $\mathbb{G}$ and 1 multiplication in $\mathbb{G}$.

The user also needs to unblind and aggregate the maintainers' signatures. Hence, we address each of them in the following.

(a) For unblinding signature: the user parses $\sigma_j^{\mathfrak{B}}$ as (h', c). Aborts if $h \neq h'$. Then, computes $\sigma_j = (h, s_j) \leftarrow (h, c \prod_{\tau=1}^{6} \beta_{j,\tau}^{-o_\tau})$. Hence, this step requires at maximum $6D$ exponentiation in $\mathbb{G}$ and $6D$ multiplication in $\mathbb{G}$. Afterwards, aborts if $e(h, \tilde{\alpha}_j \prod_{\tau=1}^{6} \tilde{\beta}_{j,\tau}^{m_\tau}) = e(s_j, \tilde{g})$ does not hold. Hence, at maximum this step requires $2D$ pairings, $6D$ exponentiation in $\tilde{\mathbb{G}}$ and $6D$ multiplication in $\tilde{\mathbb{G}}$.

(b) For aggregating signature: the user parses $\sigma_j = (h, s_j)$ and computes the signature $\sigma_{\mathbb{M}} = (h, s) \leftarrow (h, \prod_{j \in E} s_j^{l_j})$ which requires $D - t$ exponentiation in $\mathbb{G}$ and $D - t - 1$ multiplication in $\mathbb{G}$. Afterwards, aborts if $e(h, \tilde{\alpha} \prod_{\tau=1}^{6} \tilde{\beta}_\tau^{m_\tau}) = e(s, \tilde{g})$ does not hold which requires 2 pairings, 6 exponentiation in $\tilde{\mathbb{G}}$ and 6 multiplication in $\tilde{\mathbb{G}}$.

2. Each maintainer verifies the proof and re-randomized signature and generates a blind signature. All the following computation complexities are related to processing TI_s, processing TI_r requires the same computation complexity except one range proof less than processing TI_s. In our results, we have considered computation complexity for processing both TI_s and TI_r.
   (a) For verification, each maintainer does the following:
      i. Parses $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ as $(\sigma_{\mathbb{M}}^{\mathsf{int}} = (h', s'), \kappa)$ and aborts if $h' = 1$ or if $e(h', \kappa) = e(s', \tilde{g})$ does not hold which requires 1 pairing.
      ii. Verifies π and aborts if the proof is not correct which means that the maintainer acts as what explained above for the proof verification as a result, considering the presented details above this step requires 1 hash, 42 exponentiation in $\mathbb{G}$, 9 exponentiation in $\tilde{\mathbb{G}}$, 29 multiplication in $\mathbb{G}$, 8 multiplication in $\tilde{\mathbb{G}}$, 4 field addition, and $12n$ exponentiation in $\mathbb{G}$ (for TI_r it is $8n$ exponentiation in $\mathbb{G}$).
   (b) For blind signature, each maintainer does the following:
      i. Sends com to $\mathcal{F}_{\mathrm{RO}}$ and receive h' from $\mathcal{F}_{\mathrm{RO}}$. Aborts if $h \neq h'$ which requires 1 hash.
      ii. Computes $c = h^{x_j} \prod_{\tau=1}^{6} \mathsf{com}_\tau^{y_{j,\tau}}$ and sets the blind signature share $\sigma_j^{\mathfrak{B}} = (h, h^{x_j} \prod_{\tau=1}^{6} \mathsf{com}_\tau^{y_{j,\tau}})$ which requires 7 exponentiation in $\mathbb{G}$, and 6 multiplication in $\mathbb{G}$.

A.3 Details of zero-knowledge relations

In this section, we provide details on implementing the zero-knowledge relations used in *user registration, abort transaction, privacy revocation*, and *tracing* protocols based on Sigma protocols.

A.3.1 User registration. The prover and verifier execute the following Sigma protocol:

1. Prover sends:

$$\begin{aligned}
\mathsf{com}' &= g^{\eta_1} \cdot h_4^{\eta_2} \cdot h_5 \cdot h_6^{\eta_3}, \quad \mathsf{com}_1' = g^{\eta_4}, \quad \mathsf{com}_2' = g^{\eta_5}, \quad \mathsf{com}_3' = g^{\eta_6}, \\
\mathsf{com}_4' &= g^{\eta_7} \cdot h^{\eta_2}, \quad \mathsf{com}_5' = g^{\eta_8} \cdot h, \quad \mathsf{com}_6' = g^{\eta_9} \cdot h^{\eta_3}, \\
\{\tilde{\mathsf{com}}_j^* &= g^{\mu_j} h^{\gamma_j}\}_{j=1}^{D} \wedge \\
\mathsf{pk}' &= g^{\eta_2} \wedge \\
\{\tilde{\mathsf{com}}_j^{**} &= g^{\eta_3} \cdot g^{\alpha_1 j} \cdot \ldots \cdot g^{\alpha_{\beta-1} j^{\beta-1}} \cdot h^{\gamma_j}\}_{j=1}^{D}
\end{aligned}$$

   to the verifier.
2. Verifier sends back challenge C.
3. Prover sends:

$$\begin{aligned}
&\omega_1 = \eta_1 - oc, \quad \omega_2 = \eta_2 - \mathsf{sk}c, \quad \omega_3 = \eta_3 - ac, \quad \omega_4 = \eta_4 - o_1 c, \\
&\omega_5 = \eta_5 - o_2 c, \quad \omega_6 = \eta_6 - o_3 c, \quad \omega_7 = \eta_7 - o_4 c, \quad \omega_8 = \eta_8 - o_5 c, \quad \omega_9 = \eta_9 - o_6 c, \\
&\omega_1' = \mu_1 - a_1 c, \ldots, \omega_D' = \mu_D - a_D c, \\
&\omega_1'' = \gamma_1 - r_1 c, \ldots, \omega_D'' = \gamma_D - r_D c, \\
&\omega_1''' = \alpha_1 - \mathsf{coe}_1 c, \ldots, \omega_{\beta-1}''' = \alpha_{\beta-1} - \mathsf{coe}_{\beta-1} c
\end{aligned}$$

   to the verifier.

4. Verifier checks if:

$$\begin{aligned}
\mathsf{com}' &= \mathsf{com}^c \cdot g^{\omega_1} \cdot h_4^{\omega_2} \cdot h_5^{1-c} \cdot h_6^{\omega_3},\\
\mathsf{com}'_1 &= \mathsf{com}_1^c \cdot g^{\omega_4}, \quad \mathsf{com}'_2 = \mathsf{com}_2^c \cdot g^{\omega_5}, \quad \mathsf{com}'_3 = \mathsf{com}_3^c \cdot g^{\omega_6},\\
\mathsf{com}'_4 &= \mathsf{com}_4^c \cdot g^{\omega_7} \cdot h^{\omega_2}, \quad \mathsf{com}'_5 = \mathsf{com}_5^c \cdot g^{\omega_8} \cdot h^{1-c}, \quad \mathsf{com}'_6 = \mathsf{com}_6^c \cdot g^{\omega_9} \cdot h^{\omega_3},\\
\{\tilde{\mathsf{com}}_j^* &= \tilde{\mathsf{com}}_j^c \cdot g^{\omega'_j} h^{\omega''_j}\}_{j=1}^D \wedge\\
\mathsf{pk}' &= \mathsf{pk}^c \cdot g^{\omega_2} \wedge\\
\{\tilde{\mathsf{com}}_j^{**} &= \tilde{\mathsf{com}}_j^c \cdot g^{\omega_3} \cdot g^{\omega'''_1 j} \cdot \ldots \cdot g^{\omega'''_{\beta-1} j^{\beta-1}} \cdot h^{\omega''_j}\}_{j=1}^D.
\end{aligned}$$

A.3.2 Abort transaction. The prover and verifier execute the following Sigma protocol:

1. Prover computes:

$$\begin{aligned}
\mathsf{com}' &= g^{\eta_4} \cdot h_1^{\eta_5} \cdot h_2^{\eta_6} \cdot h_3^{\eta_7} \cdot h_4^{\eta_2} \cdot h_5^{\eta_{17}} \cdot h_6^{\eta_8},\\
\mathsf{com}'_1 &= g^{\eta_{10}} \cdot h^{\eta_5}, \quad \mathsf{com}'_2 = g^{\eta_{11}} \cdot h^{\eta_6}, \quad \mathsf{com}'_3 = g^{\eta_{12}} \cdot h^{\eta_7},\\
\mathsf{com}'_4 &= g^{\eta_{13}} \cdot h^{\eta_2}, \quad \mathsf{com}'_5 = g^{\eta_{14}} \cdot h^{\eta_{17}}, \quad \mathsf{com}'_6 = g^{\eta_{15}} \cdot h^{\eta_8},\\
\kappa' &= \tilde{\alpha} \cdot \tilde{\beta}_1^{\eta_5} \cdot \tilde{\beta}_2^{\eta_6} \cdot \tilde{\beta}_3^{\eta_7} \cdot \tilde{\beta}_4^{\eta_2} \cdot \tilde{\beta}_5^{\eta_9} \cdot \tilde{\beta}_6^{\eta_8} \cdot \tilde{g}^{\eta_{16}},\\
\mathsf{T}' &= g^{\eta_{17}}, \quad N' = g^{\eta_{18}} \cdot h^{\eta_9}, \quad \mathsf{com}''_5 = N^{\eta_8} \cdot g^{\eta_{19}}.
\end{aligned}$$

   Prover sends $\mathsf{com}', \mathsf{com}'_1, \mathsf{com}'_2, \mathsf{com}'_3, \mathsf{com}'_4, \mathsf{com}'_5, \mathsf{com}'_6, \kappa', \mathsf{T}', N'$ and com''_5 to the verifier.
2. Verifier sends back challenge C.
3. Prover computes:

$$\begin{aligned}
&\omega_2 = \eta_2 - \mathsf{sk}c, \quad \omega_4 = \eta_4 - oc, \quad \omega_5 = \eta_5 - B^{\mathsf{old}}c, \quad \omega_6 = \eta_6 - S^{\mathsf{old}}c,\\
&\omega_7 = \eta_7 - R^{\mathsf{old}}c, \quad \omega_8 = \eta_8 - ac, \quad \omega_9 = \eta_9 - \varphi^{\mathsf{old}}c, \quad \omega_{10} = \eta_{10} - o_1 c,\\
&\omega_{11} = \eta_{11} - o_2 c, \quad \omega_{12} = \eta_{12} - o_3 c, \quad \omega_{13} = \eta_{13} - o_4 c, \quad \omega_{14} = \eta_{14} - o_5 c,\\
&\omega_{15} = \eta_{15} - o_6 c, \quad \omega_{16} = \eta_{16} - rc, \quad \omega_{17} = \eta_{17} - \varphi^{\mathsf{new}}c, \quad \omega_{18} = \eta_{18} - r_1 c,\\
&\omega_{19} = \eta_{19} - r_2 c.
\end{aligned}$$

   Prover sends $\omega_2, \omega_4, \ldots, \omega_{19}$ to the verifier.
4. Verifier checks if:

$$\begin{aligned}
\mathsf{com}' &= \mathsf{com}^c \cdot g^{\omega_4} \cdot h_1^{\omega_5} \cdot h_2^{\omega_6} \cdot h_3^{\omega_7} \cdot h_4^{\omega_2} \cdot h_5^{\omega_{17}} \cdot h_6^{\omega_8},\\
\mathsf{com}'_1 &= \mathsf{com}_1^c \cdot g^{\omega_{10}} \cdot h^{\omega_5}, \quad \mathsf{com}'_2 = \mathsf{com}_2^c \cdot g^{\omega_{11}} \cdot h^{\omega_6}, \quad \mathsf{com}'_3 = \mathsf{com}_3^c \cdot g^{\omega_{12}} \cdot h^{\omega_7},\\
\mathsf{com}'_4 &= \mathsf{com}_4^c \cdot g^{\omega_{13}} \cdot h^{\omega_2}, \quad \mathsf{com}'_5 = \mathsf{com}_5^c \cdot g^{\omega_{14}} \cdot h^{\omega_{17}}, \quad \mathsf{com}'_6 = \mathsf{com}_6^c \cdot g^{\omega_{15}} \cdot h^{\omega_8},\\
\kappa' &= \kappa^c \cdot \tilde{\alpha}^{1-c} \cdot \tilde{\beta}_1^{\omega_5} \cdot \tilde{\beta}_2^{\omega_6} \cdot \tilde{\beta}_3^{\omega_7} \cdot \tilde{\beta}_4^{\omega_2} \cdot \tilde{\beta}_5^{\omega_9} \cdot \tilde{\beta}_6^{\omega_8} \cdot \tilde{g}^{\omega_{16}},\\
\mathsf{T}' &= \mathsf{T}^c \cdot g^{\omega_{17}}, \quad N' = N^c \cdot g^{\omega_{18}} \cdot h^{\omega_9}, \quad \mathsf{com}''_5 = \mathsf{com}_5^c \cdot N^{\omega_8} \cdot g^{\omega_{19}}.
\end{aligned}$$

A.3.3 Privacy revocation. Maintainer's witness is $\mathtt{w}_j = (\mathsf{sk}_{1,j}, \mathsf{sk}_{2,j})$, the statement is $\mathtt{x}_j = (\psi_{s,1}, \psi_{r,1}, \psi_{s,1}^{\mathsf{sk}_{1,j}} \psi_{s,1}^{\mathsf{sk}_{2,j}}, \psi_{r,1}^{\mathsf{sk}_{1,j}})$, and the relation is $\{\mathsf{pk}_{1,j} = g^{\mathsf{sk}_{1,j}} \wedge \psi_{s,1}^{\mathsf{sk}_{1,j}} \wedge \mathsf{pk}_{2,j} = g^{\mathsf{sk}_{2,j}} \wedge \psi_{s,1}^{\mathsf{sk}_{2,j}} \wedge \psi_{r,1}^{\mathsf{sk}_{1,j}}\}$. The prover and verifier execute the following Sigma protocol:

1. Prover computes: $\mathsf{pk}'_{1,j} = g^{\eta_1}, \psi_{s,1}^{\eta_1}, \mathsf{pk}'_{2,j} = g^{\eta_2}, \psi_{s,1}^{\eta_2}$ and $\psi_{r,1}^{\eta_1}$.
   Prover sends $\mathsf{pk}'_{1,j}, \psi_{s,1}^{\eta_1}, \mathsf{pk}'_{2,j}, \psi_{s,1}^{\eta_2}$ and $\psi_{r,1}^{\eta_1}$ to the verifier.
2. Verifier sends back challenge C.
3. Prover computes: $\omega_1 = \eta_1 - \mathsf{sk}_{1,j}c$ and $\omega_2 = \eta_2 - \mathsf{sk}_{2,j}c$.
   Prover sends ω_1 and ω_2 to the verifier.
4. Verifier checks if: $\mathsf{pk}'_{1,j} = \mathsf{pk}_{1,j}^c \cdot g^{\omega_1}, \psi_{s,1}^{\eta_1} = (\psi_{s,1}^{\mathsf{sk}_{1,j}})^c \cdot \psi_{s,1}^{\omega_1}, \mathsf{pk}'_{2,j} = \mathsf{pk}_{2,j}^c \cdot g^{\omega_2}, \psi_{s,1}^{\eta_2} = (\psi_{s,1}^{\mathsf{sk}_{2,j}})^c \cdot \psi_{s,1}^{\omega_2}$ and $\psi_{r,1}^{\eta_1} = (\psi_{r,1}^{\mathsf{sk}_{1,j}})^c \cdot \psi_{r,1}^{\omega_1}$

A.3.4 Tracing. Maintainer's witness is $\overline{\mathsf{w}}_j = (a_j, r_j)$, the statement is $\overline{\mathsf{x}}_j = (\tilde{\mathsf{com}}_j, \dot{g}^{a_j}, \dot{g})$, the relation is $\{\tilde{\mathsf{com}}_j = g^{a_j} \cdot h^{r_j} \wedge \dot{g}^{a_j}\}$. The prover and verifier execute the following Sigma protocol:

1. Prover computes: $\tilde{\mathsf{com}}_j^* = g^{\eta_1} \cdot h^{\eta_2}$ and $\dot{g}^{\eta_1}$.
   Prover sends $\tilde{\mathsf{com}}_j^*$ and $\dot{g}^{\eta_1}$ to the verifier.
2. Verifier sends back challenge C.
3. Prover computes: $\omega_1 = \eta_1 - a_j c$ and $\omega_2 = \eta_2 - r_j c$.
   Prover sends ω_1 and ω_2 to the verifier.
4. Verifier checks if: $\tilde{\mathsf{com}}_j^* = \tilde{\mathsf{com}}_j^c \cdot g^{\omega_1} \cdot h^{\omega_2}$ and $\dot{g}^{\eta_1} = (\dot{g}^{a_j})^c \cdot \dot{g}^{\omega_1}$.

B Cryptographic schemes and assumptions

In this section, we present the cryptographic primitives and assumptions that we use in our construction Π_{PEReDi}.

Definition 5 (Bilinear maps). *Let $\mathbb{G}$, $\tilde{\mathbb{G}}$, and $\mathbb{G}_T$ denote groups of prime order p. A map $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_T$ is called a bilinear map if it satisfies the following properties:*

- ***Bilinearity:*** *For all $g \in \mathbb{G}$, $\tilde{g} \in \tilde{\mathbb{G}}$, and $x, y \in \mathbb{Z}_p$, the map satisfies:*

$$e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}.$$

- ***Non-degeneracy:*** *For generators $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$, the element $e(g, \tilde{g})$ is a generator of $\mathbb{G}_T$.*
- ***Efficiency:*** *There exists an efficient setup algorithm $\mathcal{G}(1^k)$ that, given a security parameter k, outputs the tuple $(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_T, e, g, \tilde{g})$, where $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$ are generators. Furthermore, the computation of $e(a, b)$ for any $a \in \mathbb{G}$ and $b \in \tilde{\mathbb{G}}$ is efficient.*

In the case of Type-3 pairings, $\mathbb{G} \neq \tilde{\mathbb{G}}$, and no efficiently computable homomorphism between $\mathbb{G}$ and $\tilde{\mathbb{G}}$.

Definition 6 (d-strong decisional Diffie-Hellman (d-sDDH) assumption). *Let $\mathbb{G}$ be a cyclic group of prime order p, where p is a prime, and let g be a generator of $\mathbb{G}$. Suppose $(\mathbb{G}, p, g) \leftarrow \mathcal{G}(1^\lambda)$, where $\mathcal{G}$ is a group generation algorithm parameterized by the security parameter λ.*
For $x, x_1, \ldots, x_d \xleftarrow{\$} \mathbb{Z}_p$, the d-strong decisional Diffie-Hellman (d-sDDH) assumption asserts that the following holds relative to $\mathcal{G}$. For any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\lambda)$ such that:

$$\mathsf{Adv}_{\mathcal{A}}^{d\text{-sDDH}}(\lambda) = \Big| \Pr\left[\mathcal{A}(\mathbb{G}, p, g, g^x, g^{x^2}, \ldots, g^{x^d}) = 1\right] - \Pr\left[\mathcal{A}(\mathbb{G}, p, g, g^{x_1}, g^{x_2}, \ldots, g^{x_d}) = 1\right] \Big| \leq \mathsf{negl}(\lambda)$$

B.1 Public key encryption schemes

Definition 7 (Public key encryption scheme). *A public key encryption scheme consists of three algorithms $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, defined as follows:*

- $\mathsf{KeyGen}(1^\lambda)$*: A probabilistic algorithm that, given a security parameter λ, outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where pk is the public key and sk is the secret key. The public key pk implicitly specifies the message space $\mathcal{M}_{\mathsf{pk}}$, which represents the set of plaintext messages that can be encrypted.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*: A probabilistic algorithm that, given a public key pk and a message $m \in \mathcal{M}_{\mathsf{pk}}$, outputs a ciphertext c.*

- $\mathsf{Dec}(\mathsf{sk}, c)$*: A deterministic algorithm that, given a secret key* sk *and a ciphertext* c*, outputs the plaintext* m *if* c *is a valid ciphertext, or a special failure symbol* $\perp$ *if decryption fails.*

Definition 8 (Correctness of a public key encryption scheme). *A public key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is correct if, for all security parameters* $\lambda \in \mathbb{N}$*, for all key pairs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, and for all messages* $m \in \mathcal{M}_{\mathsf{pk}}$*, it holds that*

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$$

with overwhelming probability over the randomness of KeyGen *and* Enc*.*

Definition 9 (IND-CPA security of a public key encryption scheme). *A public key encryption (PKE) scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* IND-CPA*-secure if, for all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda)$ *in the following experiment is negligible in the security parameter* λ*:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) = \left| \Pr[\mathsf{IND\text{-}CPA}^1_{\mathsf{PKE}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{IND\text{-}CPA}^0_{\mathsf{PKE}}(\mathcal{A}, \lambda) = 1] \right| \leq \mathsf{negl}(\lambda).$$

The experiment $\mathsf{IND\text{-}CPA}^b_{\mathsf{PKE}}(\mathcal{A}, \lambda)$ *is defined as an interaction between a PPT adversary* $\mathcal{A}$ *and a challenger, parameterized by a security parameter* λ *and a random bit* $b \in \{0, 1\}$*:*

1. *The challenger generates a key pair* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and sends the public key* pk *to the adversary* $\mathcal{A}$*.*
2. *The adversary* $\mathcal{A}$ *submits two plaintext messages* (m_0, m_1) *such that* $m_0, m_1 \in \mathcal{M}_{\mathsf{pk}}$ *and* $|m_0| = |m_1|$*.*
3. *The challenger chooses a random bit* $b \xleftarrow{\$} \{0, 1\}$*, encrypts* m_b *to obtain the ciphertext* $c_b = \mathsf{Enc}(\mathsf{pk}, m_b)$*, and sends* c_b *to the adversary* $\mathcal{A}$*.*
4. *The adversary* $\mathcal{A}$ *outputs a guess* $b' \in \{0, 1\}$ *for* b*.*

The output of the experiment $\mathsf{IND\text{-}CPA}^b_{\mathsf{PKE}}(\mathcal{A}, \lambda)$ *is defined as 1 if* $b' = b$*, and 0 otherwise. The adversary's advantage in distinguishing between the two cases* $b = 0$ *and* $b = 1$ *is defined as:*

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

Definition 10 (ElGamal encryption scheme). *The ElGamal encryption scheme [37], denoted* EG*, is a public-key encryption (PKE) scheme defined over a prime-order cyclic group* $\mathbb{G}_q$ *with a generator* g*. The scheme consists of three algorithms* $\mathsf{EG} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, defined as follows:*

- $\mathsf{KeyGen}(1^\lambda)$*: The key generation algorithm proceeds as follows:*
  1. *Run a group generation algorithm* $\mathcal{G}(1^\lambda)$ *to obtain* (q, g)*, where* $\mathbb{G}_q$ *is a cyclic group of prime order* q *with generator* g*.*
  2. *Sample a secret key* $x \xleftarrow{\$} \mathbb{Z}_q$ *uniformly at random.*
  3. *Compute the public key component* $h = g^x$*.*
  4. *Output the public key* $\mathsf{pk} = (q, g, h)$ *and the secret key* $\mathsf{sk} = x$*.*
- $\mathsf{Enc}(\mathsf{pk}, M)$*: To encrypt a message* $M \in \mathbb{G}_q$*, the encryption algorithm proceeds as follows:*
  1. *Sample a random* $r \xleftarrow{\$} \mathbb{Z}_q$*.*
  2. *Compute the ciphertext components:* $u = g^r, \quad v = h^r \cdot M$*.*
  3. *Output the ciphertext* $\psi = (u, v)$*.*
- $\mathsf{Dec}(\mathsf{sk}, \psi)$*: Given a ciphertext* $\psi = (u, v)$*, the decryption algorithm proceeds as follows:*
  1. *Compute* $s = u^x$*.*
  2. *Recover the plaintext message* M *as:* $M = v \cdot s^{-1}$*.*

*The security of the ElGamal encryption scheme under chosen-plaintext attack (*IND-CPA*) relies on the hardness of the Decisional Diffie-Hellman (DDH) problem in* $\mathbb{G}_q$*.*

Definition 11 (Threshold ElGamal encryption scheme). *The Threshold ElGamal encryption scheme, denoted* T-EG*, is a threshold public-key encryption (PKE) scheme defined over a prime-order cyclic group $\mathbb{G}_q$ with a generator g. The scheme consists of four algorithms* $\mathsf{T\text{-}EG} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{PartialDec}, \mathsf{Combine})$*, defined as follows:*

- $\mathsf{Setup}(1^\lambda, t, n)$*: The setup algorithm proceeds as follows:*
  1. *Run a group generation algorithm $\mathcal{G}(1^\lambda)$ to obtain (q, g), where $\mathbb{G}_q$ is a cyclic group of prime order q with generator g.*
  2. *Sample a secret key $x \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random.*
  3. *Compute the public key component $h = g^x$.*
  4. *Use a (t, n)-threshold secret sharing scheme to split x into n shares $\{s_1, s_2, \ldots, s_n\}$, where each share $s_i = f(i) \pmod q$, with $f(z)$ being a random polynomial of degree $t-1$ such that $f(0) = x$.*
  5. *Output the public key $\mathsf{pk} = (q, g, h)$ and private shares $\{\mathsf{sk}_i = s_i\}_{i=1}^n$.*
- $\mathsf{Enc}(\mathsf{pk}, M)$*: To encrypt a message $M \in \mathbb{G}_q$, the encryption algorithm proceeds as follows:*
  1. *Sample a random $r \xleftarrow{\$} \mathbb{Z}_q$.*
  2. *Compute the ciphertext components: $u = g^r, \quad v = h^r \cdot M$.*
  3. *Output the ciphertext $\psi = (u, v)$.*
- $\mathsf{PartialDec}(\mathsf{sk}_i, \psi)$*: Given a private share $\mathsf{sk}_i = s_i$ and ciphertext $\psi = (u, v)$, a party computes its partial decryption: $d_i = u^{s_i}$. Output the partial decryption d_i.*
- $\mathsf{Combine}(\{d_i\}_{i\in T}, \psi)$*: Given partial decryptions $\{d_i\}_{i\in T}$ from a subset $T \subseteq [n]$ of size $|T| \geq t$, combine them as follows:*
  1. *Compute the Lagrange coefficients $\{\lambda_i\}_{i\in T}$ over $\mathbb{Z}_q$, where $\lambda_i = \prod_{\substack{j\in T \\ j\neq i}} \frac{j}{j-i} \pmod q$.*
  2. *Compute the shared decryption: $s = \prod_{i\in T} d_i^{\lambda_i} \pmod q$.*
  3. *Recover the plaintext message M as: $M = v \cdot s^{-1}$.*

(Our) threshold ElGamal encryption scheme (extended). Using a distributed key generation protocols (e.g., [36,42,40,26]), the secret key of the threshold ElGamal encryption scheme is generated. We denote sk_j as the secret decryption key for the j-th maintainer, and $\mathsf{pk}_j = g^{\mathsf{sk}_j}$ as the corresponding public key share. Hence, the aggregated secret key is $\mathsf{sk} = \sum_{j\in I} \mathsf{sk}_j \lambda_j$, where λ_j is the Lagrange coefficient for the j-th share, and $|I| = \beta$.

To decrypt an ElGamal ciphertext $c = (c_1, c_2) = (g^k, \mathsf{pk}^k m)$, the j-th maintainer first publishes $c_1^{\mathsf{sk}_j}$ and then generates a proof that $\log_g \mathsf{pk}_j = \log_{c_1} c_1^{\mathsf{sk}_j}$ to prove their honest contribution. Finally, the plaintext m is retrieved as $m = \frac{c_2}{\prod_{j\in I} c_1^{\mathsf{sk}_j \lambda_j}}$.

In our construction, assuming the first message is the user's public key $m_1 = \mathsf{pk}_\mathsf{U}$ and the second message is $m_2 = g^v$, where v is the value of the transaction, we extend the ciphertext such that

$$c = (c_1, c_2, c_3) = (g^k, \mathsf{pk}_{1,\mathbb{M}}^k \cdot m_1, \mathsf{pk}_{2,\mathbb{M}}^k \cdot m_2).$$

Here, the j-th maintainer has two secret keys, $\mathsf{sk}_{1,j}$ and $\mathsf{sk}_{2,j}$, such that $\mathsf{pk}_{1,j} = g^{\mathsf{sk}_{1,j}}$ and $\mathsf{pk}_{2,j} = g^{\mathsf{sk}_{2,j}}$. Similarly, the aggregated secret keys are $\mathsf{sk}_{1,\mathbb{M}} = \sum_{j\in I} \mathsf{sk}_{1,j}\lambda_{1,j}$ and $\mathsf{sk}_{2,\mathbb{M}} = \sum_{j\in I} \mathsf{sk}_{2,j}\lambda_{2,j}$, with their associated public keys being $\mathsf{pk}_{1,\mathbb{M}} = g^{\mathsf{sk}_{1,\mathbb{M}}}$ and $\mathsf{pk}_{2,\mathbb{M}} = g^{\mathsf{sk}_{2,\mathbb{M}}}$, respectively.

The j-th maintainer's decryption shares are $c_1^{\mathsf{sk}_{1,j}}$ and $c_1^{\mathsf{sk}_{2,j}}$. The maintainer also generates proofs that $\log_g \mathsf{pk}_{1,j} = \log_{c_1} c_1^{\mathsf{sk}_{1,j}}$ and $\log_g \mathsf{pk}_{2,j} = \log_{c_1} c_1^{\mathsf{sk}_{2,j}}$ hold to demonstrate their honest contribution.

The messages m_1 and m_2 are retrieved by computing

$$m_1 = \frac{c_2}{\prod_{j\in I} c_1^{\mathsf{sk}_{1,j}\lambda_{1,j}}} \quad \text{and} \quad m_2 = \frac{c_3}{\prod_{j\in I} c_1^{\mathsf{sk}_{2,j}\lambda_{2,j}}}.$$

B.2 Shamir secret sharing

Shamir's secret sharing scheme [58] allows a secret s to be divided into n shares such that any subset of at least t shares can reconstruct the secret s. However, fewer than t shares reveal no information about the secret.

Definition 12 (Shamir secret sharing (SSS)). *A (n, t)-threshold secret sharing scheme* $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ *consists of the following algorithms:*

- ***Sharing:*** $\{(x_i, y_i)\}_{i=1}^{n} \xleftarrow{\$} \mathsf{Share}^{n,t}(s)$ *This algorithm takes as input a secret $s \in \mathbb{F}_q$, where $\mathbb{F}_q$ is a finite field of prime order q, and outputs n secret shares (x_i, y_i), where $x_i \neq 0$ are distinct and $y_i = P(x_i)$. The steps are as follows:*
  1. *Choose $t-1$ random coefficients $a_1, \ldots, a_{t-1} \in \mathbb{F}_q$.*
  2. *Construct the polynomial $P(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$.*
  3. *Compute each share as a pair (x_i, y_i), where $y_i = P(x_i)$ and $x_i \neq 0$ are distinct.*
- ***Reconstruction:*** $s^* \leftarrow \mathsf{Reconstruct}^{n,t}(\{(x_i, y_i)\}_{i \in \mathcal{I}})$ *This algorithm takes as input a set of t shares $\{(x_i, y_i)\}_{i \in \mathcal{I}}$, where $|\mathcal{I}| = t$, and reconstructs the secret s using Lagrange interpolation: $s = \sum_{i \in \mathcal{I}} y_i \cdot \lambda_i$, where the Lagrange coefficients λ_i are defined as: $\lambda_i = \prod_{j \in \mathcal{I}, j \neq i} \frac{x_j}{x_j - x_i}$.*

Definition 13 (Correctness of Shamir secret sharing). *A Shamir secret sharing scheme* SSS *is correct if, for any secret $s \in \mathbb{F}_q$, any set of t or more shares $\{(x_i, y_i)\}_{i \in \mathcal{I}}$ can be used to reconstruct s with probability 1. Formally:* $\Pr[s^* = s] = 1$, *where s^* is the output of the reconstruction algorithm* $\mathsf{Reconstruct}^{n,t}$.

Definition 14 (Privacy of Shamir secret sharing). *A Shamir secret sharing scheme* SSS *satisfies privacy if, given fewer than t shares, no probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ can gain any advantage in distinguishing between two secrets $s_0, s_1 \in \mathbb{F}_q$ shared using* $\mathsf{Share}^{n,t}$.

B.3 Commitment schemes

Definition 15 (Commitment scheme). *A commitment scheme* $\mathsf{COM} = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Verify})$ *consists of the following algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$***:*** *A probabilistic algorithm that takes as input a security parameter λ and outputs a pair of keys* $(\mathsf{ck}, \mathsf{vk})$, *where:*
  - ck*: Commitment key, used by the committer to generate commitments.*
  - vk*: Verification key, used by the verifier to verify commitments.*

  In publicly verifiable schemes, it holds that $\mathsf{ck} = \mathsf{vk}$.
- $\mathsf{Commit}(\mathsf{ck}, m)$***:*** *A probabilistic algorithm that takes as input a commitment key* ck *and a message $m \in \mathsf{MessageSpace}(\mathsf{ck})$. It outputs a commitment c and an opening d:*

  $$(c, d) \leftarrow \mathsf{Commit}(\mathsf{ck}, m).$$

  The pair (c, d) enables the committer to prove the commitment corresponds to the message m in the verification phase.
- $\mathsf{Verify}(\mathsf{vk}, c, m, d)$***:*** *A deterministic algorithm that takes as input the verification key* vk, *a commitment c, a message m, and an opening d. It outputs* accept *if (c, d) correctly reconstructs m and satisfies the consistency of the commitment; otherwise, it outputs* reject*:*

  $$\mathsf{Verify}(\mathsf{vk}, c, m, d) = \mathsf{accept} \text{ if } (c, d) \text{ is valid for } m, \text{ and } \mathsf{reject} \text{ otherwise.}$$

Definition 16 (Commitment correctness). *A commitment scheme* $\mathsf{COM} = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Verify})$ *satisfies correctness if, for any security parameter* λ*, any valid message* $m \in \mathsf{MessageSpace}(\mathsf{ck})$*, and any keys* $(\mathsf{ck}, \mathsf{vk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*, the following holds:*

$$\mathsf{Verify}(\mathsf{vk}, c, m, d) = \mathsf{accept},$$

with probability 1 over the randomness of Commit*, where* $(c, d) \leftarrow \mathsf{Commit}(\mathsf{ck}, m)$*.*
In other words, when both the committer and verifier behave honestly, any valid commitment c *corresponding to a message* m *will always be accepted during verification.*

Definition 17 (Commitment hiding). *Let* $\mathsf{COM} = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Verify})$ *denote a (publicly verifiable) commitment scheme. The hiding property ensures that no probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *can distinguish between commitments of two messages. The hiding experiment* $\mathsf{Hide}^b_{\mathsf{COM}}(\mathcal{A}, \lambda)$*, parameterized by a security parameter* λ *and a bit* $b \in \{0, 1\}$*, proceeds as follows:*

1. *The challenger generates a commitment key* $\mathsf{ck} = \mathsf{vk}$ *by running* $\mathsf{KeyGen}(1^\lambda)$ *and sends* ck *to the adversary* $\mathcal{A}$*.*
2. *The adversary* $\mathcal{A}$ *submits two messages* (m_0, m_1)*, such that* $|m_0| = |m_1|$ *and* $m_0, m_1 \in \mathsf{MessageSpace}(\mathsf{ck})$*.*
3. *The challenger selects a random bit* $b \xleftarrow{\$} \{0, 1\}$*, computes the commitment* $(c, d) \leftarrow \mathsf{Commit}(\mathsf{ck}, m_b)$*, and sends the commitment* c *to the adversary* $\mathcal{A}$*.*
4. *The adversary* $\mathcal{A}$ *outputs a bit* b' *as its guess for* b*.*

The adversary's advantage in this experiment is defined as:

$$\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) = \left| \Pr[\mathsf{Hide}^1_{\mathsf{COM}}(\mathcal{A}, \lambda) = 1] - \Pr[\mathsf{Hide}^0_{\mathsf{COM}}(\mathcal{A}, \lambda) = 1] \right|.$$

A commitment scheme satisfies the hiding property if:

- *Perfect hiding: For all adversaries* $\mathcal{A}$*, it holds that:*

$$\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) = 0.$$

- *Computational hiding: For all PPT adversaries* $\mathcal{A}$*, it holds that:*

$$\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda),$$

  where $\mathsf{negl}(\lambda)$ *is a negligible function in the security parameter* λ*.*

Hiding ensures that no adversary can distinguish the commitment of m_0 *from* m_1 *with a probability significantly better than a random guess.*

Definition 18 (Commitment binding). *Let* $\mathsf{COM} = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Verify})$ *denote a (publicly verifiable) commitment scheme. The binding property ensures that no probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *can produce a valid commitment that can be opened to two different messages.*
The binding experiment $\mathsf{Bind}_{\mathsf{COM}}(\mathcal{A}, \lambda)$*, parameterized by a security parameter* λ*, proceeds as follows:*

1. *The challenger generates a commitment key* $\mathsf{ck} = \mathsf{vk}$ *by running* $\mathsf{KeyGen}(1^\lambda)$ *and sends* ck *to the adversary* $\mathcal{A}$*.*
2. *The adversary* $\mathcal{A}$ *outputs a tuple* (c, m, d, m', d')*, where* $m, m' \in \mathsf{MessageSpace}(\mathsf{ck})$*.*
3. *The challenger checks the following conditions:*

$$\mathsf{Verify}(\mathsf{vk}, c, m, d) = \mathsf{accept} \quad \textit{and} \quad \mathsf{Verify}(\mathsf{vk}, c, m', d') = \mathsf{accept}.$$

4. *If both conditions hold and* $m \neq m'$*, the adversary* $\mathcal{A}$ *wins the binding game. Otherwise, the adversary fails.*

The adversary's advantage in this experiment is defined as:

$$\mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}}(\lambda) = \mathsf{Pr}[\mathsf{Bind}_{\mathsf{COM}}(\mathcal{A}, \lambda) = 1].$$

A commitment scheme satisfies the binding property if:

- *Perfect binding: For all adversaries $\mathcal{A}$, it holds that:*

$$\mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}}(\lambda) = 0.$$

- *Computational binding: For all PPT adversaries $\mathcal{A}$, it holds that:*

$$\mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda),$$

  where $\mathsf{negl}(\lambda)$ is a negligible function in the security parameter λ.

Binding ensures that a commitment c uniquely binds the committer to a single message m. No adversary should be able to produce two distinct valid openings (m, d) and (m', d') for the same commitment c, except with negligible probability.

Definition 19 (Pedersen commitment scheme). *The Pedersen commitment scheme [52]* $\mathsf{PCOM} = (\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Verify})$ *is defined as follows:*

- $\mathsf{KeyGen}(1^\lambda)$*: A probabilistic algorithm that, given a security parameter λ, generates:*
  - *G: A cyclic group of prime order q,*
  - *g: A generator of G,*
  - *$h \xleftarrow{\$} G$: A random independent group element.*

  The output is the commitment key ck and the verification key vk: $\mathsf{ck} = \mathsf{vk} = (G, q, g, h)$.
- $\mathsf{Commit}(\mathsf{ck}, m)$*: A probabilistic algorithm that, given the commitment key $\mathsf{ck} = (G, q, g, h)$ and a message $m \in \mathbb{Z}_q$:*
  - *Samples a random opening value $d \xleftarrow{\$} \mathbb{Z}_q$,*
  - *Computes the commitment: $c = g^d \cdot h^m$.*

  The output is the commitment pair (c, d): $(c, d) \leftarrow \mathsf{Commit}(\mathsf{ck}, m)$.
- $\mathsf{Verify}(\mathsf{vk}, c, m, d)$*: A deterministic algorithm that, given the verification key $\mathsf{vk} = (G, q, g, h)$, a commitment $c \in G$, a message $m \in \mathbb{Z}_q$, and an opening value $d \in \mathbb{Z}_q$:*
  - *Verifies whether: $g^d \cdot h^m = c$.*
  - *Outputs* accept *if the equality holds, and* reject *otherwise:*

$$\mathsf{Verify}(\mathsf{vk}, c, m, d) = \mathsf{accept} \text{ if } g^d \cdot h^m = c, \text{ and } \mathsf{reject} \text{ otherwise.}$$

The Pedersen commitment scheme PCOM *satisfies the following properties [49,48]:*

- ***Perfect hiding:*** *The commitment c reveals no information about the message m, as h^m is masked by the randomness g^d.*
- ***Computational binding:*** *Given a valid commitment c, it is computationally infeasible to find distinct (m, d) and (m', d') such that: $c = g^d \cdot h^m = g^{d'} \cdot h^{m'}$.*

C Ideal functionalities

The description of the ideal functionalities used in our protocol is as follows.

C.1 Key registration functionality

The ideal functionality $\mathcal{F}_{\mathsf{KR}}$ models public key registration and public key/identity retrieval mechanism. In the *register* phase, a user U can register a public key pk uniquely linked to their identity U, provided they know the associated secret key sk, they have not registered a public key pk before or there does not exist an identity with pk already registered. The registration request waits for the adversary $\mathcal{A}$'s approval. Upon approval, the public key-identity pair is stored, and U is notified. In the *retrieve* phase, a user U^* can retrieve U's registered public key, or U^* can request an identifier for a specific pk. Retrieval involves interaction with $\mathcal{A}$, and the response depends on whether the public key or identity exists.

Functionality $\mathcal{F}_{\mathsf{KR}}$

The functionality $\mathcal{F}_{\mathsf{KR}}$ is parameterized by a secret key–public key relation $\mathcal{R}(\mathsf{sk}, \mathsf{pk})$.

- **Register.** Upon input $(\texttt{Register}, \mathsf{sid}, \mathsf{sk}, \mathsf{pk})$ from U, ignore if $\mathcal{R}(\mathsf{sk}, \mathsf{pk})$ does not hold, or there exists an entry $(\cdot, \mathsf{pk})$ or $(\mathsf{U}, \cdot)$. Else, output $(\texttt{Register}, \mathsf{sid}, \mathsf{U}, \mathsf{pk})$ to $\mathcal{A}$. Upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{U}, \mathsf{pk})$ from $\mathcal{A}$, record the pair $(\mathsf{U}, \mathsf{pk})$, and output $(\texttt{Registered}, \mathsf{sid})$ to U.
- **Retrieve key.** Upon input $(\texttt{RetrieveKey}, \mathsf{sid}, \mathsf{U})$ from U^*, sample a fresh q and set $L(q) \leftarrow (\mathsf{U}, \mathsf{U}^*)$. Output $(\texttt{RetrieveKey}, \mathsf{sid}, q)$ to $\mathcal{A}$. Upon receiving $(\texttt{Ok}, \mathsf{sid}, q)$ from $\mathcal{A}$, ignore if $L(q) = \perp$. Else, retrieve $L(q) = (\mathsf{U}, \mathsf{U}^*)$. If there exists a recorded pair $(\mathsf{U}, \mathsf{pk})$, output $(\texttt{KeyRetrieved}, \mathsf{sid}, \mathsf{U}, \mathsf{pk})$, else, output $(\texttt{KeyRetrieved}, \mathsf{sid}, \mathsf{U}, \perp)$ to U^* via private-delayed output.
- **Retrieve ID.** Upon input $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{pk})$ from U^*, sample a fresh r and set $L(r) \leftarrow (\mathsf{pk}, \mathsf{U}^*)$. Output $(\texttt{RetrieveID}, \mathsf{sid}, r)$ to $\mathcal{A}$. Upon receiving $(\texttt{Ok}, \mathsf{sid}, r)$ from $\mathcal{A}$, ignore if $L(r) = \perp$. Else, retrieve $L(r) = (\mathsf{pk}, \mathsf{U}^*)$. If there exists a recorded pair $(\mathsf{U}, \mathsf{pk})$, output $(\texttt{IDRetrieved}, \mathsf{sid}, \mathsf{U}, \mathsf{pk})$ else, output $(\texttt{IDRetrieved}, \mathsf{sid}, \perp, \mathsf{pk})$ to U^* via private-delayed output.

C.2 Communication channel functionality

For privacy-preserving requirements, $\mathcal{F}_{\mathsf{CBDC}}$ does not leak the identities of users. To realize this functionality, our protocol uses different types of communication channels $\mathcal{F}_{\mathsf{Ch}}$ to deliver messages and to meet network-level anonymity (e.g., preventing traffic analysis attacks and extracting identities) see [33] for more information.

Functionality $\mathcal{F}_{\mathsf{Ch}}$

Let define S and R as the sender and receiver of a message m respectively. Δ is defined as follows based on parameters of functionality. Message identifier mid is selected freshly by the functionality.

1. Upon input $(\texttt{Send}, \mathsf{sid}, R, m)$ from S, sample a fresh mid and record $L(\mathsf{mid}) = (S, R, m)$. Output $(\texttt{Send}, \mathsf{sid}, \Delta, \mathsf{mid})$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, ignore if $L(\mathsf{mid}) = \perp$. Else, retrieve $L(\mathsf{mid}) = (S, R, m)$. Send $(\texttt{Received}, \mathsf{sid}, S, m)$ to R.

Set Δ based on the following parameterized functions:

- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ac}}$ set $\Delta = (S, R, m)$. Upon receiving $(\texttt{Ok.Snd}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, send $(\texttt{Continue}, \mathsf{sid})$ to S.[a]
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sra}}$ set $\Delta = (S, |m|)$.
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{ssa}}$ set $\Delta = (R, |m|)$.
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{fa}}$ set $\Delta = |m|$.
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sc}}$ set $\Delta = (S, R, |m|)$. Upon receiving $(\texttt{Ok.Snd}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, send $(\texttt{Continue}, \mathsf{sid})$ to S.
- for $\mathcal{F}_{\mathsf{Ch}}^{\mathsf{sa}}$ set $\Delta = (R, m)$.
  - upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, send $(\texttt{Received}, \mathsf{sid}, m, \mathsf{mid})$ to R. Upon receiving $(\texttt{Ok.Snd}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, send $(\texttt{Continue}, \mathsf{sid})$ to S.
  - upon receiving $(\texttt{Send}, \mathsf{sid}, \mathsf{mid}, m')$ from R, output $(\texttt{Send}, \mathsf{sid}, R, m', \mathsf{mid})$ to $\mathcal{A}$. Upon receiving $(\texttt{Ok.End}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, send $(\texttt{Received}, \mathsf{sid}, R, m')$ to S.

[a] This gives more power to adversary $\mathcal{A}$ who decides when the sender can proceed as sequential message sending is required in the UC model.

C.3 Asynchronous byzantine agreement functionality

In asynchronous byzantine agreement (BA) protocol a set of D parties agree on their inputs, even facing malicious corruptions. The following functionality tolerates a malicious adversary who statically corrupts up to t parties. In our protocol Π_{PEReDi} that tolerates static malicious adversaries, we use the following asynchronous Byzantine agreement functionality $\mathcal{F}_{\mathsf{aBA}}$ which can be implemented as follows. Parties send their inputs to everyone. Once all $D-t$ inputs are received, an honest party switches its input to the bit that has the majority among $D-t$. Afterwards, parties engage in a standard Byzantine agreement protocol [16].

Functionality $\mathcal{F}^{c}_{\mathsf{aBA}}$

Running with $\mathbb{M} = \{\mathsf{M}_1, ..., \mathsf{M}_D\}$ parties; Byzantine agreement functionality $\mathcal{F}_{\mathsf{aBA}}$ proceeds as follows where initially $Q \leftarrow \perp$:

- Upon receiving $(\mathtt{Agree}, \mathsf{sid}, d_j)$ from M_j where $d_j \in \{0,1\}$, record $(\mathtt{Agree}, \mathsf{sid}, d_j, \mathsf{M}_j)$ and send $(\mathtt{Agree}, \mathsf{sid}, d_j, \mathsf{M}_j)$ to $\mathcal{A}$. Upon receiving $D-t$ distinct d_j values: Set $Q = d_j$ once $|\{j|d_j\}| \geq c \cdot t + 1$.
- Upon receiving $(\mathtt{Agree.Ok}, \mathsf{sid})$ from $\mathcal{A}$: If $Q \neq \perp$ output $(\mathtt{Agreed}, \mathsf{sid}, Q)$ to every M_j via public-delayed output. Else, ignore.

C.4 Broadcast functionality

The broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ [38] models a reliable broadcast channel where a sender P can transmit a message m to a predefined set of recipients $\mathbb{M} = \{\mathsf{M}_1, \ldots, \mathsf{M}_D\}$, ensuring that all recipients receive the identical message. Upon receiving $(\mathtt{Broadcast}, \mathsf{sid}, m)$ from P, the functionality delivers $(\mathtt{Broadcasted}, \mathsf{sid}, \mathsf{P}, m)$ to all parties in $\mathbb{M}$ and the adversary $\mathcal{A}$, ensuring consistency and eventual delivery but providing no secrecy guarantee for m.

Functionality $\mathcal{F}_{\mathsf{BC}}$

Broadcast functionality $\mathcal{F}_{\mathsf{BC}}$ parameterized by the set $\mathbb{M} = \{\mathsf{M}_1, ..., \mathsf{M}_D\}$ proceeds as follows: Upon receiving $(\mathtt{Broadcast}, \mathsf{sid}, m)$ from a party P, send $(\mathtt{Broadcasted}, \mathsf{sid}, \mathsf{P}, m)$ to all entities in the set $\mathbb{M}$ and to $\mathcal{A}$.

C.5 Random oracle functionality

The random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ [19] models an idealized cryptographic hash function that provides truly random outputs while ensuring consistency across repeated queries. It is parameterized by a message space M and an output space Y. Upon receiving a query $(\mathtt{Query}, \mathsf{sid}, m)$ from a party P, the functionality first checks whether $m \in M$; if not, it aborts. Otherwise, if m has not been queried before, it selects a fresh random value $h \in Y$, ensuring that no previous query maps to the same output, and stores the mapping (sid, m, h). For repeated queries on the same input m, the functionality returns the previously stored value. This models an idealized hash function that behaves like a truly random function but remains deterministic for repeated inputs.

Functionality $\mathcal{F}_{\mathsf{RO}}$

The functionality is parameterized by an output space Y and a message space M. Upon receiving $(\mathtt{Query}, \mathsf{sid}, m)$ from a party P:

- Abort if $m \notin M$.
- Else, if a tuple (sid, m', h) where $m' = m$ has not already been stored, select a random h from Y where there is no stored tuple (sid, m^*, h') where $h' = h$, then store (sid, m, h).
- Take the stored tuple (sid, m', h) where $m' = m$ and output $(\mathtt{Query.Re}, \mathsf{sid}, h)$ to party P.

C.6 Non-interactive zero knowledge functionality

Groth et al. [41] formalized ideal functionality of non-interactive zero-knowledge (NIZK) that was introduced by Blum et al. [15]. The NIZK functionality $\mathcal{F}_{\mathsf{NIZK}}$ models a proof system where a prover can convince any verifier of the validity of a statement without interaction. Unlike interactive zero-knowledge proofs, $\mathcal{F}_{\mathsf{NIZK}}$ allows publicly verifiable proofs without pre-specifying the verifier.

Functionality $\mathcal{F}_{\mathsf{NIZK}}$

The functionality is parameterized by a relation $\mathcal{R}$.

- **Proof.** On receiving $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$ from U, ignore if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$. Else, send $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$. Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, store $(\mathtt{x}, \pi)$ and send $(\mathtt{Proof}, \mathsf{sid}, \pi)$ to U.
- **Verify.** Upon receiving $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ from U check whether $(\mathtt{x}, \pi)$ is stored. If not send $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$. Upon receiving the answer $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, check $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ and if so, store $(\mathtt{x}, \pi)$. If $(\mathtt{x}, \pi)$ has been stored, output $(\mathtt{Verification}, \mathsf{sid}, 1)$ to U, else output $(\mathtt{Verification}, \mathsf{sid}, 0)$.

C.7 Signature of knowledge functionality

Signature of knowledge (SoK) was first formally defined by Chase et al. [28]. In SoK, by providing a valid signature, the signer proves the possession of a witness $\mathtt{w}$ to a statement $\mathtt{x}$ for a relation $\mathcal{R}$. It generalizes the notion of traditional signature where a signature under a public key serves as a proof that the signer is in possession of the corresponding secret key.

Functionality $\mathcal{F}_{\mathsf{SoK}}$

The functionality is parameterized by a relation $\mathcal{R}$. Moreover, $\mathsf{Sign}, \mathsf{Simsign}$ and $\mathsf{Extract}$ are descriptions of PPT TMs, and Verify is a description of a deterministic polytime TM.

- **Setup**. Upon receiving $(\mathtt{Setup}, \mathsf{sid})$ from U if this is the first time that $(\mathtt{Setup}, \mathsf{sid})$ is received, send $(\mathtt{Setup}, \mathsf{sid})$ to $\mathcal{A}$; upon receiving $(\mathtt{Algorithms}, \mathsf{sid}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Simsign}, \mathsf{Extract})$ from $\mathcal{A}$, store these algorithms. Output the stored $(\mathtt{Algorithms}, \mathsf{sid}, \mathsf{Sign}, \mathsf{Verify})$ to U.

- **Signature Generation.** Upon receiving $(\texttt{Sign}, \mathsf{sid}, m, \mathtt{x}, \mathtt{w})$ from U, if $(\mathtt{x}, \mathtt{w}) \notin \mathcal{R}$ ignore. Else, compute $\sigma \leftarrow \mathsf{Simsign}(m, \mathtt{x})$, and check that $\mathsf{Verify}(m, \mathtt{x}, \sigma) = 1$. If so, then output $(\texttt{Signature}, \mathsf{sid}, m, \mathtt{x}, \sigma)$ to U and record the entry $(m, \mathtt{x}, \sigma)$. Else, output an error message (Completeness error) to U and halt.
- **Signature Verification.** Upon receiving $(\texttt{Verify}, \mathsf{sid}, m, \mathtt{x}, \sigma)$ from U_j, if $(m, \mathtt{x}, \sigma')$ is stored for some σ', then output $(\texttt{Verified}, \mathsf{sid}, m, \mathtt{x}, \sigma, \mathsf{Verify}(m, \mathtt{x}, \sigma))$ to U_j. Else let $\mathtt{w} \leftarrow \mathsf{Extract}(m, \mathtt{x}, \sigma)$; if $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$, output $(\texttt{Verified}, \mathsf{sid}, m, \mathtt{x}, \sigma, \mathsf{Verify}(m, \mathtt{x}, \sigma))$ to U_j. Else if $\mathsf{Verify}(m, \mathtt{x}, \sigma) = 0$, output $(\texttt{Verified}, \mathsf{sid}, m, \mathtt{x}, \sigma, 0)$ to U_j. Else output an error message (Unforgeability error) to U_j and halt.