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

Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh*

University of Edinburgh and IOHK, UK
akiayias@inf.ed.ac.uk, mkohlwei@ed.ac.uk, and amirreza.sarencheh@ed.ac.uk

Abstract. Central Bank Digital Currencies (CBDCs) aspire to offer a digital replacement for physical cash and as such need to tackle two fundamental requirements that are in conflict. On the one hand, it is desired they are *private* so that a financial "panopticon" is avoided, while on the other, they should be *regulation friendly* in the sense of facilitating any threshold-limiting, tracing, and counterparty auditing functionality that is necessary to comply with regulations such as Know Your Customer (KYC), Anti Money Laundering (AML) and Combating Financing of Terrorism (CFT) as well as financial stability considerations. In this work, we put forth a new model for CBDCs and an efficient construction that, for the first time, fully addresses these issues simultaneously. Moreover, recognizing the importance of avoiding a *single point of failure*, our construction is distributed so that all its properties can withstand a suitably bounded minority of participating entities getting corrupted by an adversary. Achieving all the above properties efficiently is technically involved; among others, our construction uses suitable cryptographic tools to thwart man-in-the-middle attacks, it showcases a novel traceability mechanism with significant performance gains compared to 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 communication overhead when the sender and receiver are honest. Going beyond "simple" payments, we also discuss how our scheme can facilitate one-off large transfers complying with Know Your Transaction (KYT) disclosure requirements. Our CBDC concept is expressed and realized in the Universal Composition (UC) framework providing in this way a modular and secure way to embed it within a larger financial ecosystem.

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

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 [30] 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) [11] and has a different operational structure than other forms of central bank money [40]. 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 [24] which was followed by designs explored by a number of central banks [1,22,10]. 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

* Corresponding author.

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., ZCash [8] or Monero [34]) 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. [4].

Privacy in payment systems can interfere with three main regulatory obligations:

1. Know-Your-Customer (KYC), which requires the positive identification of counterparties before they are able to transact.
2. Anti-Money Laundering (AML), which requires that sources of funds should be legitimate.
3. 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. [1,7,10], 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.

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. 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:

1. To the best of our knowledge, this is the first time that a fully privacy-preserving and comprehensively regulated CBDC is modeled formally. Our formal model is in the Universal Composition (UC) setting [17]. This modeling enables the composition of the system as payment infrastructure within larger systems.
2. We review the regulatory compliance in the context of payment systems (KYC, AML, CFT, auditing, etc.) and argue how our ideal functionality for CBDCs captures such requirements.
3. We put forth a distributed construction that realizes our CBDC ideal functionality in an efficient manner based on standard cryptographic assumptions. Notably our construction demonstrates that Byzantine broadcast or agreement is not 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.
4. 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 context of tracing users in conditionally anonymous payment systems.
5. We describe how our distributed CBDC construction 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.

1.1 Related Work

The first system for anonymous electronic cash was introduced by Chaum [21] 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 that they possess, a performance consideration that was addressed in follow up work [19,14].

Regarding the problem of revealing the transaction value to the bank, transferable e-cash [16,6] 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. [15] proposed a token-based e-payment solution in which the bank can enforce simple rules such as per-user payment limits. Privacy of senders of transactions is preserved, nonetheless, the recipient identity and payment amount are leaked.

Considering blockchain-ledger-based anonymous payment systems like Zerocash [8], Garman et al. [27] addressed how regulation rules could be enforced in such constructions. 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 should 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. Moreover, users are supposed to download the whole ledger and decrypt all transactions to conclude 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 (this technique eliminates the need to synchronize with the ledger state, which is only necessary for auditing).

Danezis et al., [24] introduced RSCoin, a central bank currency framework which is built around an efficient broadcast mechanism. In RSCoin, the central bank delegates the responsibility of verifying transactions to a set of entities called mintettes. Different from 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.

Wüst et al. [39] 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 meet full anonymity as validators can link different transactions and it does not have auditability. Hence, the authorities cannot investigate suspicious transactions or counterparties on demand.

Androulaki et al. [5] 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 they do not offer a comprehensive approach to regulatory compliance as we do.

Damgård et al.'s work [23] 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 [23], for each account generation, requires the account holder to compute a pseudorandom-function PRF using its secret key. There is no concrete implementation for tracing in their work as they use a secure multi-party computation for PRF in a black-box manner. More importantly, the input of PRF is only restricted to be in 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. [38] introduced Platypus which is a privacy preserving and centralized payment system. Platypus relies on a single authority, our scheme is distributed such that it is robust against single points of failure with respect to regulation enforcement, and can work even if the central bank is completely offline. Furthermore, our scheme offers encrypted (distributed) ledgers which allow compliance with regulation like AML and CFT, by enabling the set of authorities to trace a malicious user and to discover the transfer value and identities of the counterparties in any suspicious transaction. Platypus [38] does not offer such 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 take place and should be addressed by 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 something which may limit the composability of their construction, cf. [18]. Finally, another drawback of Platypus [38] is that the technical details on their regulation approach, currency issuance by the central bank, and addressing concurrent and aborted transactions are not formally studied within their security model.

Tomescu et al.[37] introduced a decentralized payment system called UTT. Their construction rely on Byzantine fault tolerant infrastructure. However, PEReDi obviates Byzantine agreement and byzantine broadcast from the optimistic execution path of a transaction. Hence, we have 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 be able 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 UTT setting is limited by a monthly budget. PEReDi, on the other hand, allows for comprehensive regulatory compliance, and can also enforce them from the recipient's standpoint.

2 CBDC Security Desiderata and Modeling

2.1 CBDC Entities

We abstract a CBDC system to 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 are as follows.

1. *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 [22], 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 [10], and [22] for more context on the role of central banks.
2. *Maintainers:* The authority of validating transactions and facilitating 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 needed to be active in any of the system's day to day operations (except for issuing currency for the former). The maintainers share the state of 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 for instance, commercial banks, financial institutions, and 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 the 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, there exist various operations that are subject to regulatory rules. Examples of relevant

entities developing and/or enforcing such rules are the Financial Conduct Authority (FCA) in the UK or 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 that are introduced subject to the approval of the maintainers.

3. *Users and Payment interface Providers (PIPs):* As any digital currency system, in a CBDC system, the 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 of the system are untrusted, i.e. they may behave maliciously against honest users or other system entities. Privacy of payments should be satisfied between an honest sender and an honest receiver in a transaction.

2.2 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 which are more specific to payment systems and CBDCs; these are as follows.

1. *Financial and Regulatory Integrity.* No one should be able to update the account of another user. Furthermore, currency in circulation or the amount of CBDC that is 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 account of both of them should be updated correctly considering all parameters that are included in users' accounts for the purpose of checking financial and regulatory rules.
2. *Comprehensive Regulatory Compliance.* This term means achieving all the following four items at the same time.
   (a) Balance Limit: It limits the amount of funds that a particular user can possess in a specific period of time. Bank of England [1] and a report from several Central Banks (that details the principles, motivations, and risks of CBDC) [10] have mentioned that balance limit can help prevent bank runs and evasion of wealth tax. Moreover, the Bank of England (BoE) [1] and the European central bank (ECB) [9] have addressed that to manage the implications of a CBDC for financial stability, limits of how much CBDC any individual can hold is necessary.
   (b) Receiving and Sending Limit: It limits the amount of received and sent funds that a particular user can receive or send in a specific period of time. The sent and received amounts should not exceed a predefined threshold. European central bank [7], and several central banks [10] have mentioned that limiting receiving and sending values can help achieve AML and prevent tax evasion.
   (c) Transaction Value Limit and KYT: Reporting requirements and disclosure of 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 have 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.

   (d) Auditability: In cases of suspicious activities, additional auditing actions are needed (for e.g., filing suspicious activity reports called SARs [2]). The auditing functionality has two components:
      i. Privacy Revocation: Given an anonymous transaction, authorities can reveal the real world identities of involved parties and the transferred value of that transaction.
      ii. Tracing: Given a real world identity of a user, authorities can trace anonymous payments in which the user has engaged (as a sender or recipient).
3. *Full Privacy.* This property means achieving all the following three items at the same time.
   (a) Identity Privacy: It means 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 involved as a sender or receiver.
   (b) Transaction Privacy: The transferred value by the sender to the recipient cannot be revealed (except when auditing) and 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., sum of all sent and received values) are hidden from all network entities.
   (c) Full Unlinkability: It contains two parts that are as follows.
      i. User Unlinkability: Given an anonymous payment's real world identities of the sender or receiver it should not be possible to link the sender or receiver's other transactions to the given transaction.
      ii. 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.
4. *Accountability.* When a user makes a payment it should not be able to deny it later — there is an obligation to accept the responsibilities that come with a finalized transaction.

2.3 Notations

In this paper, for uniquely identifying parties, we denote the central bank by B, the user and its key pair with U and $(\mathsf{pk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}})$ respectively. U also has another secret key a used for generating *per-transaction* tracing tag. This tag is denoted by T. We denote the account of U by acc. The notation M_j is used for the j-th maintainer and $\mathbb{M}$ for the set of all maintainers. We assume $|\mathbb{M}| = D$ and there are two thresholds, α is the threshold number of maintainers required for verifying transactions on behalf of the central bank and the regulator, and β is the threshold number of maintainers required for executing the *Auditing* protocol. Maintainers of which β number is required for executing the *Auditing* protocol is called *audit committee.* Set of honest and malicious maintainers are denoted by $\mathbf{H}$ and $\mathbf{C}$, and their associated identifiers (indexes) by $\mathcal{H}$ and $\mathcal{C}$ respectively. We assume $|\mathbf{C}| = t$. Honest maintainer is denoted by M_w and malicious maintainer is denoted by M_t.

$\mathcal{L}_j$ denotes the j-th ledger maintained by j-th maintainer M_j which is initially empty. We denote the user record which is saved in $\mathcal{L}$ with UR. The sender and receiver of a payment are denoted by U_s and U_r respectively. The value of transaction that is transferred from a sender (B or U_s) to a recipient is denoted by v and the transaction identifier is denoted by t_{id}.

The balance of U is denoted by B, and sum of all sent and received values of U by S and R respectively. B_{max}, S_{max}, R_{max}, and V_{max} are regulatory limits on maximum allowed: balance, sum of all sent values, sum of all received values, and transaction value respectively. We denote transaction counter of a user which is incremented for each transaction (*Currency Issuance* or *Payment*) by x.

The notation $\{e_i\}_{i=1}^{N}$ is used to denote a set $\{e_1, ..., e_N\}$ with N elements. If for every positive polynomial p, there is an integer i_0 where for all integers $i > i_0$, $\mathsf{negl}(i) < \frac{1}{p(i)}$ holds, the function negl is *negligible*. We use $\mathbb{F}_q$ to denote a field with q elements. PPT stands for probabilistic polynomial time.

2.4 CBDC Formal Model

We formalize the objectives of a CBDC system as an ideal functionality in the Universal Composition framework [17]. 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 called *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 $D = 3t + 1$ holds. The functionality $\mathcal{F}_{\mathsf{CBDC}}$ maintains the following tables and mappings:

1. $T(\mathsf{U})$ 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 $\perp$.
2. Users to their accounts' state: $W = (B, S, R, x) \leftarrow \mathbb{K}(\mathsf{U})$. Initially, $\mathbb{K}(\mathsf{U}) \leftarrow \perp$.
3. $U(\mathsf{U})$ outputs pid if the user U has 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 user is in the Idle state, therefore, can start a new transaction.
4. Payment identifiers pid to transaction identifiers t_{id}: $t_{\mathsf{id}} \leftarrow P(\mathsf{pid})$.
5. Set of maintainers who engage in a specific transaction whose identifier is t_{id}: $\mathcal{M}(t_{\mathsf{id}})$.
6. 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, $\mathsf{Tid}(\mathsf{U}) \leftarrow \perp$.
7. Transaction identifiers to transaction metadata $(\mathsf{U}_s, \mathsf{U}_r, v) \leftarrow \mathsf{Rvk}(t_{\mathsf{id}})$.
8. Users to all their transaction identifiers and their role in each of them $\{t_{\mathsf{id}}^{\tau}, \mathsf{role}^{\tau}\}_{\tau=1}^{x} \leftarrow \mathsf{Trc}(\mathsf{U})$.

We note that session identifiers are of the form $\mathsf{sid} = (\mathsf{B}, \mathbb{M}, \mathsf{sid}')$ such that $\mathbb{M} = \{\mathsf{M}_j\}_{j=1}^{D}$. Initially, $\mathsf{init} \leftarrow 0$ where $\mathsf{init} \in \{0, 1\}$. At the end of *Initialization* init is set to 1 afterwards in the beginning of all parts of the functionality namely *User Registration, Currency Issuance, Payment, Abort Transaction, Privacy Revocation* and *Tracing* the functionality ignores the received message if $0 \leftarrow \mathsf{init}$.

In the following, by sending a message to $\mathbb{M}$ via delayed output, we mean, $\mathcal{F}_{\mathsf{CBDC}}$ lets adversary $\mathcal{A}$ to decide the order of maintainers who receive the message, and whether or not deliver the message to each M included in the set $\mathbb{M}$.

Functionality $\mathcal{F}_{\mathsf{CBDC}}$, part I: *Registration* and *Issuance*

Initialization.

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

User Registration.

1. Upon receiving a message $(\texttt{GenAcc}, \mathsf{sid})$ from U: If $\mathbb{K}(\mathsf{U}) = \perp$, output $(\texttt{GenAcc}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{A}$. Else, ignore.
2. Upon receiving $(\texttt{Ok.GenAcc}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{A}$: Output $(\texttt{AccGened}, \mathsf{sid}, \mathsf{U})$ to $\mathbb{M}$ via public-delayed output. Output $(\texttt{AccGened}, \mathsf{sid})$ 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.

1. Upon receiving a message $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ from B: Ignore if B not in sid. Else, generate a new pid. 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 $\mathcal{A}$.
2. Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, v)$ from U: If $\mathbb{K}(\mathsf{U}) = \perp$ or $U(\mathsf{U}) \neq \perp$, ignore. Else, retrieve $W \leftarrow \mathbb{K}(\mathsf{U})$. 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 $\mathcal{A}$.
3. Upon receiving $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ from $\mathcal{A}$: If already exits a $\mathsf{pid}' \neq \mathsf{pid}$ where $t_{\mathsf{id}} \leftarrow P(\mathsf{pid}')$, ignore. Else, if $P(\mathsf{pid}) = \perp$, set $P(\mathsf{pid}) \leftarrow t_{\mathsf{id}}$. Else, retrieve $t'_{\mathsf{id}} \leftarrow P(\mathsf{pid})$, ignore if $t'_{\mathsf{id}} \neq t_{\mathsf{id}}$. Set $\mathsf{Tid}(\mathsf{U}) \leftarrow (\mathsf{B}, \mathsf{U}, t_{\mathsf{id}}, v)$.
4. Upon receiving $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ from $\mathcal{A}$: Ignore if $\perp \leftarrow P(\mathsf{pid})$. Else, retrieve $t_{\mathsf{id}} \leftarrow P(\mathsf{pid})$. Set $\mathcal{M}(t_{\mathsf{id}}) \leftarrow \mathcal{M}(t_{\mathsf{id}}) \cup \mathsf{M}_k$ and output $(\mathtt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ to M_k via public-delayed output. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq t + 1$: Set $\mathbb{K}(\mathsf{U}) \leftarrow (B + v, S, R + v, x + 1)$, $\mathsf{Rvk}(t_{\mathsf{id}}) \leftarrow (\mathsf{B}, \mathsf{U}_r, v)$, $\mathsf{Trc}(\mathsf{U}) \leftarrow \mathsf{Trc}(\mathsf{U}) \cup (t_{\mathsf{id}}, \mathsf{receiver})$. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq 2t + 1$: Output $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{B}, v)$ to U via private-delayed output and set $U(\mathsf{U}) \leftarrow \perp$ when delivered.

In more details the components of our functionality are as follows.

1. *Initialization.*This step merely ensures that the relevant parties (B and all D maintainers $\mathbb{M}$) have been activated; the functionality keeps a record of all parties that have been initialized for the scheme.
2. *User Registration.* At the registration phase, a user U should get their account ratified by the system. If the user U has already been registered by maintainers $\mathbb{M}$, it cannot be registered again. Note that as it is common in the Universal Composition setting, we allow the adversary $\mathcal{A}$ communications and hence also block registration (i.e., we do not model denial of service attacks). The balance and regulation-related information of user W are set to initial values which are zero for balance B, sum of all sent values S, sum of all received values R of user, and transaction counter x. The maintainers are notified for each successful user registration.
3. *Currency Issuance.* In the *Currency Issuance* process different from *Payment*, only the central bank B is allowed to be the payer and there are no limits imposed to the funds that the central bank possesses. First of all, functionality $\mathcal{F}_{\mathsf{CBDC}}$ checks whether the receiver of digital currency U is a valid registered user in the system or not which means if the user U has not already been registered by maintainers $\mathbb{M}$, it cannot obtain any digital currency. The functionality imposes the regulatory restrictions of $B + v \leq B_{\mathsf{max}}$ and $R + v \leq R_{\mathsf{max}}$, where v is the amount of currency that is issued following the central bank's instructions. We remark that based on different regulatory rules in each jurisdiction, some of the restrictions such as upper bounding the value central bank B issues $v \leq V_{\mathsf{max}}$ can be easily captured or the mentioned checks $B+v \leq B_{\mathsf{max}}$ and $R+v \leq R_{\mathsf{max}}$ can be ignored for currency issuance transactions (note that as our construction is account-based rather than token-based; adding or removing such regulatory compliance constraints is relatively straightforward). Currency issuance is not a unilateral action from the central bank B as it also needs the activation of the user U who receives the digital currency. This highlights one of the distinctions of our setting compared to blockchain systems: the recipient of funds U is online during transaction and the protocol is interactive. The state of the receiver's account is updated after each currency issuance action. As before, the adversary $\mathcal{A}$ may block the currency issuance from going forward. A successful currency issuance will increase the balance of the receiver U by the indicated amount v. Transaction value, and identity of the receiver is hidden from the adversary. The ideal-world adversary $\mathcal{A}$ is also required to assign a unique transaction identifier t_{id}

and all transaction metadata are stored by the functionality in a table $\mathsf{Rvk}(t_{\mathsf{id}})$ while the t_{id} is stored in $\mathsf{Trc}(\mathsf{U})$, where U is the recipient.

4. *Payment.* As in the case of *Currency Issuance*, the *Payment* process involves both the sender U_s and the receiver U_r being activated. Contrary to issuance transaction, the functionality during payment performs the important check that the sender U_s has sufficient balance to fund the payment $B_s - v \geq 0$. Interactive payment is necessary as we claim that $\mathcal{F}_{\mathsf{CBDC}}$ captures regulatory compliance (e.g., AML, CFT) considering both parties which means both of them are supposed to know with whom they are making a payment. Hence, it is vital for the receiver U_r to actively engage in each payment. A successful payment protocol will increase the balance of the receiver U_r by the indicated amount v as well as subtract that amount from the balance of the sender U_s. Additionally, account information of each user is updated to capture different regulation policies. As in the case of issuance a unique transaction identifier t_{id} is determined by the ideal-world adversary $\mathcal{A}$ and the transaction metadata are stored in table $\mathsf{Rvk}(t_{\mathsf{id}})$ while the t_{id} is stored in $\mathsf{Trc}(\mathsf{U}_r)$ and $\mathsf{Trc}(\mathsf{U}_s)$, where U_r and U_s are the sender and recipient of the payment. Note that the adversary $\mathcal{A}$ is not aware of the transaction value, and identities of sender and receiver (unless one of them is malicious) and the t_{id} is selected independently of them.
5. *Privacy Revocation.* Privacy revocation is initiated by the maintainers who submit the transaction identifier of a fully anonymous payment they wish to revoke. If a sufficient number of them (this is set to β) agrees on the revocation of a specific transaction the functionality will recover the metadata of the specific transaction and return them to the maintainers.
6. *Tracing.* As in the case of revocation, the maintainers have to agree they want to trace a specific user. If the quorum is reached (requiring β maintainers) then the set of transaction identifiers that correspond to the agreed users will be returned to the maintainers.

Functionality $\mathcal{F}_{\mathsf{CBDC}}$, part II: *Payment* and *Auditing*

Payment.

1. Upon receiving a message $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ from U_s: If $\mathbb{K}(\mathsf{U}_s) = \perp$ or $U(\mathsf{U}_s) \neq \perp$ ignore. Else, retrieve $W_s \leftarrow \mathbb{K}(\mathsf{U}_s)$. If $S_s + v > S_{\mathsf{max}}$, or $B_s - v < 0$, or $v > V_{\mathsf{max}}$, ignore. Else, generate a new pid and set $U(\mathsf{U}_s) \leftarrow \mathsf{pid}$. If U_r is corrupted, output $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s, \mathsf{U}_r, v)$ to $\mathcal{A}$. Else, retrieve $T(\mathsf{U}_s)$: (a) If $T(\mathsf{U}_s) = 0$, output $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{A}$. (b) Else, output $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s)$ to $\mathcal{A}$.
2. Upon receiving $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{U}_s, v)$ from U_r: If $\mathbb{K}(\mathsf{U}_r) = \perp$ or $U(\mathsf{U}_r) \neq \perp$ ignore. Else, retrieve $W_r \leftarrow \mathbb{K}(\mathsf{U}_r)$. 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 $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid})$ to $\mathcal{A}$. (b) Else, output $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_r)$ to $\mathcal{A}$.
3. Upon receiving $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ from $\mathcal{A}$: If already exits a $\mathsf{pid}' \neq \mathsf{pid}$ where $t_{\mathsf{id}} \leftarrow P(\mathsf{pid}')$, ignore. Else, if $P(\mathsf{pid}) = \perp$, set $P(\mathsf{pid}) \leftarrow t_{\mathsf{id}}$. Else, retrieve $t'_{\mathsf{id}} \leftarrow P(\mathsf{pid})$, 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 $\mathsf{Tid}(\mathsf{U}_r) \leftarrow (\mathsf{U}_s, \mathsf{U}_r, t_{\mathsf{id}}, v)$.
4. Upon receiving $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ from $\mathcal{A}$: Ignore if $\perp \leftarrow P(\mathsf{pid})$. Else, retrieve $t_{\mathsf{id}} \leftarrow P(\mathsf{pid})$. Set $\mathcal{M}(t_{\mathsf{id}}) \leftarrow \mathcal{M}(t_{\mathsf{id}}) \cup \mathsf{M}_k$, and output $(\mathtt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ to M_k via public-delayed output. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq t + 1$: Set $\mathbb{K}(\mathsf{U}_s) \leftarrow (B_s - v, S_s + v, R_s, x_s + 1), \mathbb{K}(\mathsf{U}_r) \leftarrow (B_r + v, S_r, R_r + v, x_r + 1)$, $\mathsf{Rvk}(t_{\mathsf{id}}) \leftarrow (\mathsf{U}_s, \mathsf{U}_r, v)$, $\mathsf{Trc}(\mathsf{U}_s) \leftarrow \mathsf{Trc}(\mathsf{U}_s) \cup (t_{\mathsf{id}}, \mathsf{sender})$, and $\mathsf{Trc}(\mathsf{U}_r) \leftarrow \mathsf{Trc}(\mathsf{U}_r) \cup (t_{\mathsf{id}}, \mathsf{receiver})$. Once $|\mathcal{M}(t_{\mathsf{id}})| \geq 2t + 1$: Output $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$ to U_r via private-delayed output and set $U(\mathsf{U}_r) \leftarrow \perp$ when delivered. Output $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ to U_s via private-delayed output and set $U(\mathsf{U}_s) \leftarrow \perp$ when delivered.

Abort Transaction.

1. Upon receiving a message $(\texttt{AbrTnx}, \mathsf{sid})$ from U^a: If $\mathbb{K}(\mathsf{U}) = \perp$ or $\mathsf{Tid}(\mathsf{U}) = \perp$, ignore. Else, retrieve $(\mathsf{U}_s, \mathsf{U}_r, t_{\mathsf{id}}, v) \leftarrow \mathsf{Tid}(\mathsf{U})$. Send $(\texttt{AbrTnx}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{AbrTnx.Ok}, \mathsf{sid}, t_{\mathsf{id}})$ from $\mathcal{A}$: Set $\mathsf{Tid}(\mathsf{U}) \leftarrow \perp$. (a) If $|\mathcal{M}(t_{\mathsf{id}})| \leq t$: 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 $U(\mathsf{U}) \leftarrow \perp$ 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 $U(\mathsf{U}_r) \leftarrow \perp$ when delivered. Output $(\texttt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ to U_s via private-delayed output and set $U(\mathsf{U}_s) \leftarrow \perp$ when delivered. Output $(\texttt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathbb{M}$ via public-delayed output.

Privacy Revocation.

1. Upon receiving a message $(\texttt{RvkAnm}, \mathsf{sid}, t^j_{\mathsf{id}})$ from maintainer M_j: If $\mathsf{Rvk}(t^j_{\mathsf{id}}) = \perp$, 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 $|\{j | t^j_{\mathsf{id}} = t_{\mathsf{id}}\}| \geq t+1$, set $X \leftarrow t_{\mathsf{id}}$.
2. 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 $(\mathsf{U}_s, \mathsf{U}_r, v) \leftarrow \mathsf{Rvk}(X)$. Output $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)^b$ to $\mathbb{M}$ via public-delayed output.

Tracing.

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

[a] Either $\mathsf{U} = \mathsf{U}_s$ or $\mathsf{U} = \mathsf{U}_r$ holds.
[b] For a currency issuance transaction $\mathsf{U}_s = \mathsf{B}$.

3 Our Construction

In our construction, we aim to achieve all the financial, regulatory and security properties described informally in Sec. 2.2 and formally in Sec. 2.4. We assume that the whole number of maintainers are $3t+1$ and t of them can be corrupted by the adversary. Hence, we set the thresholds of blind signature scheme and auditing as $\alpha = 2t+1$ and $\beta = t+1$ respectively.

3.1 High-level Technical Overview of Our Protocol

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 restrictions. Users update their accounts when transacting. For each new *Currency Issuance* or payment transaction, the involved parties 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 to the previous account snapshot acc (which has been signed by at least α maintainers). A user also discloses $\sigma^{\mathsf{Rnd}}_{\mathbb{M}}$ that is a re-randomization of the consolidated signature $\sigma_{\mathbb{M}}$ on their previous account snapshot. The disclosed tags are stored by maintainers for the purpose of enforcing users to use their most updated accounts (as in Chaum's double-spending prevention for

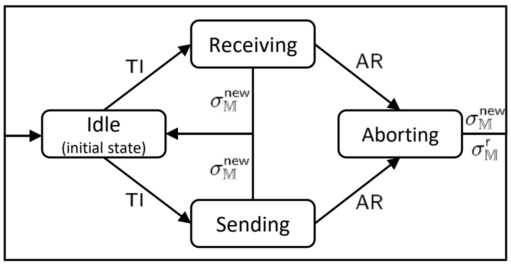

Idle
(initial state)
Receiving
Sending
Aborting
TI
TI
AR
AR
$\sigma_{\mathbb{M}}^{\mathsf{new}}$
$\sigma_{\mathbb{M}}^{\mathsf{new}}$
$\sigma_{\mathbb{M}}^{\mathsf{new}}$
$\sigma_{\mathbb{M}}^{\mathsf{r}}$

Fig. 1. User's State Transition in PEReDi's Transactions. TI: Transaction Information. AR: Abort Request. $\sigma_{\mathbb{M}}^{\mathsf{new}}$: Maintainers' signature on the new account of the user. $\sigma_{\mathbb{M}}^{\mathsf{r}}$: Maintainers' signature on the refreshed account of the user.

online cash [21]). To support tracing, the protocol in fact computes tags pseudo-randomly so that they can be recomputed by the *Auditing* protocol (in a distributed fashion by the maintainers).

The newly updated account $\mathsf{acc}^{\mathsf{new}}$ is given to $\mathbb{M}$ for signing together with a proof that the new account $\mathsf{acc}^{\mathsf{new}}$ is consistent with the previous account snapshot acc and the transaction value v issued by B to U in the *Currency Issuance* protocol or transferred from U_s to U_r in the *Payment* protocol. To this end, users prove to $\mathbb{M}$ in a privacy-preserving way that their new accounts snapshots $\mathsf{acc}^{\mathsf{new}}$ are updated honestly. For instance, the same value is deducted from U_s's account and that value is added to U_r's account. Similarly, the account of the receiver is updated with respect to the value of digital currency issued by the central bank while making sure that attacks such as a replay attack on central bank's message is prohibited. Moreover, both U_s and U_r's new accounts $\mathsf{acc}_s^{\mathsf{new}}$ and $\mathsf{acc}_r^{\mathsf{new}}$ comply with the system's regulatory compliance rules. The parties engaged in a payment should acquire at least α number of maintainers' blind signature shares $\sigma^{\mathsf{new},\mathfrak{B}}$ on their new accounts. They locally unblind these signature shares σ^{new} and aggregate them to create a single consolidated signature $\sigma_{\mathbb{M}}^{\mathsf{new}}$ on their new account snapshot.

Furthermore, every transaction results in a 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 so that we achieve full privacy. It is only retrievable and reconstructable by an audit committee using the information saved in $\mathcal{L}$ for the purpose of privacy revocation and tracing. In other words, an audit can be done when the audit committee has been convinced that a specific transaction or user is suspicious enough for anonymity to be revoked or have its counterparties be traced respectively. Note that tracing and revocation can be applied in an adaptive fashion to reconstruct a set of counterparties across a sequence of payments.

In the following, we describe user's state transition (in *Currency Issuance* and *Payment* protocols) in the PEReDi's setting which is depicted in Fig. 1. Upon receiving environment's $\mathcal{Z}$ command (of the form $(\texttt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, v)$ or $(\texttt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s, v)$ or $(\texttt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$) to make a transaction, if the user U is in:

1. The Idle state, it 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 is changed to Receiving (from central bank B or from another user U_s) or Sending (to another user U_r).
2. One of the states Receiving or Sending (which means U's most recent transaction is pending), U ignores $\mathcal{Z}$'s message.

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

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

Upon $\mathcal{Z}$'s instruction (of the form $(\mathtt{AbrTnx}, \mathsf{sid})$) for sending 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 the state of U 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 two following scenarios are for the case when U is in the Aborting state:

1. If at least $t + 1$ maintainers have saved a *sender-receiver* TI pair in their ledgers (which guarantees at least one honest maintainer has the pair), maintainers ignore $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ and send their signatures for $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ to U. Upon generating unblinded-consolidated maintainers' signature on the new account $\sigma_{\mathbb{M}}^{\mathsf{new}}$, state is changed to Idle.
2. Else, 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 unblinded-consolidated maintainers' signature on the refreshed account $\sigma_{\mathbb{M}}^{\mathsf{r}}$, state is changed to Idle.

Furthermore, you can find a pictorial representation of all the sub-protocols of our construction in Fig. 2-6. Note that for simplicity, in the figures we do not include the messages between the environment $\mathcal{Z}$ and the parties. For the same reason, we do not depict AR messages as well.

3.2 Details of the Construction

In this section, we describe our CBDC protocol Π_{PEReDi}. We will prove that Π_{PEReDi} securely realizes $\mathcal{F}_{\mathsf{CBDC}}$. Our construction uses several concrete cryptographic components (see Appendix A) and ideal functionalities (see Appendix B). Our scheme uses the Coconut Threshold Blind Signature scheme (TBS) [36,33] and the Threshold ElGamal Encryption (TE) scheme [25,31,28]. We will reduce the unforgeability of Coconut to its underlying Pointcheval-Sanders [32] signature component. PEReDi employs the following functionalities: a Key-Registration functionality $\mathcal{F}_{\mathsf{KR}}$, a communication Channels functionality $\mathcal{F}_{\mathsf{C}}$ (parameterized by different labels, e.g., "sa" for a sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$), a Broadcast functionality $\mathcal{F}_{\mathsf{B}}$, a Byzantine Agreement functionality $\mathcal{F}_{\mathsf{BA}}$, a Non-Interactive Zero Knowledge functionality $\mathcal{F}_{\mathsf{NIZK}}$ and a Signature of Knowledge functionality $\mathcal{F}_{\mathsf{SoK}}$.

We will assume that transacting parties communicate through variants of $\mathcal{F}_{\mathsf{C}}$ as specified. We note that some sender-anonymity is necessary for privacy, as otherwise network "leakage" will trivially reveal the counterparties of a transaction irrespective of the strength of cryptographic protections at the transactional level. We note that in a real-world deployment such network leakage may be considered tolerable — our analysis would apply directly to such setting as well, exhibiting the unavoidable concession that the adversary may break privacy via traffic analysis.

Throughout this section we use the notation of Sec. 2.3. Each maintainer M has its own ledger $\mathcal{L}$ for storing registration and transaction information. In the *Currency Issuance* and *Payment* protocols of the construction below, the sender (U_s or B) and receiver (U or U_r) separately send their transaction information TI to all maintainers $\mathbb{M}$. However, a plausible alternative communication pattern could have the sender sending its transaction information TI to the receiver and then the receiver sending both the sender's TI and its own TI to $\mathbb{M}$.

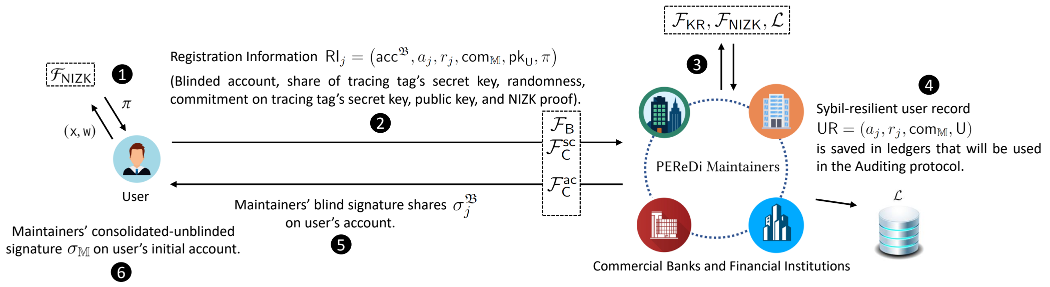

$\mathcal{F}_{\mathsf{NIZK}}$
1
$(\mathtt{x}, \mathtt{w})$
π
User
Registration Information $\mathsf{RI}_j = (\mathsf{acc}^{\mathfrak{B}}, a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}, \pi)$
(Blinded account, share of tracing tag's secret key, randomness, commitment on tracing tag's secret key, public key, and NIZK proof).
2
$\mathcal{F}_{\mathsf{B}}$
$\mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$
$\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$
Maintainers' blind signature shares $\sigma_j^{\mathfrak{B}}$ on user's account.
5
Maintainers' consolidated-unblinded signature $\sigma_{\mathbb{M}}$ on user's initial account.
6
$\mathcal{F}_{\mathsf{KR}}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{L}$
3
PEReDi Maintainers
Commercial Banks and Financial Institutions
4
Sybil-resilient user record $\mathsf{UR} = (a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{U})$ is saved in ledgers that will be used in the Auditing protocol.
$\mathcal{L}$

Fig. 2. *User Registration* Protocol

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 algorithm as outputs. Participants of the network independently call the key generation algorithm for each underlying cryptographic scheme to generate their keys (see Appendix A). The public keys of all parties are maintained in a public-key directory and are assumed to be accessible on demand by calling $\mathcal{F}_{\mathsf{KR}}$ with input $(\mathtt{RetrieveKey}, \mathsf{sid}, \mathsf{P})$ for party P.

User Registration. Maintainers $\mathbb{M}$ enroll a user U in the CBDC system by creating a signature on the user's initial account. Afterwards, U uses the signature to create transactions. For registration, U with a pair of public-secret key $(\mathsf{pk}_{\mathsf{U}}, \mathsf{sk}_{\mathsf{U}})$ and a secret key a (used in tag generation) engages in a threshold blind signature TBS protocol with $\mathbb{M}$ where U proves honest creation of its initial account to $\mathbb{M}$. The output of this protocol is a signed account $\sigma_{\mathbb{M}}$ for U (needed for its first transaction) and the user record UR saved in the ledger $\mathcal{L}$ of each maintainer M (required for additional investigation during the *Auditing* protocol). Every user's account consists of a tuple of values

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

During registration, U sets B, S and R to 0 and x to 1.

Upon receiving $(\mathtt{GenAcc}, \mathsf{sid})$ (from $\mathcal{Z}$), U who is initially in the Idle state initiates the *User Registration* protocol, see Fig. 2, to get the account signed by $\mathbb{M}$. U generates its registration information

$$\mathsf{RI}_j = \left(\mathsf{acc}^{\mathfrak{B}}, a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}, \pi\right)$$

as follows:

1. Given acc, it calls $\mathsf{PrepareBlindSign}$ algorithm of the threshold blind signature scheme TBS to obtain a blinded account $\mathsf{acc}^{\mathfrak{B}}$.
2. It calls $\{a_j\}_{j=1}^{D} \overset{\$}{\leftarrow} \mathsf{SSH.Share}^{D,\beta}(a)$ to secret share a and computes $\mathsf{com}_j = g^{a_j} h^{r_j}$ for $r_j \overset{\$}{\leftarrow} \mathbb{Z}_p^*$. It sets $\mathsf{com}_{\mathbb{M}} = \{\mathsf{com}_j\}_{j=1}^{D}$.
3. It calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$, and receives $(\mathtt{Proof}, \mathsf{sid}, \pi)$ where π is a NIZK proof of knowledge for statement

$$\mathtt{x} = \left(\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}\right)$$

and witness

$$\mathtt{w} = \left(\mathsf{acc}, \{a_j\}_{j=1}^{D}, \mathsf{r}_{\mathrm{bacc}}, \mathsf{r}_{\mathsf{com}}\right)$$

We denote the randomness used to create the blinded account $\mathsf{acc}^{\mathfrak{B}}$ and the commitment $\mathsf{com}_{\mathbb{M}}$ by $\mathsf{r}_{\mathrm{bacc}}$ and $\mathsf{r}_{\mathsf{com}}$, respectively and define the relation $\mathtt{R}(\mathtt{x}, \mathtt{w})$ of NIZK as follows:

(a) The secret key sk_U in the blinded account $\mathsf{acc}^{\mathfrak{B}}$ is the secret key associated with public key pk_U.
(b) 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 $\mathsf{com}_{\mathbb{M}}$.
(c) $\mathsf{acc}^{\mathfrak{B}}$ is generated such that $B = S = R = 0$ and $x = 1$ hold.
(d) The user U knows the randomness $\mathsf{r}_{\mathrm{bacc}}$ and $\mathsf{r}_{\mathsf{com}}$.

4. It calls $\mathcal{F}_\mathsf{B}$ with $(\mathtt{Broadcast}, \mathsf{sid}, \mathsf{com}_{\mathbb{M}})$.
5. It calls $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{RI}_j)$ to the secure channel $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$ for $j = 1, \ldots, D$. Specifically, U calls $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{RI}_k)$ $(1 \le k \le D-1)$ and waits for $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$ to send back $(\mathtt{Continue}, \mathsf{sid})$ then U proceeds by calling $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{k+1}, \mathsf{RI}_{k+1})$.

Each maintainer (e.g., M_j):

1. Generates pairs of messages. Each pair contains the received message from $\mathcal{F}_\mathsf{B}$ and $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$ where both messages have the same identifier U of the user. In other words, it receives $(\mathtt{Broadcasted}, \mathsf{sid}, \mathsf{U}', \mathsf{com}_{\mathbb{M}})$ from $\mathcal{F}_\mathsf{B}$ and $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}', \mathsf{RI}_j)$ from the secure channel $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$. If $\mathsf{U}' = \mathsf{U}$, M_j generates a pair of messages containing the received messages from $\mathcal{F}_\mathsf{B}$ and $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$. Else, waits to receive such messages.
2. If $\mathsf{com}_{\mathbb{M}}$ received from $\mathcal{F}_\mathsf{B}$ is not equal to $\mathsf{com}_{\mathbb{M}}$ included in RI_j received from $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$, aborts.
3. Else, ignores the message if at least one of the following conditions holds:
   (a) There already exists a user record UR' in $\mathcal{L}_j$ where $\mathsf{U}' = \mathsf{U}$.
   (b) Upon calling $\mathcal{F}_{\mathsf{KR}}$ with $(\mathtt{RetrieveKey}, \mathsf{sid}, \mathsf{U})$, it receives $(\mathtt{KeyRetrieved}, \mathsf{sid}, \mathsf{U}, \mathsf{pk}')$ such that $\mathsf{pk}_\mathsf{U} \neq \mathsf{pk}'$.
   (c) Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$, it receives $(\mathtt{Verification}, \mathsf{sid}, 0)$.
   (d) Given (a'_j, r'_j) received from $\mathcal{F}_\mathsf{C}^{\mathsf{sc}}$ included in RI_j, it computes $g^{a'_j} h^{r'_j}$ which is not equal to com_j for $\mathsf{com}_j \in \mathsf{com}_{\mathbb{M}}$.
   (e) Know Your Customer (KYC) guidelines for U is not verified.
4. Else, the user record:
$$\mathsf{UR} = (a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{U})$$
is saved in $\mathcal{L}_j$.
5. Signs associated information of $\mathsf{acc}^{\mathfrak{B}}$ using the $\mathsf{BlindSign}$ algorithm of TBS scheme to obtain blind signature share $\sigma_j^{\mathfrak{B}}$.
6. Sends back $\sigma_j^{\mathfrak{B}}$ to U using the authenticated channel $\mathcal{F}_\mathsf{C}^{\mathsf{ac}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, \sigma_j^{\mathfrak{B}})$.
7. Outputs $(\mathtt{AccGened}, \mathsf{sid}, \mathsf{U})$ (to $\mathcal{Z}$).

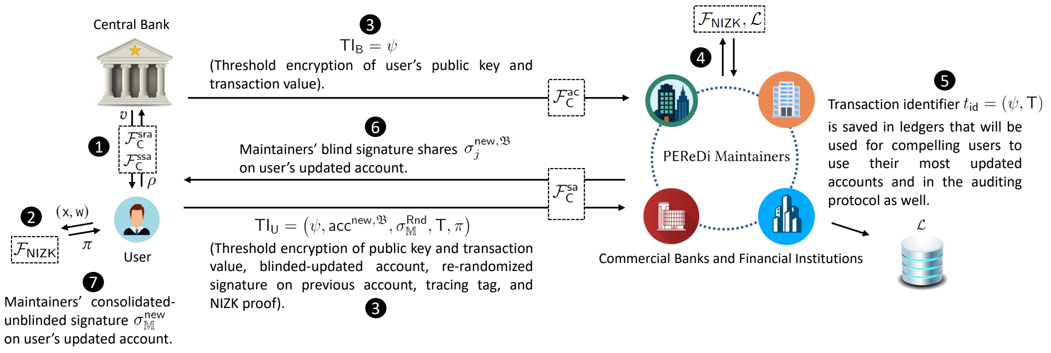
The user U:

1. Receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathfrak{B}})$ for different j from the authenticated channel $\mathcal{F}_\mathsf{C}^{\mathsf{ac}}$.
2. Unblinds at least α different signature shares $\{\sigma_j\}_{j=1}^{\alpha}$ using the $\mathsf{Unblind}$ algorithm of the TBS scheme.
3. Aggregates unblinded signature shares using the $\mathsf{TBS.Agg}$ algorithm of the TBS scheme to form one consolidated signature $\sigma_{\mathbb{M}}$.
4. Outputs $(\mathtt{AccGened}, \mathsf{sid})$ (to $\mathcal{Z}$).

Currency Issuance. Upon receiving $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ (from $\mathcal{Z}$), B initiates *Currency Issuance* protocol as shown in Fig. 3. To issue a digital currency worth of v for U, first of all, B sends v to U using the secure-receiver anonymous channel $\mathcal{F}_\mathsf{C}^{\mathsf{sra}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}, v)$ so that U receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, v)$. Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, v)$ (from $\mathcal{Z}$), if U is in Idle state, it sends the randomness ρ of ψ to B using the secure-sender anonymous channel $\mathcal{F}_\mathsf{C}^{\mathsf{ssa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \rho)$[1].

ψ is ElGamal threshold encryption of U's public key pk_U and g^v.

[1] Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, v)$ (from $\mathcal{Z}$), if U is in one of the $\mathsf{Sending}$ or $\mathsf{Receiving}$ state, U ignores the message.



Central Bank
3
$\mathsf{TI_B} = \psi$
(Threshold encryption of user's public key and transaction value).
$\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$
$\mathcal{F}_{\mathsf{NIZK}}, \mathcal{L}$
4
5
Transaction identifier $t_{\mathsf{id}} = (\psi, \mathsf{T})$ is saved in ledgers that will be used for compelling users to use their most updated accounts and in the auditing protocol as well.
v
1
$\mathcal{F}_{\mathsf{C}}^{\mathsf{sra}}$
$\mathcal{F}_{\mathsf{C}}^{\mathsf{ssa}}$
ρ
6
Maintainers' blind signature shares $\sigma_j^{\mathsf{new},\mathfrak{B}}$ on user's updated account.
$\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$
PEReDi Maintainers
2
(x, w)
π
$\mathcal{F}_{\mathsf{NIZK}}$
User
$\mathsf{TI_U} = (\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi)$
(Threshold encryption of public key and transaction value, blinded-updated account, re-randomized signature on previous account, tracing tag, and NIZK proof).
3
Commercial Banks and Financial Institutions
$\mathcal{L}$
7
Maintainers' consolidated-unblinded signature $\sigma_{\mathbb{M}}^{\mathsf{new}}$ on user's updated account.

Fig. 3. *Currency Issuance* Protocol

U waits for the message $(\mathtt{Continue}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ssa}}$ and after receiving it, U sends its transaction information $\mathsf{TI_U}$ to $\mathbb{M}$ which is of the form:

$$\mathsf{TI_U} = \left(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi\right)$$

The components of $\mathsf{TI_U}$ is computed by U who does the following:

1. Computes threshold ElGamal encryption as follows setting its public key $\mathsf{pk_U}$ and g^v as plaintexts:
$$\psi = (c_1, c_2, c_3) = \left(g^{\rho}, \mathsf{pk}_{1,\mathbb{M}}^{\rho} \cdot \mathsf{pk_U}, \mathsf{pk}_{2,\mathbb{M}}^{\rho} \cdot g^{v}\right)$$
2. Computes $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$. Similar to $\mathsf{acc}^{\mathfrak{B}}$ at *User Registration* protocol, to obtain blind signature shares of $\mathbb{M}$ on U's new account which is as follows:
$$\mathsf{acc}^{\mathsf{new}} = (B + v, S, R + v, \mathsf{sk_U}, x + 1, a)$$
U should prove that it has a valid signature $\sigma_{\mathbb{M}}$ on its previous account acc 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 $\mathsf{PrepareBlindSign}$ algorithm and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is computed for U's previous account acc (for which it has consolidated signature $\sigma_{\mathbb{M}}$) using the $\mathsf{ProveSig}$ algorithm of the TBS scheme.
3. Computes
$$\mathsf{T} = g^{a^x}$$
that is a tag used for compelling users to use their most updated accounts in which x is an incrementing value per transaction. As we will see, same value is used for tracing the user when it is necessary.
4. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$, and obtains $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from it in which π is a NIZK proof for the statement
$$\mathtt{x} = \left(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}\right)$$
We denote the randomness used to create $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ and threshold encryption ψ by $\mathsf{r_{reg}}$. The witness of π is
$$\mathtt{w} = (\mathsf{acc}, \mathsf{r_{reg}}, v)$$
for the following relation $\mathtt{R}(\mathtt{x}, \mathtt{w})$:
   (a) The secret key $\mathsf{sk_U}$ used in $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ is the secret key associated with public key $\mathsf{pk_U}$ in the threshold encryption ψ generated under the public keys of maintainers $\mathsf{pk}_{1,\mathbb{M}}$ and $\mathsf{pk}_{2,\mathbb{M}}$.
   (b) T is generated using a and x included in acc (for which user reveals $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$).

   (c) $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is re-randomization of $\sigma_{\mathbb{M}}$ which is α different valid aggregated signature shares of maintainers on acc.
   (d) $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ is generated considering acc and v in ψ. Hence, $B^{\mathsf{new}} = B + v, S^{\mathsf{new}} = S, R^{\mathsf{new}} = R + v, \mathsf{sk}_{\mathsf{U}}^{\mathsf{new}} = \mathsf{sk}_{\mathsf{U}}, x^{\mathsf{new}} = x + 1$ and $a^{\mathsf{new}} = a$ hold for $\mathsf{acc}^{\mathsf{new}}$. Additionally, $B^{\mathsf{new}} \leq B_{\mathsf{max}}$, and $R^{\mathsf{new}} \leq R_{\mathsf{max}}$ hold[2].
   (e) U knows the randomness $\mathsf{r}_{\mathsf{reg}}$.
5. Calls the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{U}})$ for $j = 1, \ldots, D$. Specifically, U calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_{\mathsf{U}})$ $(1 \leq k \leq D-1)$ and waits for functionality to send back $(\mathtt{Continue}, \mathsf{sid})$, then U proceeds by calling $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{k+1}, \mathsf{TI}_{\mathsf{U}})$.

Upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}, \rho)$ from the secure-sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ssa}}$, B also sends its transaction information

$$\mathsf{TI}_{\mathsf{B}} = \psi$$

to $\mathbb{M}$. The central bank B calls the authenticated channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}})$ for $j = 1, \ldots, D$. Specifically, B calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_{\mathsf{B}})$ $(1 \leq k \leq D-1)$ and waits for $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ to send back $(\mathtt{Continue}, \mathsf{sid})$ then B proceeds by calling $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{k+1}, \mathsf{TI}_{\mathsf{B}})$.

Each maintainer (e.g., M_j):

1. Receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid})$ from the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ and parses TI_{U} as $\left(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi\right)$ (resp. receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, \mathsf{TI}_{\mathsf{B}})$ from the authenticated channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ and parses TI_{B} as ψ).
2. Ignores TI_{U} if at least one of the following conditions holds:
   (a) There already exists a transaction identifier t'_{id} (for an issuance transaction or an aborted transaction) in its ledger $\mathcal{L}_j$ where $\mathsf{T}' = \mathsf{T}$ or $\psi' = \psi$ (the latter only applies for t'_{id} of issuance transaction).
   (b) There already exists a transaction identifier t'_{id} (for a payment transaction) in $\mathcal{L}_j$ where $\mathsf{T}'_s = \mathsf{T}$ or $\mathsf{T}'_r = \mathsf{T}$.
   (c) Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$, it receives $(\mathtt{Verification}, \mathsf{sid}, 0)$.
3. Else, records TI_{U} (resp. TI_{B}) in $\mathcal{L}_j$.
4. Upon receiving TI_{B} (resp. TI_{U} that passes all the checks) that has ψ' value where $\psi' = \psi$, it saves a *sender-receiver* pair $(\mathsf{TI}_{\mathsf{B}}, \mathsf{TI}_{\mathsf{U}})$ in $\mathcal{L}_j$[3].
5. Saves transaction identifier
$$t_{\mathsf{id}} = (\psi, \mathsf{T})$$
   in $\mathcal{L}_j$.
6. Signs associated information of $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$ using $\mathsf{BlindSign}$ algorithm to obtain blind signature share $\sigma_j^{\mathsf{new},\mathfrak{B}}$.
7. Calls the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}, \sigma_j^{\mathsf{new},\mathfrak{B}})$.
8. Outputs $(\mathtt{Issued}, \mathsf{sid}, t_{\mathsf{id}})$ (to $\mathcal{Z}$).

The user U:

1. Receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{new},\mathfrak{B}})$ for different j from the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$.
2. Unblinds at least α different maintainers' signature shares $\left\{\sigma_j^{\mathsf{new}}\right\}_{j=1}^{\alpha}$ using $\mathsf{Unblind}$ algorithm.
3. Aggregates unblinded signature shares based on $\mathsf{TBS.Agg}$ algorithm to form one consolidated signature $\sigma_{\mathbb{M}}^{\mathsf{new}}$.
4. Outputs $(\mathtt{Issued}, \mathsf{sid}, v)$ (to $\mathcal{Z}$).

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

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

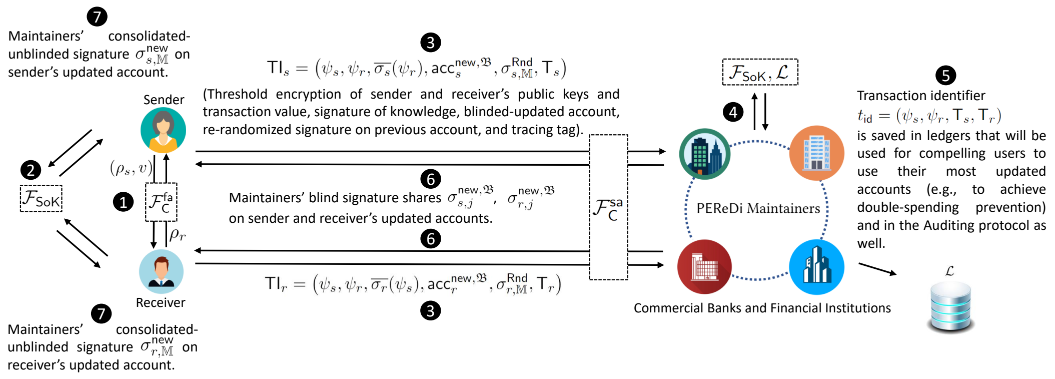

7
Maintainers' consolidated-unblinded signature $\sigma^{\text{new}}_{s,\mathbb{M}}$ on sender's updated account.
3
$\mathsf{TI}_s = (\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}^{\text{new},\mathfrak{B}}_s, \sigma^{\text{Rnd}}_{s,\mathbb{M}}, \mathsf{T}_s)$
(Threshold encryption of sender and receiver's public keys and transaction value, signature of knowledge, blinded-updated account, re-randomized signature on previous account, and tracing tag).
Sender
2
$\mathcal{F}_{\mathsf{SoK}}$
(ρ_s, v)
1
$\mathcal{F}^{\mathsf{fa}}_{\mathsf{C}}$
ρ_r
6
Maintainers' blind signature shares $\sigma^{\text{new},\mathfrak{B}}_{s,j}$, $\sigma^{\text{new},\mathfrak{B}}_{r,j}$ on sender and receiver's updated accounts.
6
$\mathcal{F}^{\mathsf{sa}}_{\mathsf{C}}$
Receiver
$\mathsf{TI}_r = (\psi_s, \psi_r, \overline{\sigma_r}(\psi_s), \mathsf{acc}^{\text{new},\mathfrak{B}}_r, \sigma^{\text{Rnd}}_{r,\mathbb{M}}, \mathsf{T}_r)$
3
7
Maintainers' consolidated-unblinded signature $\sigma^{\text{new}}_{r,\mathbb{M}}$ on receiver's updated account.
$\mathcal{F}_{\mathsf{SoK}}, \mathcal{L}$
4
PEReDi Maintainers
Commercial Banks and Financial Institutions
5
Transaction identifier $t_{\text{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ is saved in ledgers that will be used for compelling users to use their most updated accounts (e.g., to achieve double-spending prevention) and in the Auditing protocol as well.
$\mathcal{L}$

Fig. 4. *Payment* Protocol

Payment. To make a payment, upon receiving $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ (from $\mathcal{Z}$), if U_s is in Idle state, it initiates the *Payment* protocol as shown in Fig. 4 by sending randomness ρ_s of ψ_s and the value of transaction v to the receiver U_r via fully anonymous channel $\mathcal{F}^{\mathsf{fa}}_{\mathsf{C}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}_r, (\rho_s, v))$[4].

ψ_s is ElGamal threshold encryption of U_s's public key pk_s and g^v.

On receiving $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{U}_s, v)$ (from $\mathcal{Z}$), if U_r is in Idle state, it sends back randomness ρ_r used in ψ_r to U_s using the fully anonymous channel $\mathcal{F}^{\mathsf{fa}}_{\mathsf{C}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}_s, \rho_r)$[5].

ψ_r is ElGamal threshold encryption of U_r's public key pk_r.

Furthermore, U_s and U_r generate their transaction information. The transaction information TI of U_s is of the form:

$$\mathsf{TI}_s = \left(\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}^{\text{new},\mathfrak{B}}_s, \sigma^{\text{Rnd}}_{s,\mathbb{M}}, \mathsf{T}_s\right)$$

The components of TI_s is computed by U_s who does the following:

1. Computes threshold ElGamal encryptions

$$\psi_s = (c_{s,1}, c_{s,2}, c_{s,3}) = \left(g^{\rho_s}, \mathsf{pk}^{\rho_s}_{1,\mathbb{M}} \cdot \mathsf{pk}_s, \mathsf{pk}^{\rho_s}_{2,\mathbb{M}} \cdot g^v\right)$$

   and

$$\psi_r = (c_{r,1}, c_{r,2}) = \left(g^{\rho_r}, \mathsf{pk}^{\rho_r}_{1,\mathbb{M}} \cdot \mathsf{pk}_r\right)$$

2. Computes $\mathsf{acc}^{\text{new},\mathfrak{B}}_s, \sigma^{\text{Rnd}}_{s,\mathbb{M}}$, and T_s similar to the *Currency Issuance* protocol where the new account of U_s is as follows:

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

3. Calls $\mathcal{F}_{\mathsf{SoK}}$ on input $(\mathtt{Sign}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \mathtt{w}_s)$ and receives $(\mathtt{Signature}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$ from it in which $\overline{\sigma_s}(\psi_r)$ is U_s's signature of knowledge on ψ_r that also binds the message ψ_r to the proof so that it proves knowledge of $\mathtt{w}_s$ satisfying the relation $\mathtt{R}(\mathtt{x}_s, \mathtt{w}_s)$ for the statement

$$\mathtt{x}_s = \left(\psi_s, \mathsf{acc}^{\text{new},\mathfrak{B}}_s, \sigma^{\text{Rnd}}_{s,\mathbb{M}}, \mathsf{T}_s\right)$$

   and the message of signature ψ_r.

   We denote the set of all random values $\mathsf{acc}^{\text{new},\mathfrak{B}}_s, \sigma^{\text{Rnd}}_{s,\mathbb{M}}$, and ψ_s by r_s. The witness of $\overline{\sigma_s}(\psi_r)$ is

$$\mathtt{w}_s = (\mathsf{acc}_s, \mathsf{r}_s, v)$$

[4] Upon receiving $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ (from $\mathcal{Z}$), if U_s is in one of the $\mathsf{Sending}$ or $\mathsf{Receiving}$ state, it ignores the message.

[5] Upon receiving $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{U}_s, v)$ (from $\mathcal{Z}$), if U_r is in one of the $\mathsf{Sending}$ or $\mathsf{Receiving}$ state, it ignores the message.

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

(a) The secret key sk_s used in $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$ is the secret key associated with public key pk_s in the threshold encryption ψ_s generated under the public keys of maintainers $\mathsf{pk}_{1,\mathbb{M}}$ and $\mathsf{pk}_{2,\mathbb{M}}$.
(b) T_s is generated using a_s and x_s included in acc_s (for which user reveals $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$).
(c) $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$ is re-randomization of $\sigma_{s,\mathbb{M}}$ which is α different valid aggregated signature shares of maintainers on acc_s.
(d) $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$ is generated considering acc_s and v in ψ_s. Hence, $B_s^{\mathsf{new}} = B_s - v, S_s^{\mathsf{new}} = S_s + v, R_s^{\mathsf{new}} = R_s, \mathsf{sk}_s^{\mathsf{new}} = \mathsf{sk}_s, x_s^{\mathsf{new}} = x_s + 1$ and $a_s^{\mathsf{new}} = a_s$ hold for $\mathsf{acc}_s^{\mathsf{new}}$. Additionally, $0 \leq B_s^{\mathsf{new}}$ $S_s^{\mathsf{new}} \leq S_{\mathsf{max}}$ and $v \leq V_{\mathsf{max}}$ hold.
(e) U_s knows the randomness r_s.

The transaction information of U_r, TI_r is similar to TI_s with values associated to U_r which is

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

Hence, everything is similar to what has been described for U_s except that $\mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}$ is generated considering acc_r (for which user reveals $\sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}$) and v in $c_{s,3}$ (U_r gets to know ρ_s). The new account of the receiver is

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

Hence, $B_r^{\mathsf{new}} = B_r + v, S_r^{\mathsf{new}} = S_r, R_r^{\mathsf{new}} = R_r + v, \mathsf{sk}_r^{\mathsf{new}} = \mathsf{sk}_r, x_r^{\mathsf{new}} = x_r + 1$ and $a_r^{\mathsf{new}} = a_r$ hold for $\mathsf{acc}_r^{\mathsf{new}}$. Additionally, $B_r^{\mathsf{new}} \leq B_{\mathsf{max}}$ and $R_r^{\mathsf{new}} \leq R_{\mathsf{max}}$ hold[6].
The sender U_s (resp. receiver U_r):

1. Calls the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_s)$ (resp. $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_r)$) for $j = 1, \ldots, D$. Specifically, U_s (resp. U_r) calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_s)$ (resp. $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{TI}_r)$) $(1 \leq k \leq D-1)$ and waits for the channel to send back $(\mathtt{Continue}, \mathsf{sid})$, then U_s (resp. U_r) proceeds by calling $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{k+1}, \mathsf{TI}_s)$ (resp. $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{k+1}, \mathsf{TI}_r)$).
   Note that, U_s, after receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}_r, \rho_r)$ from $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ sends its TI_s to $\mathbb{M}$. U_r waits for the message $(\mathtt{Continue}, \mathsf{sid})$ from $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ and then sends TI_r to $\mathbb{M}$.

Each maintainer (e.g., M_j):

1. Receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_s, \mathsf{mid}_s)$ (resp. $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_r, \mathsf{mid}_r)$) from the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ and parses TI_s as $\left(\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s\right)$ (resp. parses TI_r as $\left(\psi_s, \psi_r, \overline{\sigma_r}(\psi_s), \mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}, \sigma_{r,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_r\right)$).
2. Ignores TI_s (resp. TI_r) if at least one of the following conditions holds:
   (a) There already exists a transaction identifier t'_{id} (for an issuance transaction or an aborted transaction) in its ledger $\mathcal{L}_j$ where $\mathsf{T}' = \mathsf{T}_s$ (resp. $\mathsf{T}' = \mathsf{T}_r$).
   (b) There already exists a transaction identifier t'_{id} (for a payment transaction) in $\mathcal{L}_j$ where $\mathsf{T}'_s = \mathsf{T}_s$ or $\mathsf{T}'_r = \mathsf{T}_s$ (resp. $\mathsf{T}'_s = \mathsf{T}_r$ or $\mathsf{T}'_r = \mathsf{T}_r$).
   (c) Upon calling $\mathcal{F}_{\mathsf{SoK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$ (resp. $(\mathtt{Verify}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s))$), it receives $(\mathtt{Verified}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r), 0)$ (resp. $(\mathtt{Verified}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s), 0)$).
3. Else, records TI_s (resp. TI_r) in $\mathcal{L}_j$.
4. Upon receiving a transaction information (which has not been ignored w.r.t. the conditions above) that has (ψ'_s, ψ'_r) value where $(\psi'_s, \psi'_r) = (\psi_s, \psi_r)$, it saves a *sender-receiver* pair $(\mathsf{TI}_s, \mathsf{TI}_r)$ in $\mathcal{L}_j$.
5. Saves transaction identifier
$$t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$$
   in $\mathcal{L}_j$.

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

6. Signs associated information of $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$ and $\mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}$ using $\mathsf{BlindSign}$ algorithm to obtain blind signature shares $\sigma_{s,j}^{\mathsf{new},\mathfrak{B}}$ and $\sigma_{r,j}^{\mathsf{new},\mathfrak{B}}$ that belong to U_s and U_r respectively.
7. Calls the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_s, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}})$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_r, \sigma_{r,j}^{\mathsf{new},\mathfrak{B}})$.
8. Outputs $(\mathtt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ (to $\mathcal{Z}$).

The sender U_s (resp. receiver U_r):

1. Receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}})$ (resp. $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_{r,j}^{\mathsf{new},\mathfrak{B}})$) for different j from the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$.
2. Unblinds α different maintainers' signature shares $\{\sigma_{s,j}^{\mathsf{new}}\}_{j=1}^{\alpha}$ (resp. $\{\sigma_{r,j}^{\mathsf{new}}\}_{j=1}^{\alpha}$) using $\mathsf{Unblind}$ algorithm.
3. Aggregates unblinded signature shares based on $\mathsf{TBS.Agg}$ algorithm to form one consolidated signature $\sigma_{s,\mathbb{M}}^{\mathsf{new}}$ (resp. $\sigma_{r,\mathbb{M}}^{\mathsf{new}}$).
4. Outputs $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ (to $\mathcal{Z}$) (resp. U_r outputs $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$).

Abort Transaction. In *Currency Issuance* and *Payment* protocols it can be the case that a user's specific transaction is pending which means the transaction has passed the checks maintainers do. However, sufficiently enough maintainers have not received a valid TI of user's counterparty so far. As a result, a pair of *sender-receiver* has not been generated on sufficiently enough maintainers' sides which implies that the user has not received α valid signature shares on its new account so far. In this case, upon receiving environment's instruction $(\mathtt{AbrTnx}, \mathsf{sid})$ for aborting the transaction, U sends an abort request AR to $\mathbb{M}$ which is of the form

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

in which

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

is a refreshed account of the user and $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ is a blinded version of it. $\mathsf{T} = g^{a^x}$ is the most recent tag used in the user's most recent transaction. π is a NIZK proof of knowledge. Specifically, the user U acts as follows:

1. Computes $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$. $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ is computed for U's refreshed account $\mathsf{acc}^{\mathsf{r}}$ using $\mathsf{PrepareBlindSign}$ algorithm and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is computed for U's previous account acc (for which it has consolidated signature $\sigma_{\mathbb{M}}$) using the $\mathsf{ProveSig}$ algorithm of the TBS scheme.
2. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x}, \mathtt{w})$, and obtains $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from it in which π is a NIZK proof for the statement

$$\mathtt{x} = \left(\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}\right)$$

We denote the randomness used to create $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ and $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ by $\mathsf{r}_{\mathsf{abr}}$. The witness of π is

$$\mathtt{w} = (\mathsf{acc}, \mathsf{r}_{\mathsf{abr}})$$

for the following relation $\mathtt{R}(\mathtt{x}, \mathtt{w})$:
(a) T is generated using a and x included in acc (for which user reveals $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$).
(b) $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is re-randomization of $\sigma_{\mathbb{M}}$ which is α different valid aggregated signature shares of maintainers on acc.
(c) $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$ is generated considering acc. Hence, $B^{\mathsf{r}} = B, S^{\mathsf{r}} = S, R^{\mathsf{r}} = R, \mathsf{sk}_{\mathsf{U}}^{\mathsf{r}} = \mathsf{sk}_{\mathsf{U}}, x^{\mathsf{r}} = x+1$ and $a^{\mathsf{r}} = a$ hold for $\mathsf{acc}^{\mathsf{r}}$.
(d) U knows the randomness $\mathsf{r}_{\mathsf{abr}}$.
3. Calls the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{AR})$ for $j = 1, \ldots, D$. Specifically, U calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_k, \mathsf{AR})$ $(1 \le k \le D-1)$ and waits for channel to send back $(\mathtt{Continue}, \mathsf{sid}, \mathsf{mid}_k)$, then U proceeds by calling $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{k+1}, \mathsf{AR})$.

Each maintainer M_j,

1. Receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{AR}, \mathsf{mid})$ from the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ and parses AR as $\left(\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi\right)$.
2. Ignores AR if at least one of the following items holds:
   (a) There already exists a transaction identifier t'_{id} (for an aborted transaction) in its ledger $\mathcal{L}_j$ where $\mathsf{T}' = \mathsf{T}$.
   (b) Upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$, it receives $(\mathtt{Verification}, \mathsf{sid}, 0)$.
3. Else, calls Byzantine Agreement $\mathcal{F}_{\mathsf{BA}}$ with input $(\mathtt{Agree}, \mathsf{sid}, d_j)$. The value of d_j is set to 1 if M_j sees a t_{id} (for issuance or payment transaction) in $\mathcal{L}_j$ that contains T. Else, d_j is set to 0. The output of $\mathcal{F}_{\mathsf{BA}}$ is $(\mathtt{Agreed}, \mathsf{sid}, Q)$.
4. If $Q = 1$ (which means at least one honest maintainer has saved a t_{id} in its ledger that contains T)[7]:
   (a) Each maintainer (e.g., M_i) who has already had *sender-receiver* pair saved in $\mathcal{L}_i$, sends it to others by calling the authenticated channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, (\mathsf{TI}_s, \mathsf{TI}_r))$ for $j = 1, \ldots, D \ \wedge \ j \neq i$.
   (b) Maintainers receive *sender-receiver* pair from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ (e.g., M_j receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_i, (\mathsf{TI}_s, \mathsf{TI}_r))$ sent by M_i).
   (c) Each maintainer verifies the validity of TI_s and TI_r by calling $\mathcal{F}_{\mathsf{SoK}}$ and ignore them if they are not valid.
   (d) Else, each maintainer e.g., M_k signs the new-blinded accounts of users $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$ and $\mathsf{acc}_r^{\mathsf{new},\mathfrak{B}}$ included in TI_s and TI_r to obtain $\sigma_{s,k}^{\mathsf{new},\mathfrak{B}}$ and $\sigma_{r,k}^{\mathsf{new},\mathfrak{B}}$.
   (e) M_k saves the associated $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ in $\mathcal{L}_k$ (if it has not already done so).
   (f) M_k calls the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_s, \sigma_{s,k}^{\mathsf{new},\mathfrak{B}})$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}_r, \sigma_{r,k}^{\mathsf{new},\mathfrak{B}})$.
   (g) M_k outputs $(\mathtt{TnxDone}, \mathsf{sid}, t_{\mathsf{id}})$ (to $\mathcal{Z}$).

   The sender U_s (resp. receiver U_r):
   (a) Receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_k, \sigma_{s,k}^{\mathsf{new},\mathfrak{B}})$ (resp. $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_k, \sigma_{r,k}^{\mathsf{new},\mathfrak{B}})$) for different k from the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$.
   (b) Unblinds α different maintainers' signature shares $\left\{\sigma_{s,k}^{\mathsf{new}}\right\}_{k=1}^{\alpha}$ (resp. $\left\{\sigma_{r,k}^{\mathsf{new}}\right\}_{k=1}^{\alpha}$) using $\mathsf{Unblind}$ algorithm.
   (c) Aggregates unblinded signature shares based on $\mathsf{TBS.Agg}$ algorithm to form one consolidated signature $\sigma_{s,\mathbb{M}}^{\mathsf{new}}$ (resp. $\sigma_{r,\mathbb{M}}^{\mathsf{new}}$).
   (d) Outputs $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_r, v)$ (to $\mathcal{Z}$) (resp. U_r outputs $(\mathtt{TnxDone}, \mathsf{sid}, \mathsf{U}_s, v)$).

   Having the signature on its refreshed account $\sigma_{\mathbb{M}}^{\mathsf{r}}$, the user can enter into new transaction by generating its new account $\mathsf{acc}^{\mathsf{new}}$ using $\mathsf{acc}^{\mathsf{r}}$ (rather than acc).
5. Else (meaning that $Q = 0$):
   (a) Maintainers who have already saved $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ in their ledgers where T_s or T_r equals to T delete it.
   (b) Each maintainer saves the aborted transaction identifier in its ledger which is of the form
   $$t_{\mathsf{id}} = (\mathsf{Aborted}, \mathsf{T})$$
   (c) Each maintainer e.g., M_j signs the refreshed-blinded account of the user $\mathsf{acc}_j^{\mathsf{r},\mathfrak{B}}$ to obtain $\sigma_j^{\mathsf{r},\mathfrak{B}}$.
   (d) M_j calls the sender anonymous channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}, \sigma_j^{\mathsf{r},\mathfrak{B}})$.
   (e) Outputs $(\mathtt{TnxAborted}, \mathsf{sid}, t_{\mathsf{id}})$ (to $\mathcal{Z}$).

   The user U:

[7] For the ease of understanding, in the protocol description, we address payment transactions. Issuance transactions are similar as we have $(\mathsf{TI}_{\mathsf{B}}, \mathsf{TI}_{\mathsf{U}})$ instead of $(\mathsf{TI}_s, \mathsf{TI}_r)$ thus the tag T is only checked against TI_{U}. Moreover, maintainers who have TI_{B}, they send the proof of receiving it from authenticated channel so that others make sure that TI_{B} has been sent by B.

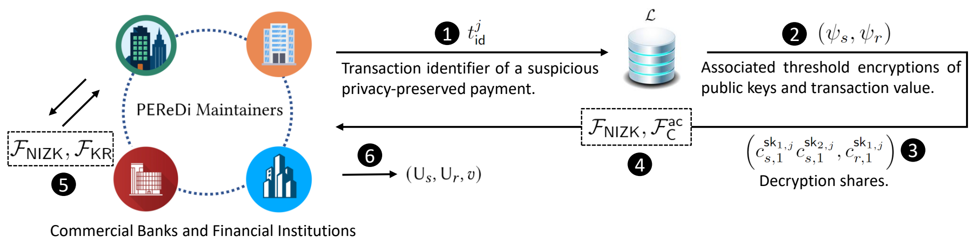

❶ t^j_{id}
Transaction identifier of a suspicious privacy-preserved payment.
$\mathcal{L}$
❷ (ψ_s, ψ_r)
Associated threshold encryptions of public keys and transaction value.
PEReDi Maintainers
$\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}^{\mathsf{ac}}_{\mathsf{C}}$
❹
$\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{KR}}$
❺
❻ $(\mathsf{U}_s, \mathsf{U}_r, v)$
$\left(c_{s,1}^{\mathsf{sk}_{1,j}} c_{s,1}^{\mathsf{sk}_{2,j}}, c_{r,1}^{\mathsf{sk}_{1,j}}\right)$ ❸
Decryption shares.
Commercial Banks and Financial Institutions

Fig. 5. *Privacy Revocation* Protocol

(a) Receives $(\texttt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{r},\mathfrak{B}})$ for different j from the sender anonymous channel $\mathcal{F}^{\mathsf{sa}}_{\mathsf{C}}$.
(b) Unblinds α different maintainers' signature shares $\{\sigma^{\mathsf{r}}_j\}_{k=1}^{\alpha}$ using $\mathsf{Unblind}$ algorithm.
(c) Aggregates unblinded signature shares based on $\mathsf{TBS.Agg}$ algorithm to form one consolidated signature $\sigma^{\mathsf{r}}_{\mathbb{M}}$.
(d) Outputs $(\texttt{TnxAborted}, \mathsf{sid})$ (to $\mathcal{Z}$).

Auditing. For achieving auditability, we make use of trust dispersal, cf. [1]. Users trust several authorities independently serving in different roles so that no single authority has unlimited power or authority over any user. Hence, for privacy revocation and user tracing, we also take advantage of threshold cryptography. For executing any type of auditing the participation of at least $\beta = t + 1$ maintainers is required where t is the maximum number of maintainer that can be corrupted by the adversary. As we have set the threshold of TBS scheme to $\alpha = 2t + 1$, always there exists at least $t + 1$ honest maintainers that have the transaction identifier t_{id} of a transaction saved in their ledgers.

This protocol parses as two sub-protocols *Privacy Revocation* and *Tracing* which are as follows.

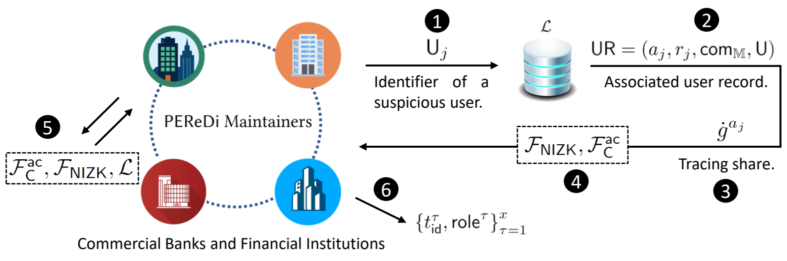
① *Privacy Revocation:* Given a privacy-preserved payment made by a specific *sender-receiver* pair, the audit committee revokes the privacy of the transaction by decrypting the ciphertexts and identifying transaction participants and value of the transaction. Upon receiving a message $(\texttt{RvkAnm}, \mathsf{sid}, t^j_{\mathsf{id}})$ (from $\mathcal{Z}$) the j-th maintainer M_j does the following as shown in Fig. 5:

1. Finds for the associated (ψ_s, ψ_r) saved in its ledger $\mathcal{L}_j$ for the given t^j_{id}[8].
2. Computes its decryption shares that are $c_{s,1}^{\mathsf{sk}_{1,j}}$ and $c_{s,1}^{\mathsf{sk}_{2,j}}$ for ψ_s, and $c_{r,1}^{\mathsf{sk}_{1,j}}$ for ψ_r.
3. Calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\texttt{Prove}, \mathsf{sid}, \mathtt{x}_j, \mathtt{w}_j)$, and obtains $(\texttt{Proof}, \mathsf{sid}, \pi_j)$ from it in which π_j is a NIZK proof for the statement

$$\mathtt{x}_j = \left(c_{s,1}, c_{r,1}, c_{s,1}^{\mathsf{sk}_{1,j}} c_{s,1}^{\mathsf{sk}_{2,j}}, c_{r,1}^{\mathsf{sk}_{1,j}}\right)$$

The witness of π_j is
$$\mathtt{w}_j = (\mathsf{sk}_{1,j}, \mathsf{sk}_{2,j})$$
for the following relation $\mathtt{R}(\mathtt{x}_j, \mathtt{w}_j)$: $\log_g \mathsf{pk}_{1,j} = \log_{c_{s,1}} c_{s,1}^{\mathsf{sk}_{1,j}}$, $\log_g \mathsf{pk}_{2,j} = \log_{c_{s,1}} c_{s,1}^{\mathsf{sk}_{2,j}}$, and $\log_g \mathsf{pk}_{1,j} = \log_{c_{r,1}} c_{r,1}^{\mathsf{sk}_{1,j}}$ hold.

[8] (For currency issuance transaction, given the fact that the sender is B, the cryptographic information saved for auditing only contains ψ. However, in the following, we describe the steps of *Privacy Revocation* protocol for a payment transaction and currency issuance transaction is similar.



1
U_j
Identifier of a suspicious user.
$\mathcal{L}$
2
$\mathsf{UR} = (a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{U})$
Associated user record.
$\dot{g}^{a_j}$
Tracing share.
3
$\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$
4
PEReDi Maintainers
5
$\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{L}$
6
$\{t_{\mathsf{id}}^{\tau}, \mathsf{role}^{\tau}\}_{\tau=1}^{x}$
Commercial Banks and Financial Institutions

Fig. 6. *Tracing* Protocol

4. Calls the authenticated channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with the input $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_i, (\mathtt{x}_j, \pi_j))$ for $i = 1, \ldots, D \ \wedge \ i \neq j$.
5. Considering the following equations

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

such that $|I| = \beta$ and λ is Lagrange coefficient, upon obtaining β valid decryption shares from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ (sent by other maintainers), computes pk_s, g^v, and pk_r. Validity of shares is checked by calling $\mathcal{F}_{\mathsf{NIZK}}$.
6. Calls $\mathcal{F}_{\mathsf{KR}}$ with $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{pk}_s)$ and $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{pk}_r)$ to retrieve 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 $\mathcal{F}_{\mathsf{KR}}$. Computes v from g^v[9].
7. Outputs $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$ (to $\mathcal{Z}$).

Ⓘ *Tracing:* Given a suspicious user's unique identifier, the audit committee traces all the privacy preserved transactions made by that user. First of all, they find user's record generated in *User Registration* protocol. Using secret shares of a, maintainers engage in a protocol explained below to compute all tracing tags of the user without revealing a. We will see that to achieve simulatability a should not be revealed. The maintainers mutually compute tracing tags such that the last computation results in a tag that does not exist in their ledgers. In this way, tracing authorities know the most recent transaction of U. As described in *Currency Issuance* and *Payment* protocols, all transactions contain tracing tag values of the form g^{a^x} in which a is user's (tracing tag) secret key and x is its transaction counter (note that for aborted transactions the user also increments x by one). The threshold for TBS is α which results the fact that always there exists at least β honest maintainers who have t_{id} of a specific transaction saved in their ledgers. However, the number of honest maintainers who have the whole t_{id} of all transactions of a specific user is not β (we do not use any agreement in the main body of payment). Hence, we have to make sure that at each step of threshold tag computation all maintainers are able to compute the tag T and afterwards check their ledgers to see if such a tag has already existed or not. They do so, by sending their next tag-computation shares in a provable way to others so that having β shares, the next tag is computed. This process is done up to the point that maintainers do not see the computed tag T in their ledgers so that there is no β shares for computing the next tag.

Upon receiving a message $(\texttt{Trace}, \mathsf{sid}, \mathsf{U}_j)$ the j-th maintainer M_j does the following as shown in Fig. 6:

1. Finds associated user record $\mathsf{UR} = (a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{U})$ saved in $\mathcal{L}_j$.

[9] Note that to have an 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 is having a limit on transaction value $v < V_{\mathsf{max}}$ which makes extracting v from g^v efficient for $\mathbb{M}$ in this sub-protocol.

2. Proves that the share contributed by itself to the threshold tag computation (equation at step 4) is consistent with j-th commitment $\mathsf{com}_j \in \mathsf{com}_{\mathbb{M}}$ (broadcasted at *User Registration* protocol to $\mathbb{M}$). More specifically, for the witness
$$\overline{\mathtt{w}}_j = (a_j, r_j)$$
and a given group element $\dot{g} = g^{a^e}$ where initially $e \leftarrow 0$ and the statement
$$\overline{\mathtt{x}}_j = (\mathsf{com}_j, \dot{g}^{a_j}, \dot{g})$$
it calls $\mathcal{F}_{\mathsf{NIZK}}$ with input $(\mathtt{Prove}, \mathsf{sid}, \overline{\mathtt{x}}_j, \overline{\mathtt{w}}_j)$, and receives $(\mathtt{Proof}, \mathsf{sid}, \overline{\pi}_j)$ from $\mathcal{F}_{\mathsf{NIZK}}$. Verification of $\overline{\pi}_j$ outputs $(\mathtt{Verification}, \mathsf{sid}, 1)$ if $\mathtt{R}(\overline{\mathtt{x}}_j, \overline{\mathtt{w}}_j)$ (which is as follows) holds: com_j is the commitment to the same a_j as what is in the maintainer's share $\dot{g}^{a_j}$.
3. Calls the authenticated channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with the input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_i, (\overline{\mathtt{x}}_j, \overline{\pi}_j))$ for $i = 1, \ldots, D \ \wedge \ i \neq j$.
4. Considering the following equation
$$\dot{g}^a = \prod_{j \in I} (\dot{g}^{a_j})^{\prod_{i \in I, i \neq j} (i/(i-j))}$$
where $|I| = \beta$ and $\prod_{i \in I, i \neq j} (i/(i-j))$ is a Lagrange coefficient, upon obtaining β valid tracing shares from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ (sent by other maintainers), computes $\dot{g}^a$. In other words, upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_i, (\overline{\mathtt{x}}_i, \overline{\pi}_i))$ from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ (sent by M_i), M_j calls $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \overline{\mathtt{x}}_i, \overline{\pi}_i)$.
M_j ignores the received message from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ if $\mathcal{F}_{\mathsf{NIZK}}$ outputs $(\mathtt{Verification}, \mathsf{sid}, 0)$.
Else, having β valid shares it computes $\dot{g}^a$ based on the equation above.
5. If there already exists a transaction identifier t_{id} (for issuance, payment, or aborted transaction) in its ledger $\mathcal{L}_j$ that includes $\dot{g}^a$ as a tag T, proceeds from step 2 with $e \leftarrow e + 1$ and records the associated t_{id} of computed T and role.
Else, sends a message to all maintainers via calling $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_i, (0, \dot{g}^a))$ for $i = 1, \ldots, D \ \wedge \ i \neq j$ (which means it has not seen $\dot{g}^a$ in $\mathcal{L}_j$).
6. If it receives $2t + 1$ messages of the form $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_i, (0, \dot{g}^a))$ (in which $\dot{g} = g^{a^e}$) from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$, outputs transaction identifiers and corresponding roles $(\mathtt{Traced}, \mathsf{sid}, \{t_{\mathsf{id}}^{\tau}, \mathsf{role}^{\tau}\}_{\tau=1}^{x})$ (to $\mathcal{Z}$), and aborts.
Else, waits for at least $t + 1$ messages of the form $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_i, (\overline{\mathtt{x}}_i, \overline{\pi}_i))$ (in which $\dot{g} = g^{a^{e+1}}$) from $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ and proceeds from step 2.

For currency issuance transaction t_{id} only contains tracing tag of receiver and for payment transaction it contains tracing tags of both sender and receiver. Based on the computed tracing tags each maintainer knows that the traced user was sender or receiver of the transaction for which t_{id} is retrieved (tag of the sender appears first in t_{id}). Hence, $\mathbb{M}$ output $(\mathtt{Traced}, \mathsf{sid}, \{t_{\mathsf{id}}^{\tau}, \mathsf{role}^{\tau}\}_{\tau=1}^{x})$ (to $\mathcal{Z}$) such that role can be *sender* or *receiver*. Note that given the $\{t_{\mathsf{id}}^{\tau}\}_{\tau=1}^{x}$ values, the counterparties of the suspicious user can be revealed using the *Privacy Revocation* protocol described above. To make tracing efficient, at *User Registration* protocol each user proves that x starts from 1 and then increments by one for each transaction at *Currency Issuance* and *Payment* protocols.

3.3 Know Your Transaction for Large Payments

Enforcing limits on transaction value and sum of all sent values are two general regulatory rules. The maximum allowed values for the former is denoted by V_{max} and for the latter is denoted by S_{max}. While such limits serve a purpose, a user may want to exceed them when a large payment is required. 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. In such circumstances, the user can exceed the mentioned thresholds up to the new limits V'_{max}

and S'_{max}. The new limit is computed by adding all values whose sources are proven as verified sources to the pre-defined general limit value. For instance, assuming the setting in which the user has accumulated funds during a long period of time and now it wants to spend them at once (e.g., in the process of buying a property) this will result in a transaction value far exceeding V_{max} (note we assume $B_{\mathsf{max}} \gg V_{\mathsf{max}}$, as otherwise there is no need for the mechanism of this paragraph). The user saves the relevant information of transactions for which it will make a claim. The user points to transaction identifiers of associated transactions in which it has received money during a period of time from an acceptable source. The user can make such a claim to $\mathbb{M}$ who will facilitate the excess thresholds. We denote the sum of all values for which the user makes a claim by δ, then following the above explanation $V'_{max} = V_{\mathsf{max}}+\delta$ and $S'_{max} = S_{\mathsf{max}}+\delta$ hold. The user points to the relevant transaction identifier of l transactions $\{t^{\tau}_{\mathsf{id}}\}^{l}_{\tau=1}$ that contain associated threshold encryptions $\{(\psi_s, \psi_r)^{\tau}\}^{l}_{\tau=1}$. Given the fact that the user knows the randomness of threshold encryptions it provides the proof of knowledge and proves that sum of all values in threshold encryptions equals to δ. Moreover, using the relevant random values it convinces $\mathbb{M}$ regarding the sender of transactions. More generally, $\mathbb{M}$ can recognize a third party auditor who will verify the user's claim and in this case the user needs only to present a certification of this transaction by that auditor.

4 Proof of PEReDi Security

Our main theorem is given below.

Theorem 1. *Assuming that Pedersen commitments are perfectly hiding, Pointcheval-Sanders signatures are* EUF-CMA *secure in the random oracle model, ElGamal encryption is* IND-CPA *secure, and the d-strong Diffie-Hellman problem is hard, 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 $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ with advantage better than $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}} + p_c \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} + p_u \cdot \mathsf{Adv}^{\mathsf{d\text{-}sDDH}}_{\mathcal{A}}$ in the $\{\mathcal{F}_{\mathsf{KR}}, \mathcal{F}_{\mathsf{C}}, \mathcal{F}_{\mathsf{BA}}, \mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{SoK}}\}$-hybrid model with static corruptions in the presence of arbitrary number of malicious users, up to t malicious maintainers out of $D = 3t + 1$ total maintainers and a potentially malicious central bank.*

In the remaining of the section, we provide a sketch of the security proof.

4.1 Sketch of Proof of Theorem 1.

We denote the real-world protocol and adversary by Π_{PEReDi} and $\mathcal{A}$ respectively. The simulator $\mathcal{S}$ described in a detailed manner in the Appendix C makes the view of real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{PEReDi}},\mathcal{A},\mathcal{Z}}$ and ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$ for any PPT environment $\mathcal{Z}$ indistinguishable. Session identifier denoted by sid is chosen by $\mathcal{Z}$. The simulator $\mathcal{S}$ internally runs a version of Π_{PEReDi} and makes the view of the dummy adversary $\mathcal{A}$ in the ideal world 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 of the network. $\mathcal{S}$ reads these corruption messages and tells $\mathcal{F}_{\mathsf{CBDC}}$ which parties are corrupted by sending the message $(\mathtt{Corrupt}, \mathsf{sid}, \mathsf{P})$, the simulator $\mathcal{S}$ also stores the corrupted parties identifiers. $\mathcal{S}$ internally emulates the functionalities $\mathcal{F}_{\mathsf{KR}}, \mathcal{F}_{\mathsf{C}}, \mathcal{F}_{\mathsf{BA}}, \mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{B}}$ and $\mathcal{F}_{\mathsf{SoK}}$. $\mathcal{A}$ instructs corrupted parties arbitrarily. $\mathcal{S}$ interacts with $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of 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 $\mathcal{F}_{\mathsf{CBDC}}$.

Sequence of games 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 $\Pr\left[\mathsf{Game}^{\mathsf{i}}\right]$ 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 $\mathcal{F}^0_{\mathsf{CBDC}}$ and the associated simulator $\mathcal{S}^0$ and gradually go toward the main functionality $\mathcal{F}_{\mathsf{CBDC}}$ and the simulator $\mathcal{S}$.

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 in Game^1 we change w-th honest maintainer's blind signature share on U's account to $\sigma^{\mathfrak{B}}_w$ which is simulated by $\mathcal{S}^1$. To do so, in this game, $\mathcal{S}^1$ selects the secret signing key of non-threshold Pointcheval-Sanders signature and then computes non-threshold signature $\sigma_{\mathbb{M}} = (h, s)$. Note that after Game^1 the simulator $\mathcal{S}^i$ for $i \geq 2$ never uses the secret signing key of non-threshold signature scheme (as we will see it receives a non-threshold signature from the challenger of non-threshold signature's unforgeability game –see Def. 6– in the associated reduction). Also, by selecting the secret key of malicious maintainers, $\mathcal{S}^1$ computes partial blind signatures of malicious maintainers that are $\sigma^{\mathfrak{B}}_t$ for $t \in \mathcal{C}$. As a result of having $\sigma_{\mathbb{M}}$ and $\sigma^{\mathfrak{B}}_t$ for $t \in \mathcal{C}$ the simulator $\mathcal{S}^1$ computes honest maintainers' signature shares $\sigma^{\mathfrak{B}}_w$ for $w \in \mathcal{H}$ as follows.

When $\mathcal{A}$ initiates the protocol in which it requests a signature on the blinded account $\mathsf{acc}^{\mathfrak{B}}$, $\mathcal{S}^1$ who 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, $\mathcal{S}^1$ computes $\sigma_{\mathbb{M}}$. $\mathcal{S}^1$ selects the secret keys of malicious maintainers[10] and computes associated public keys. $\mathcal{S}^1$ uses Lagrange interpolation to compute public keys of $\mathsf{M}_w \in \mathbf{H}$ using computed public keys for $\mathsf{M}_t \in \mathbf{C}$ and public key of non-threshold signature. Hence, all public keys are consistent with the public key of non-threshold signature. First of all, $\mathcal{S}^1$ computes blind signature shares for $\forall \mathsf{M}_t \in \mathbf{C}$ using selected $\mathsf{sk}_t = \left(x_t, \{y_{t,\tau}\}^q_{\tau=1}\right)$ to obtain $\sigma^{\mathfrak{B}}_t = \left(h, h^{x_t} \prod^q_{\tau=1} \mathsf{com}_\tau^{y_{t,\tau}}\right)$. As described above, $\mathcal{S}^1$ has extracted witness of NIZK or SoK proofs, hence, it knows $\{o_\tau\}^q_{\tau=1}$ which lets him to compute unblinded signature shares in the following way: $\sigma_t = \left(h, c \prod^q_{\tau=1} \beta_{t,\tau}^{-o_\tau}\right) = (h, s_t)$ in which $s_t = h^{x_t} \prod^q_{\tau=1} h^{m_\tau y_{t,\tau}}$. Then, $\mathcal{S}^1$ computes unblinded signature shares for $\forall \mathsf{M}_w \in \mathbf{H}$ as follows (note that $k \neq 0$ as 0 does not exist in the corrupted maintainers' indexes $\mathcal{C}$).

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

Having the extracted witness $\{o_\tau\}^q_{\tau=1}$, the simulator computes blind signature shares for $\forall \mathsf{M}_w \in \mathbf{H}$ using the computed σ_w as follows: $\sigma^{\mathfrak{B}}_w = \left(h, \prod^q_{\tau=1} s_w \beta_{w,\tau}^{o_\tau}\right)$.

As a result, in this game we changed w-th honest maintainer's blind signature share on U's account to $\sigma^{\mathfrak{B}}_w$ which is simulated by $\mathcal{S}^1$ as described above. Based on $\mathsf{Unblind}$ algorithm which is run by $\mathcal{A}$, the unblinded signature is computed as follows $\sigma_w = \left(h, c \prod^q_{\tau=1} \beta_{w,\tau}^{-o_\tau}\right)$ for the c simulated by $\mathcal{S}^4$ as $c = \prod^q_{\tau=1} s_w \beta_{w,\tau}^{o_\tau}$. As a result, we have $\sigma_w = (h, s_w)$ that passes the verification algorithm $e\left(h, \tilde{\alpha}_w \prod^q_{\tau=1} \tilde{\beta}_{w,\tau}^{m_\tau}\right) = e(s_w, \tilde{g})$ which means that the following equation holds.

$$\Pr\left[\mathsf{Game}^1\right] = \Pr\left[\mathsf{Game}^0\right]$$

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

[10] Note that $\mathcal{Z}$ triggers (ideal-word) adversary to corrupt parties with a message $(\texttt{Corrupt}, \mathsf{sid}, \mathsf{P})$, therefore, $\mathcal{S}^1$ has already known $\mathcal{C}$.

Associated Reduction (existential unforgeability of signature). If $\mathcal{A}$ forges threshold blind signature TBS used in our construction, it can be used to construct another $\mathcal{A}'$ who breaks unforgeability property (see Def. 6) of non-threshold Pointcheval-Sanders signature used in threshold blind signature TBS scheme. The partial blind signatures of all honest maintainers $\sigma_w^{\mathfrak{B}}$ for $\forall w \in \mathcal{H}$ can be reconstructed from the partial blind signatures of malicious maintainers that are $\sigma_t^{\mathfrak{B}}$ 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^1 in which $\mathcal{S}^1$ selected the secret signing key of non-threshold signature) using the Lagrange interpolation for the other shares. We omit writing the details as the algorithm is similar to what $\mathcal{S}^1$ does in Game^1 except the fact 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 $\mathcal{A}$ which are partial signatures that the honest maintainers are contributed which implies that $\mathcal{A}$ cannot forge messages in the threshold setting of our construction unless $\mathcal{A}$ forges it in the non-threshold one. In other words, for $\mathcal{Z}$, Game^2 is the same as running a threshold signature TBS with real-world maintainers rather than simulated maintainers by $\mathcal{A}'$. Hence, if $\mathcal{A}$ forges in the real world, it will forge in this threshold setting and $\mathcal{A}'$ uses this forgery as a forgery for the non-threshold scheme. As a result, TBS is simulatable that together with unforgeability property of non-threshold Pointcheval-Sanders signature makes TBS unforgeable in our construction's setting.

Therefore, under the unforgeability property of non-threshold Pointcheval-Sanders signature the following inequality holds (see Def. 6 for the definition of $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$):

$$\left|\Pr\left[\mathsf{Game}^2\right] - \Pr\left[\mathsf{Game}^1\right]\right| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$$

Game^3: Same as Game^2 except that $\mathcal{S}^3$ computes $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ for the honest user without knowing the account values of the user. In the real world, having the consolidated signature $\sigma_{\mathbb{M}} = (h, s)$, $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ is computed as $(h', s') = \left(h^{r'}, s^{r'} h^{r'r}\right)$ such that $r \xleftarrow{\$} \mathbb{Z}_p$ and $r' \xleftarrow{\$} \mathbb{Z}_p$. Assume a random value as η, set $h = g^{\eta}$ by programming the random oracle. Hence, we have

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

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

Also, in the real world in $\mathsf{ProveSig}$ algorithm the user U computes κ as well which is of the form:

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

$\mathcal{S}^3$ randomly selects $u \xleftarrow{\$} \mathbb{Z}_p$ and $u' \xleftarrow{\$} \mathbb{Z}_p$. Then, sets $h' \leftarrow g^{u'}$, $s' \leftarrow g^{uu'}$ and hence sets $\sigma_{\mathbb{M}}^{\mathsf{Rnd}} \leftarrow (g^{u'}, g^{uu'})$. Finally, sets $\kappa \leftarrow \tilde{g}^{u}$. Computed values pass the verification $e(h', \kappa) = e(s', \tilde{g})$ as we have $e\left(g^{u'}, \tilde{g}^{u}\right) = e\left(g^{uu'}, \tilde{g}\right)$. As $d = (x + \sum_{\tau=1}^{q}(m_\tau y_\tau) + r)$ and $d' = \eta r'$ are random values, they match the distribution of u and u' which concludes the fact that

$$\Pr\left[\mathsf{Game}^3\right] = \Pr\left[\mathsf{Game}^2\right]$$

Game^4: Same as Game^3 except that in Game^4, $\mathcal{S}^4$ simulate the decryption shares $c_{s,1}^{\mathsf{sk}_{1,w}}$ and $c_{s,1}^{\mathsf{sk}_{2,w}}$ (for ψ_s), and $c_{r,1}^{\mathsf{sk}_{1,w}}$ (for ψ_r), of honest maintainer M_w, $w \in \mathcal{H}$, and for s-th and r-th honest users' threshold encryptions using the values of a non-threshold ElGamal encryption scheme. In this game, plaintexts of ψ_s and ψ_r are the same as real-world values (in $\mathsf{Game}_{\mathsf{i}}^4$ for $i \geq 1$, we will change plaintexts to dummy values selected by the simulator $\mathcal{S}_i^4$). To do so, in this game, $\mathcal{S}^4$ selects the secret decryption keys of non-threshold ElGamal encryption $\mathsf{sk}_{1,\mathbb{M}}$ and $\mathsf{sk}_{2,\mathbb{M}}$ (note that from Game^4 onward 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 Def. 5, associated with Game^5). Then, $\mathcal{S}^4$ computes $c_s = (c_{s,1}, c_{s,2}, c_{s,3}) = \left(g^{\rho_s}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_s} \cdot \mathsf{pk}_s, \mathsf{pk}_{2,\mathbb{M}}^{\rho_s} g^v\right)$ and $c_r = (c_{r,1}, c_{r,2}) = \left(g^{\rho_r}, \mathsf{pk}_{1,\mathbb{M}}^{\rho_r} \cdot \mathsf{pk}_r\right)$. The plaintexts of threshold encryptions are retrieved as $\mathsf{pk}_s = c_{s,2}/\prod_{j\in I} c_{s,1}^{\mathsf{sk}_{1,j}\lambda_{1,j}}$, $\mathsf{pk}_r = c_{r,2}/\prod_{j\in I} c_{r,1}^{\mathsf{sk}_{1,j}\lambda_{1,j}}$ and $g^v = c_{s,3}/\prod_{j\in I} c_{s,1}^{\mathsf{sk}_{2,j}\lambda_{2,j}}$. Now, $\mathcal{S}^4$ should simulate honest maintainers' decryption shares such that decrypted values become $\mathsf{pk}_s, \mathsf{pk}_r$ and g^v respectively that are consistent with $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$ received from the leakage of $\mathcal{F}_{\mathsf{CBDC}}^4$.

$\mathcal{S}^4$ computes $c_{s,2}/\mathsf{pk}_s$ which results in $\mathsf{pk}_{1,\mathbb{M}}^{\rho_s} = \left(g^{\mathsf{sk}_{1,\mathbb{M}}}\right)^{\rho_s} = c_{s,1}^{sk_{1,\mathbb{M}}}$ that is used in the computation of honest maintainers' shares in the following equation. $\mathcal{S}^4$ computes w-th honest maintainer's decryption share as follows with knowing malicious maintainers shares $c_{s,1}^{\mathsf{sk}_{1,t}}$ and $c_{s,1}^{\mathsf{sk}_{2,t}}$ for ψ_s, and $c_{r,1}^{\mathsf{sk}_{1,t}}$ for ψ_r for $\forall t \in \mathcal{C}$ such that $|\mathcal{C}| \leq \beta - 1 = t$ (note that in the equation below $k \neq 0$ as 0 does not exist in the corrupted maintainers' indexes $\mathcal{C}$).

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

$\mathcal{S}^4$ computes $c_{r,2}/\mathsf{pk}_r$ which results in $\mathsf{pk}_{1,\mathbb{M}}^{\rho_r} = \left(g^{\mathsf{sk}_{1,\mathbb{M}}}\right)^{\rho_r} = c_{r,1}^{sk_{1,\mathbb{M}}}$ then computes $c_{r,1}^{\mathsf{sk}_{1,w}}$ as follows:

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

$\mathcal{S}^4$ computes $c_{s,3}/g^v$ which results in $\mathsf{pk}_{2,\mathbb{M}}^{\rho_s} = \left(g^{\mathsf{sk}_{2,\mathbb{M}}}\right)^{\rho_s} = c_{s,1}^{sk_{2,\mathbb{M}}}$ then computes $c_{s,1}^{\mathsf{sk}_{2,w}}$ as follows:

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

$\mathcal{F}_{\mathsf{NIZK}}$ emulation allows $\mathcal{S}^4$ to provide faked proofs about the contribution of honest maintainers which is unconditionally secure. Moreover, changing honest maintainers' decryption shares is information theoretically indistinguishable. Moreover, the simulated decryption shares work in the threshold decryption computation (as shown above), thus, we have the following equation:

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

Game^5: Same as Game^4 except that in Game^5 we change all plaintexts of threshold encryptions to dummy values selected by $\mathcal{S}^5$. Hence, Game^5 equals to Game^4 except the fact that $\mathcal{S}^5$ 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 simulated in a way that computation of $c_{s,2}/\prod_{j\in I} c_{s,1}^{\mathsf{sk}_{1,j}\lambda_{1,j}}$, $c_{r,2}/\prod_{j\in I} c_{r,1}^{\mathsf{sk}_{1,j}\lambda_{1,j}}$ and $c_{s,3}/\prod_{j\in I} c_{s,1}^{\mathsf{sk}_{2,j}\lambda_{2,j}}$ result in $\mathsf{pk}_s, \mathsf{pk}_r$ and g^v respectively that are consistent with $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$ received from $\mathcal{F}_{\mathsf{CBDC}}^5$ (rather than dummy values $\mathsf{pk}_s^*, \mathsf{pk}_r^*$ and g^{*v}). We ignore writing the details as it is similar to Game^4. Hence, any difference between Game^5 and Game^4 is because of breaking the IND-CPA security of threshold encryption used in our construction which allows us to bound the probability that $\mathcal{Z}$ distinguishes Game^5 from Game^4 as follows.

We define $\mathsf{Game}_0^4 = \mathsf{Game}^4$ and p_c as the upper bound on the number of all ciphertexts of honest users. Also, lets define Game_1^4 as a game similar to Game_0^4 except in Game_1^4 we change the plaintext of first ciphertext from the real-world value to the ideal-world dummy value. The reduction between Game_0^4 and Game_1^4 is similar to the described reduction below so that any difference between Game_0^4 and Game_1^4 is upper bounded by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$ (see

Def. 5 for the definition of $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$). Similarly, we change the plaintexts of ciphertexts of i-th honest user to dummy values and finally we do the same for the last ciphertext of last honest user such that in $\mathsf{Game}^4_{\mathsf{p_c}}$ (which equals to Game^5) all ciphertexts are generated from dummy plaintexts. The reduction between $\mathsf{Game}^4_{\mathsf{p_c}-1}$ and $\mathsf{Game}^4_{\mathsf{p_c}}$ is similar to the described reduction below so that the any difference between $\mathsf{Game}^4_{\mathsf{p_c}-1}$ and $\mathsf{Game}^4_{\mathsf{p_c}}$ is upper bounded by $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$.

Associated Reduction Between $\mathsf{Game}^4_{\mathsf{i}-1}$ and $\mathsf{Game}^4_{\mathsf{i}}$ (IND-CPA security of encryption). If $\mathcal{A}$ distinguishes $\mathsf{Game}^4_{\mathsf{i}-1}$ and $\mathsf{Game}^4_{\mathsf{i}}$ it can be used to construct another $\mathcal{A}'$ who breaks IND-CPA security of non-threshold ElGamal encryption used in threshold encryption scheme. The decryption shares of all honest maintainers $c_{s,1}^{\mathsf{sk}_{1,w}}$ and $c_{s,1}^{\mathsf{sk}_{2,w}}$ for ψ_s, and $c_{r,1}^{\mathsf{sk}_{1,w}}$ for ψ_r for $\forall w \in \mathcal{H}$ can be reconstructed from the decryption shares of malicious maintainers that are $c_{s,1}^{\mathsf{sk}_{1,t}}$ and $c_{s,1}^{\mathsf{sk}_{2,t}}$ for ψ_s, and $c_{r,1}^{\mathsf{sk}_{1,t}}$ 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 the Lagrange interpolation for the other shares. We omit writing the details as the algorithm is similar to the described algorithm in Game^4 except the fact that $\mathcal{A}'$ obtains the non-threshold encryptions c_s and c_r from the challenger of IND-CPA game. Hence, given the non-threshold ciphertexts, $\mathcal{A}'$ simulates the entire view of $\mathcal{A}$ which are decryption shares that the honest maintainers contribute which implies that $\mathcal{A}$ cannot distinguish $\mathsf{Game}^4_{\mathsf{i}-1}$ from $\mathsf{Game}^4_{\mathsf{i}}$ unless $\mathcal{A}$ distinguishes non-threshold ciphertexts c_s and c_r generated by real-world values as plaintexts from ciphertexts generated by ideal-world dummy values as plaintexts. In other words, for $\mathcal{Z}$, $\mathsf{Game}^4_{\mathsf{i}}$ is the same as running a threshold encryption with real-world maintainers rather than simulated maintainers by $\mathcal{A}'$. Hence if $\mathcal{Z}$ distinguishes $\mathsf{Game}^4_{\mathsf{i}-1}$ from $\mathsf{Game}^4_{\mathsf{i}}$, $\mathcal{A}'$ uses this to win the non-threshold encryption scheme's IND-CPA game. As a result, threshold encryption is simulatable that together with IND-CPA property of non-threshold encryption scheme makes threshold encryption IND-CPA secure in our construction's setting.

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

$$\left|\Pr\left[\mathsf{Game}^5\right] - \Pr\left[\mathsf{Game}^4\right]\right| \le p_c \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}$$

Game^6: Same as Game^5, except that for honest maintainer M_w the simulator $\mathcal{S}^6$ computes the tracing tag share $\dot{g}^{a_w}$ for tracing 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 (in $\mathsf{Game}^6_{\mathsf{i}}$ for $i \ge 1$, we will change these tags to dummy values selected by the simulator $\mathcal{S}^6_i$). $\mathcal{S}^6$ knows $\{g^{z_\tau}\}_{\tau=1}^{x}$ from the transaction identifiers leaked from $\mathcal{F}^6_{\mathsf{CBDC}}$ which are $(\mathtt{Traced}, \mathsf{sid}, \{t^\tau_{\mathsf{id}}, \mathsf{role}^\tau\}_{\tau=1}^{x})$ and computes M_w's first share as follows ($\dot{g} = g$):

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

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

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

Changing honest maintainers' shares is information theoretically indistinguishable and emulating $\mathcal{F}_{\mathsf{NIZK}}$ (which is unconditionally secure) allows $\mathcal{S}^6$ to provide faked proofs. As a result, we have

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

Game^7: Same as Game^6 except that in Game^7 we change the tracing tags of honest users to the dummy values selected by $\mathcal{S}^7$. Hence, Game^7 equals to Game^6 except the fact that

$\mathcal{S}^7$ computes tracing tags and submits them to $\mathcal{F}^7_{\mathsf{CBDC}}$ as part of transaction identifiers in *Currency Issuance* and *Payment* protocols. However, the tracing tag shares of honest maintainers simulated in a way that computation of tags result in $\{g^{z_\tau}\}_{\tau=1}^{x}$ that are consistent with transaction identifiers t_{id} leaked from $\mathcal{F}^7_{\mathsf{CBDC}}$. We ignore writing the details as it is similar to Game^6. Hence, any difference between Game^7 and Game^6 is because of distinguishing g^{a^x} values from g^z values which allows us to bound the probability that $\mathcal{Z}$ distinguishes Game^7 from Game^6 as follows.

We define $\mathsf{Game}^6_0 = \mathsf{Game}^6$ and p_u as the upper bound on the number of all honest users. Also, lets define Game^6_1 as a game similar to Game^6_0 except in Game^6_1 we change the tracing tags of first honest user from real-world values to ideal-world dummy values. The reduction between Game^6_0 and Game^6_1 is described below so that any difference between Game^6_0 and Game^6_1 is upper bounded by $\mathsf{Adv}^{\mathsf{d\text{-}sDDH}}_{\mathcal{A}}$ (see Def. 2 for the definition of $\mathsf{Adv}^{\mathsf{d\text{-}sDDH}}_{\mathcal{A}}$). Similarly, we change the tracing tags of i-th honest user to the dummy values and finally we do the same for the last honest user such that in $\mathsf{Game}^6_{\mathsf{p_u}}$ (which equals to Game^7) all tracing tags are dummy values. The reduction between $\mathsf{Game}^6_{\mathsf{p_u}-1}$ and $\mathsf{Game}^6_{\mathsf{p_u}}$ is similar to the described reduction below so that the any difference between $\mathsf{Game}^6_{\mathsf{p_u}-1}$ and $\mathsf{Game}^6_{\mathsf{p_u}}$ is upper bounded by $\mathsf{Adv}^{\mathsf{d\text{-}sDDH}}_{\mathcal{A}}$.

Associated Reduction Between $\mathsf{Game}^6_{\mathsf{i}-1}$ and $\mathsf{Game}^6_{\mathsf{i}}$ (hardness of d-sDDH). If $\mathcal{A}$ distinguishes $\mathsf{Game}^6_{\mathsf{i}-1}$ from $\mathsf{Game}^6_{\mathsf{i}}$ it can be used to construct another $\mathcal{A}'$ who breaks hardness of 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 that 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 the Lagrange interpolation for the other shares. We omit writing the details as the algorithm is similar to the described algorithm in Game^6. Hence, $\mathcal{A}'$ simulates the entire view of $\mathcal{A}$ which are tracing tag computation shares that the honest maintainers contribute which implies that $\mathcal{A}$ cannot distinguish $\mathsf{Game}^6_{\mathsf{i}-1}$ from $\mathsf{Game}^6_{\mathsf{i}}$ unless $\mathcal{A}$ breaks the hardness of d-strong Diffie-Hellman problem. Hence, if $\mathcal{Z}$ distinguishes $\mathsf{Game}^6_{\mathsf{i}-1}$ from $\mathsf{Game}^6_{\mathsf{i}}$, $\mathcal{A}'$ uses this to win the indistinguishability game of d-strong Diffie-Hellman problem. As a result, under hardness of d-strong Diffie-Hellman problem the following inequality holds:

$$\left|\Pr\left[\mathsf{Game}^7\right] - \Pr\left[\mathsf{Game}^6\right]\right| \leq p_u \cdot \mathsf{Adv}^{\mathsf{d\text{-}sDDH}}_{\mathcal{A}}$$

As $\mathcal{F}^7_{\mathsf{CBDC}} = \mathcal{F}_{\mathsf{CBDC}}$ and $\mathcal{S}^7 = \mathcal{S}$ which means Game^7 corresponds to the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{CBDC}},\mathcal{S},\mathcal{Z}}$, we argue that 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 or, in other worlds, 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 $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}} + p_c \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} + p_u \cdot \mathsf{Adv}^{\mathsf{d\text{-}sDDH}}_{\mathcal{A}}$ that concludes the sketch of security proof.

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. 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)
5. 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`

6. 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
7. Bank, E.C.: Exploring anonymity in central bank digital currencies (2019)
8. 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
9. Bindseil, U.: Tiered CBDC and the Financial System (2020), Available in this Link
10. 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
11. Bjerg, O.: Designing new money-the policy trilemma of central bank digital currency (2017)
12. 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
13. 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)
14. 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, `https://doi.org/10.1007/11426639_18`
15. 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
16. 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
17. 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
18. Canetti, R.: Security and composition of cryptographic protocols: A tutorial. Cryptology ePrint Archive, Report 2006/465 (2006), `https://eprint.iacr.org/2006/465`
19. 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, `https://doi.org/10.1007/BFb0054154`
20. Chase, M., Lysyanskaya, A.: On signatures of knowledge. Cryptology ePrint Archive, Report 2006/184 (2006), `https://eprint.iacr.org/2006/184`
21. 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)
22. Chaum, D., Grothoff, C., Moser, T.: How to issue a central bank digital currency. arXiv preprint arXiv:2103.00254 (2021)
23. 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
24. Danezis, G., Meiklejohn, S.: Centrally banked cryptocurrencies. In: NDSS 2016. The Internet Society (Feb 2016)
25. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (Aug 1984)
26. 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)

27. 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)
28. 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
29. 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)
30. Kumhof, M., Noone, C.: Central bank digital currencies-design principles and balance sheet implications (2018)
31. Pedersen, T.P.: A threshold cryptosystem without a trusted party (extended abstract) (rump session). In: Davies, D.W. (ed.) EUROCRYPT'91. LNCS, vol. 547, pp. 522–526. Springer, Heidelberg (Apr 1991). https://doi.org/10.1007/3-540-46416-6_47
32. Pointcheval, D., Sanders, O.: Short randomizable signatures. Cryptology ePrint Archive, Paper 2015/525 (2015), `https://eprint.iacr.org/2015/525`
33. Rial, A., Piotrowska, A.M.: Security analysis of coconut, an attribute-based credential scheme with threshold issuance. Cryptology ePrint Archive (2022)
34. Saberhagen, N.: Cryptonote v. 2.0. https://cryptonote.org/whitepaper.pdf (October 17 2013)
35. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
36. 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)
37. Tomescu, A., Bhat, A., Applebaum, B., Abraham, I., Gueta, G., Pinkas, B., Yanai, A.: Utt: Decentralized ecash with accountable privacy. Cryptology ePrint Archive (2022)
38. 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`
39. 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
40. Yao, Q.: A systematic framework to understand central bank digital currency. Science China Information Sciences **61**(3), 1–8 (2018)

A Cryptographic Schemes

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

A.1 ElGamal Encryption Scheme

The security of ElGamal encryption scheme [25] depends on the hardness of the discrete logarithm problem. It contains the three following algorithms:

1. KeyGen: Let p be a large prime and g be a generator of $\mathbb{Z}_p^*$. The receiver (maintainer in Π_{PEReDi}) randomly chooses the secret key $\mathsf{sk} \xleftarrow{\$} \mathbb{Z}_p^*$ and computes $\mathsf{pk} = g^{\mathsf{sk}} \bmod p$. Then, the receiver publishes public parameters (g, p, pk) while keeping sk secret.
2. $\mathsf{Encryption}$: For encrypting a message $m \in \mathbb{Z}_p$, the user randomly chooses an integer $k \xleftarrow{\$} \mathbb{Z}_p^*$ and set the ciphertext $c = (c_1, c_2) = \left(g^k, \mathsf{pk}^k m\right) \bmod p$.
3. $\mathsf{Decryption}$: Given $c = (c_1, c_2)$, the receiver computes the message $m = c_2/{c_1}^{\mathsf{sk}} \bmod p$.

A.2 Threshold ElGamal Encryption Scheme

Using a distributed key generation protocol introduced in [31] or [28] the secret key of threshold ElGamal encryption scheme is generated. We denote that sk_j is the secret key of decryption for j-th maintainer and $\mathsf{pk}_j = g^{\mathsf{sk}_j}$ is the corresponding public key share. Hence, we have $\mathsf{sk} = \sum_{j \in I} \mathsf{sk}_j \lambda_j$ such that λ_j is the Lagrange coefficient for the j-th share and $|I| = \beta$. To decrypt an ElGamal ciphertext $c = (c_1, c_2) = \left(g^k, \mathsf{pk}^k m\right)$, j-th maintainer first publishes $c_1^{\mathsf{sk}_j}$, and then generates a proof that $\log_g \mathsf{pk}_j = \log_{c_1} c_1^{\mathsf{sk}_j}$ hold to prove its honest contribution. Finally, the plaintext m can be retrieved as $m = c_2 / \prod_{j \in I} c_1^{\mathsf{sk}_j \lambda_j}$.

In our construction, assuming the first message as user's public key $m_1 = \mathsf{pk}_\mathsf{U}$ and the second message as $m_2 = g^v$ in which v is the value of transaction, we extend the ciphertext such that $c = (c_1, c_2, c_3) = \left(g^k, \mathsf{pk}_{1,\mathbb{M}}^k m_1, \mathsf{pk}_{2,\mathbb{M}}^k m_2\right)$. Hence, 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, we have $\mathsf{sk}_{1,\mathbb{M}} = \sum_{j \in I} \mathsf{sk}_{1,j} \lambda_{1,j}$ and $\mathsf{sk}_{2,\mathbb{M}} = \sum_{j \in I} \mathsf{sk}_{2,j} \lambda_{2,j}$, and their associated public keys are $\mathsf{pk}_{1,\mathbb{M}} = g^{\mathsf{sk}_{1,\mathbb{M}}}$ 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 $c_1^{\mathsf{sk}_{2,j}}$. It also generates a proof 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 prove its honest contribution. The messages m_1 and m_2 are retrieved by computing $m_1 = c_2 / \prod_{j \in I} c_1^{\mathsf{sk}_{1,j} \lambda_{1,j}}$ and $m_2 = c_3 / \prod_{j \in I} c_1^{\mathsf{sk}_{2,j} \lambda_{2,j}}$ respectively.

A.3 Secret Sharing

Shamir introduced (D, β)-threshold scheme [35]. A β-out-of-D threshold secret sharing of a secret message m is sharing m into D parts such that any β shares together can be used to reconstruct the secret m. However, fewer shares provide no information at all about m.

Definition 1. *A (D, β)-secret sharing scheme* $\mathsf{SSH} = (\mathsf{SSH.Share}, \mathsf{SSH.Agg})$ *consists of the following algorithms:*

1. $\{m_i\}_{i=1}^D \xleftarrow{\$} \mathsf{SSH.Share}^{D,\beta}(m)$*: Upon receiving m as input it outputs D secret shares such that m_i denotes the i-th share of m.*
2. $m^* \leftarrow \mathsf{SSH.Agg}^{D,\beta} \{m_i\}_{i \in I}$*: Upon receiving β shares ($|I| = \beta$) as input it outputs a reconstructed secret m^*.*

Moreover, SSH *generally satisfies Correctness and Privacy. The former guarantees that* $\Pr[m^* = m] = 1$ *and the latter requires that given $\beta - 1$ or fewer shares of either m_1 or m_0 no PPT adversary $\mathcal{A}$ can guess which message was shared with probability better than* $\frac{1}{2} + \mathsf{negl}(\lambda)$.

A.4 Pointcheval-Sanders Signature Scheme

Pointcheval-Sanders signature scheme [32] is existentially unforgeable and randomizable which consists of the following algorithms:

1. $\mathsf{KeyGen}(1^\lambda, q)$: Run $\mathcal{G}(1^\lambda)$ to obtain a pairing group setup $\eta = \left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}\right)$. Pick random secret key $\mathsf{sk} = (x, \{y_\tau\}_{\tau=1}^q)$ from $\mathbb{Z}_p^{q+1}$. Set the public key $\mathsf{pk} = (\eta, \alpha, \{\beta_\tau\}_{\tau=1}^q) \leftarrow (\eta, \tilde{g}^x, \{\tilde{g}^{y_\tau}\}_{\tau=1}^q)$.
2. $\mathsf{Sign}(\mathsf{sk}, \{m_\tau\}_{\tau=1}^q)$: Select random r from $\mathbb{Z}_p$, and set $h \leftarrow g^r$. Output the signature $\sigma = (h, s) \leftarrow (h, h^{x + \{y_\tau m_\tau\}_{\tau=1}^q})$
3. $\mathsf{VerifySig}(\mathsf{pk}, \sigma, \{m_\tau\}_{\tau=1}^q)$: Output 1 if $h \neq 1$ and $e(h, \alpha \prod_{\tau=1}^q \beta_\tau^{m_\tau}) = e(s, \tilde{g})$. Else, output 0.

A.5 Threshold Blind Signature

Coconut [36] is an optional declaration credential construction supporting distributed threshold issuance based on Pointcheval-Sanders signature [32]. Unlinkable optional attribute disclosures, and public and private attributes are supported by the framework of [36] even when a part of issuing authorities are malicious or offline. Recently, Rial et al. [33] have analyzed the security properties of Coconut [36] by introducing an ideal functionality which captures all the security properties of a threshold blind signature TBS. They introduced a new construction that follows Coconut with a few modifications to realize TBS ideal functionality. They have some changes for issuing blind signatures and for signature show.

Informally TBS scheme satisfies unforgeability, unlinkability and blindness. Unforgeability guarantees unfeasibility for a corrupted user to convince an honest verifier that it has a valid signature if in fact it has not. Blindness guarantees unfeasibility for a corrupted signer to learn any information about the message m during the execution of IssueSig protocol, except for the fact that m satisfies a predicate. Unlinkability guarantees unfeasibility for a corrupted signer or verifier to learn anything about the message m, except that it satisfies a predicate, or to link the execution of ProveSig with either another execution of ProveSig or with the execution of IssueSig.

We use the improved version of Coconut [36] introduced in [33] with modifications (to their construction such as modelling the communication between the user and the signing maintainer, and embedding the NIZK proofs needed throughout the TBS scheme into proofs generated in our construction as we describe in Sec. 3.2) as a TBS scheme. The scheme TBS = (TBS.KeyGen, IssueSig, TBS.Agg, ProveSig, VerifySig) consists of the following algorithms and protocols (maintainers and signers are interchangeable).

① TBS.KeyGen

1. Run $\left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}\right) \leftarrow \mathcal{G}\left(1^\lambda\right)$ and pick q random generators $\{h_\tau\}_{\tau=1}^q \leftarrow \mathbb{G}$ and set the parameters $par \leftarrow \left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}, \{h_\tau\}_{\tau=1}^q\right)$.
2. Choose $(q+1)$ polynomials $(v, \{w_\tau\}_{\tau=1}^q)$ of degree $(\alpha - 1)$ with random coefficients in $\mathbb{Z}_p$ and set $(x, \{y_\tau\}_{\tau=1}^q) \leftarrow (v(0), \{w_\tau(0)\}_{\tau=1}^q)$.
3. For $j = 1$ to D, set the secret key sk_j of each maintainer M_j as $\mathsf{sk}_j = \left(x_j, \{y_{j,\tau}\}_{\tau=1}^q\right) \leftarrow (v(j), \{w_\tau(j)\}_{\tau=1}^q)$ and set the verification key pk_j of each maintainer M_j as $\mathsf{pk}_j = \left(\tilde{\alpha}_j, \left\{\beta_{j,\tau}, \tilde{\beta}_{j,\tau}\right\}_{\tau=1}^q\right) \leftarrow (\tilde{g}^{x_j}, \{g^{y_{j,\tau}}, \tilde{g}^{y_{j,\tau}}\}_{\tau=1}^q)$.
4. Set $\mathsf{pk} = \left(par, \tilde{\alpha}, \left\{\beta_\tau, \tilde{\beta}_\tau\right\}_{\tau=1}^q\right) \leftarrow (par, \tilde{g}^x, \{g^{y_\tau}, \tilde{g}^{y_\tau}\}_{\tau=1}^q)$.

② IssueSig protocol consists three following algorithms (PrepareBlindSign, BlindSign, Unblind).
2.1. PrepareBlindSign algorithm is run by user U which is as follows:

1. Parse acc as $\{m_\tau\}_{\tau=1}^q$. Pick a random value $o \xleftarrow{\$} \mathbb{Z}_p$ and compute $\mathsf{com} = g^o \prod_{\tau=1}^q h_\tau^{m_\tau}$ and send com to $\mathcal{F}_{\text{RO}}$[11] and receive h from $\mathcal{F}_{\text{RO}}$.
2. Compute commitments to each of the messages. For $\{\tau\}_{\tau=1}^q$, pick random $o_\tau \leftarrow \mathbb{Z}_p$ and set $\mathsf{com}_\tau = g^{o_\tau} h^{m_\tau}$.
3. Compute a NIZK proof π_s for the following relation: $\pi_s = \mathsf{NIZK}\{(\{m_\tau\}_{\tau=1}^q, o, \{o_\tau\}_{\tau=1}^q), \mathsf{com} = g^o \prod_{\tau=1}^q h_\tau^{m_\tau} \wedge \{\mathsf{com}_\tau = g^{o_\tau} h^{m_\tau}\}_{\forall \tau \in [1,q]} \wedge \phi\{m_\tau\}_{\tau=1}^q = 1\}$ and set

$$\mathsf{acc}^{\mathfrak{B}} = (\mathsf{com}, \{\mathsf{com}_\tau\}_{\tau=1}^q, h)$$

2.2. BlindSign algorithm is run by maintainer M_j which is as follows:

1. Send com to $\mathcal{F}_{\text{RO}}$ and receive h' from $\mathcal{F}_{\text{RO}}$. Abort if $h \neq h'$ or π_s is not correct.
2. Compute $c = h^{x_j} \prod_{\tau=1}^q \mathsf{com}_\tau^{y_{j,\tau}}$ and set the blind signature share

$$\sigma_j^{\mathfrak{B}} = (h, h^{x_j} \prod_{\tau=1}^q \mathsf{com}_\tau^{y_{j,\tau}})$$

.

[11] $\mathcal{F}_{\text{RO}}$ denotes functionality of random oracle which is a black box that provides a truly random response from an output domain for every unique request.

2.3. $\mathsf{Unblind}$ algorithm is run by user U which is as follows:

1. Parse $\sigma_j^{\mathfrak{B}}$ as (h', c). Abort if $h \neq h'$.
2. Compute
$$\sigma_j = (h, s_j) \leftarrow \left(h, c \prod_{\tau=1}^{q} \beta_{j,\tau}^{-o_\tau}\right)$$
3. Abort if $e\left(h, \tilde{\alpha}_j \prod_{\tau=1}^{q} \tilde{\beta}_{j,\tau}^{m_\tau}\right) = e\left(s_j, \tilde{g}\right)$ does not hold.

③ $\mathsf{TBS.Agg}$ User U does the following:

1. Let $E \in [1, D]$ be a set of α indices of maintainers in $\mathbb{M}$.
2. For all $j \in E$, evaluate at 0 the Lagrange basis polynomials
$l_j = \left[\prod_{i \in E, i \neq j} (i)/(i-j)\right] \bmod p$.
3. For all $j \in E$, take $\sigma_j = (h, s_j)$ from $\mathbb{M}$ and compute the signature
$$\sigma_{\mathbb{M}} = (h, s) \leftarrow \left(h, \prod_{j \in E} s_j^{l_j}\right)$$
4. Abort if $e\left(h, \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_\tau^{m_\tau}\right) = e(s, \tilde{g})$ does not hold.

④ $\mathsf{ProveSig}$
User U does the following:

1. Parse $\sigma_{\mathbb{M}}$ as (h, s), pick $r \xleftarrow{\$} \mathbb{Z}_p$ and $r' \xleftarrow{\$} \mathbb{Z}_p$.
2. Compute
$$\sigma_{\mathbb{M}}^{\mathsf{int}} = (h', s') \leftarrow \left(h^{r'}, s^{r'} (h')^{r}\right)$$
3. Parse acc as $\{m_\tau\}_{\tau=1}^{q}$. Compute
$$\kappa \leftarrow \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^{r}$$
4. Compute the NIZK proof π_v for the relation: $\pi_v = \mathsf{NIZK}\{(\{m_\tau\}_{\tau=1}^{q}, r) : \kappa = \tilde{\alpha} \prod_{\tau=1}^{q} \tilde{\beta}_\tau^{m_\tau} \tilde{g}^{r} \wedge \varphi\left(\{m_\tau\}_{\tau=1}^{q}\right) = 1\}$
5. Set
$$\sigma_{\mathbb{M}}^{\mathsf{Rnd}} = \left(\sigma_{\mathbb{M}}^{\mathsf{int}}, \kappa\right)$$

⑤ $\mathsf{VerifySig}$
Maintainer M_j does the following:

1. Parse $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ as $\left(\sigma_{\mathbb{M}}^{\mathsf{int}}, \kappa\right)$ and abort if $h' = 1$ or if $e\left(h', \kappa\right) = e\left(s', \tilde{g}\right)$ does not hold.
2. Verify π_v and abort if the proof is not correct.

B Functionalities

The description of the ideal functionalities used in our protocol is as follows. Functionality $\mathcal{F}_{\mathsf{KR}}$ models key registration. A party calls the functionality with $(\mathtt{Register}, \mathsf{sid}, key)$ to register a key for the identifier U of the party. Later, all parties can call the functionality with $(\mathtt{RetrieveKey}, \mathsf{sid}, \mathsf{U})$ to receive the registered key key of party U; or they can call the functionality with $(\mathtt{RetrieveID}, \mathsf{sid}, \mathsf{key})$ to obtain the identifier of the owner of key.

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

Register.
Upon input $(\texttt{Register}, \mathsf{sid}, \mathsf{key})$ from U, output $(\texttt{Register}, \mathsf{sid}, \mathsf{U}, \mathsf{key})$ to $\mathcal{A}$. Upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{A}$, record the pair $(\mathsf{U}, \mathsf{key})$, and output $(\texttt{Registered}, \mathsf{sid})$ to U.
Retrieve.
Upon input $(\texttt{RetrieveKey}, \mathsf{sid}, \mathsf{U})$ from U_j, output $(\texttt{RetrieveKey}, \mathsf{sid}, \mathsf{U}, \mathsf{U}_j)$ to $\mathcal{A}$. Upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{U}, \mathsf{U}_j)$ from $\mathcal{A}$, if there exists a recorded pair $(\mathsf{U}, \mathsf{key})$, output $(\texttt{KeyRetrieved}, \mathsf{sid}, \mathsf{U}, \mathsf{key})$ to U_j. Else, output $(\texttt{KeyRetrieved}, \mathsf{sid}, \mathsf{U}, \perp)$ to U_j.
Upon input $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{key})$ from U_j, output $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{key}, \mathsf{U}_j)$ to $\mathcal{A}$. Upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{key}, \mathsf{U}_j)$ from $\mathcal{A}$, if there exists a recorded pair $(\mathsf{U}, \mathsf{key})$, output $(\texttt{IDRetrieved}, \mathsf{sid}, \mathsf{U}, \mathsf{key})$ to U_j. Else, output $(\texttt{IDRetrieved}, \mathsf{sid}, \mathsf{key}, \perp)$ to U_j.

For privacy-preserving requirement $\mathcal{F}_{\mathsf{CBDC}}$ does not leak identities of users. To realize this functionality, our protocol uses different types of communication channels $\mathcal{F}_{\mathsf{C}}$ to deliver messages and to meet network level anonymity (e.g., preventing traffic analysis attack and extracting identities).

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

Let define a set of parties where S and R denote two parties of the set 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, output $(\texttt{Send}, \mathsf{sid}, \Delta, \mathsf{mid})$ to $\mathcal{A}$.
2. Upon receiving $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, send $(\texttt{Received}, \mathsf{sid}, S, m)$ to R.

Set Δ based on the following parameterized functions:

- for $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ 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 $\mathcal{F}_{\mathsf{C}}^{\mathsf{sra}}$ set $\Delta = (S, |m|)$.
- for $\mathcal{F}_{\mathsf{C}}^{\mathsf{ssa}}$ set $\Delta = (R, |m|)$.
- for $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ set $\Delta = |m|$.
- for $\mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$ 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 $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ set $\Delta = (R, m)$.
  1. 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.
  2. 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.

In 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 interactive protocol Π_{PEReDi} that tolerates static malicious adversaries we use the following Byzantine Agreement functionality $\mathcal{F}_{\mathsf{BA}}$ which can be implemented as follows. Parties send their input to everyone. Once all inputs are received, an honest party (with input 0) switches its input to 1 if at most $2t$ zeros are received. Afterwards, parties engage in a standard Byzantine Agreement protocol [13].

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

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

1. Upon receiving $(\texttt{Agree}, \mathsf{sid}, d_j)$ from M_j where $d_j \in \{0,1\}$, record $(\texttt{Agree}, \mathsf{sid}, d_j, \mathsf{M}_j)$ and send $(\texttt{Agree}, \mathsf{sid}, d_j, \mathsf{M}_j)$ to $\mathcal{A}$. Once $|\{j|d_j = 1\}| \geq t+1$, set $Q = 1$. Once $|\{j|d_j = 0\}| \geq 2t+1$, set $Q = 0$.
2. Upon receiving $(\texttt{Agree.Ok}, \mathsf{sid})$ from $\mathcal{A}$: If $Q \neq \perp$ output $(\texttt{Agreed}, \mathsf{sid}, Q)$ to every M_j via public-delayed output. Else, ignore.

In the following, we define the standard Broadcast functionality $\mathcal{F}_{\mathsf{B}}$ from [26] where it does not guarantee secrecy for the message m.

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

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

Groth et al. [29] formalized ideal functionality of Non-Interactive Zero Knowledge (NIZK) that was introduced by Blum et al. [12]. $\mathcal{F}_{\mathsf{NIZK}}$ does not specify the verifier in advance different from the interactive zero-knowledge proof. The generated proof can be verified by anyone.

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

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

Signature of knowledge (SoK) was first formally defined by Chase et al. [20]. 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 $\mathtt{R}$. 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 $\mathtt{R}$. 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 $(\texttt{Setup}, \mathsf{sid})$ from U if this is the first time that $(\texttt{Setup}, \mathsf{sid})$ is received, send $(\texttt{Setup}, \mathsf{sid})$ to $\mathcal{A}$; upon receiving

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

C Simulation

We describe a simulator $\mathcal{S}$ that reproduces the real-world view of $\mathcal{A}$ and emulate the execution of honest parties. The simulator internally emulates the functionalities $\mathcal{F}_{\mathsf{KR}}, \mathcal{F}_{\mathsf{C}}, \mathcal{F}_{\mathsf{BA}}, \mathcal{F}_{\mathsf{B}}$, $\mathcal{F}_{\mathsf{NIZK}}$ and $\mathcal{F}_{\mathsf{SoK}}$. To do so, it needs to maintain specific lists associated to each functionality. However, without loss of generality, we assume $\mathcal{S}$ internally keeps track of states of functionalities and omit addressing all these lists explicitly. It also maintains the lists $\texttt{List}_{\texttt{UR}}$ for keeping track of registered users and $\texttt{List}_{\texttt{tid}}$ for keeping track of transaction identifiers. $\mathcal{S}$ interacts with the dummy adversary $\mathcal{A}$ and with 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.*

C.1 Simulation of *User Registration*

In all the following cases, $\mathcal{S}$ receives $(\texttt{GenAcc}, \mathsf{sid}, \mathsf{U})$ from $\mathcal{F}_{\mathsf{CBDC}}$ and at the end of the simulation, $\mathcal{S}$ sends the message $(\texttt{Ok.GenAcc}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{F}_{\mathsf{CBDC}}$ if the user receives at least $2t+1$ valid signature shares from maintainers[12]. Throughout the simulation of *User Registration* $\mathcal{S}$ knows the identifier of the user U (regardless of the fact that U is corrupted or not).

Honest U and at most t malicious maintainers: $\mathcal{S}$ initiates *User Registration* protocol by emulating the honest user U.

1. Communication from U to $\mathbb{M}$:
   Emulating $\mathcal{F}_{\mathsf{KR}}$, $\mathcal{S}$ internally records the pair $(\mathsf{U}, \mathsf{pk}_{\mathsf{U}})$ for a randomly chosen value as pk_{U}.
   Simulator on behalf of honest user U is supposed to provide $\mathsf{RI}_j = \big(\mathsf{acc}^{\mathfrak{B}}, a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}, \pi\big)$ to malicious maintainers and associated leakage of channel to adversary $\mathcal{A}$.
   It does so by computing $\mathsf{acc}^{\mathfrak{B}}$ based on dummy values. It also, selects two values as a_j and r_j randomly from $\mathbb{Z}_p^*$ per maintainer and computes $\mathsf{com}_j = g^{a_j} h^{r_j}$. It sets $\mathsf{com}_{\mathbb{M}} \leftarrow \{\mathsf{com}_j\}_{j=1}^{D}$.
   $\mathcal{S}$ also stores $\mathsf{UR} = (a_j, r_j, \mathsf{M}_j, \mathsf{U})$ (per maintainer) in $\texttt{List}_{\texttt{UR}}$.
   $\mathcal{S}$ sets $\texttt{x} \leftarrow \big(\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}\big)$. Emulating $\mathcal{F}_{\mathsf{NIZK}}$ the simulator sends $(\texttt{Prove}, \mathsf{sid}, \texttt{x})$ to the dummy (internally run) adversary $\mathcal{A}$ as the leakage of $\mathcal{F}_{\mathsf{NIZK}}$. Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, the simulator stores $(\texttt{x}, \pi)$.

[12] Doing so makes $\mathcal{F}_{\mathsf{CBDC}}$ to update its internally maintained mappings and output $(\texttt{AccGened}, \mathsf{sid})$ to U and $(\texttt{AccGened}, \mathsf{sid}, \mathsf{U})$ to $\mathbb{M}$.

Emulating $\mathcal{F}_{\mathsf{B}}$ it sends $(\mathtt{Broadcasted}, \mathsf{sid}, \mathsf{U}, \mathsf{com}_{\mathbb{M}})$ to the dummy $\mathcal{A}$ (both as the leakage of $\mathcal{F}_{\mathsf{B}}$ and the message malicious maintainers receive).
Emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$, the simulator leaks $(\mathtt{Send}, \mathsf{sid}, (\mathsf{U}, \mathsf{M}_j, |\mathsf{RI}_j|), \mathsf{mid})$ to $\mathcal{A}$. And upon receiving $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, leaks the next leakage $(\mathtt{Send}, \mathsf{sid}, (\mathsf{U}, \mathsf{M}_{j+1}, |\mathsf{RI}_{j+1}|), \mathsf{mid})$.
Finally, $\mathcal{A}$ (malicious maintainer e.g., M_t) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}, \mathsf{RI}_t)$ from $\mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$.
$\mathcal{S}$ gives that message to $\mathcal{A}$ (who controls M_t) upon receiving $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ from the dummy $\mathcal{A}$ as $\mathcal{S}$ has already simulated all information included in RI_t.
2. Communication among honest and malicious maintainers:
$\mathcal{S}$ outputs $(\mathtt{RetrieveKey}, \mathsf{sid}, \mathsf{U}, \mathsf{M}_j)$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{KR}}$. $\mathcal{S}$ has already emulated the honest user and $\mathcal{F}_{\mathsf{NIZK}}$ by storing $(\mathtt{x}, \pi)$. Hence, upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ via $\mathcal{A}$ (malicious maintainer), the simulator outputs $(\mathtt{Verification}, \mathsf{sid}, 1)$ to $\mathcal{A}$.
$\mathcal{S}$ keeps track of adversary's blockage on the message each honest maintainer receives. As soon as one honest maintainer receives RI the simulator submits $(\mathtt{Ok.GenAcc}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{F}_{\mathsf{CBDC}}$.
3. Communication from $\mathbb{M}$ to U:
Then, $\mathcal{S}$ needs to simulate the view of $\mathcal{A}$ considering the information that is leaked to $\mathcal{A}$ when each maintainer sends back the blind signature share of U's account. To do so, $\mathcal{S}$ sends $(\mathtt{Send}, \mathsf{sid}, (\mathsf{M}_j, \mathsf{U}, \sigma_j^{\mathfrak{B}}), \mathsf{mid})$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ in which $\sigma_w^{\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ and $\sigma_t^{\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$.
Emulating channel functionality allows $\mathcal{S}$ to keep track of active malicious maintainers who participate at generating valid signatures. If at least $2t+1$ valid signature shares are sent by maintainers to the channel functionality the simulator lets functionality output $(\mathtt{AccGened}, \mathsf{sid})$ to U.

Malicious U and at most t malicious maintainers: $\mathcal{A}$ on behalf of malicious U initiates *User Registration* protocol.

1. Communication from U to $\mathbb{M}$:
Once adversary $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{B}}$ with $(\mathtt{Broadcast}, \mathsf{sid}, \mathsf{com}_{\mathbb{M}})$ $\mathcal{S}$ sends $(\mathtt{Broadcasted}, \mathsf{sid}, \mathsf{U}, \mathsf{com}_{\mathbb{M}})$ to $\mathcal{A}$ (both as the leakage of $\mathcal{F}_{\mathsf{B}}$ and the message malicious maintainers receive).
$\mathcal{A}$ calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$ that is emulated by $\mathcal{S}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{RI}_j)$. $\mathcal{S}$ leaks $(\mathtt{Send}, \mathsf{sid}, (\mathsf{U}, \mathsf{M}_j, |\mathsf{RI}_j|), \mathsf{mid})$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$. Upon receiving $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, the simulator sends $(\mathtt{Continue}, \mathsf{sid})$ to $\mathcal{A}$ (malicious U).
$\mathcal{S}$ sends $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}, \mathsf{RI}_t)$ to $\mathcal{A}$ (malicious maintainer M_t) as the output of $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$.
2. Communication among honest and malicious maintainers:
$\mathcal{S}$ who emulates $\mathcal{F}_{\mathsf{KR}}$ checks internally maintained list for $\mathcal{F}_{\mathsf{KR}}$ to see if $(\mathsf{U}, \mathsf{pk}_{\mathsf{U}})$ has already been saved or not. If not, it ignores RI.
$\mathcal{S}$ who emulates $\mathcal{F}_{\mathsf{B}}, \mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$ and honest maintainers waits to receive a message from $\mathcal{F}_{\mathsf{B}}$ and $\mathcal{F}_{\mathsf{C}}^{\mathsf{sc}}$ where the message is sent from $\mathcal{A}$ on behalf of one specific U and $\mathsf{com}_{\mathbb{M}}$ received from both functionalities is the same.
Afterwards, $\mathcal{S}$ checks whether there is a user record UR saved in $\mathtt{List}_{\mathtt{UR}}$ for U. If there is, it ignores RI.
Given the received (a_w, r_w) from $\mathcal{A}$, it computes $g^{a_w}h^{r_w}$ and ignores if it is not equal to $\mathsf{com}_w \in \mathsf{com}_{\mathbb{M}}$ for $\forall w \in \mathcal{H}$.
Else, $\mathcal{S}$ checks if $(\mathtt{x}, \pi)$ such that $\mathtt{x} = \left(\mathsf{acc}^{\mathfrak{B}}, \mathsf{com}_{\mathbb{M}}, \mathsf{pk}_{\mathsf{U}}\right)$ is stored. Otherwise, (to extract the witness) sends $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{NIZK}}$. Upon receiving the answer $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, checks $(\mathtt{x}, \mathtt{w}) \in \mathtt{R}$ and if so, stores $(\mathtt{x}, \pi)$. Else, ignore the message.
$\mathcal{S}$ saves $\mathsf{UR} = (a_j, r_j, \mathsf{com}_{\mathbb{M}}, \mathsf{U})$ in $\mathtt{List}_{\mathtt{UR}}$.
$\mathcal{S}$ keeps track of adversary's blockage on the message each honest maintainer receives. As soon as one honest maintainer receives RI the simulator submits $(\mathtt{Ok.GenAcc}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{F}_{\mathsf{CBDC}}$.

3. Communication from $\mathbb{M}$ to U:
$\mathcal{S}$ sends $(\mathtt{Send}, \mathsf{sid}, (\mathsf{M}_j, \mathsf{U}, \sigma_j^{\mathfrak{B}}), \mathsf{mid})$ to $\mathcal{A}$ as leakage of communication channel in which $\sigma_w^{\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ (as described in the sequences of games Sec. 4.1) and $\sigma_t^{\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$. $\mathcal{S}$ also sends $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathfrak{B}})$ to $\mathcal{A}$ as the message $\mathcal{A}$ (malicious user) receives in the real world once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$.
Emulating channel functionality allows $\mathcal{S}$ to keep track of active malicious maintainers who participate at generating valid signatures ($\mathcal{S}$ has already extracted the witness from malicious user's RI so that it can check the validity of malicious maintainers' signatures). If at least $2t + 1$ valid signature shares are received by the user (note that honest maintainers are emulated by $\mathcal{S}$ itself, and $\mathcal{S}$ generates signatures on behalf of the honest maintainers who have received the RI) the simulator lets functionality output $(\mathtt{AccGened}, \mathsf{sid})$ to U.

C.2 Simulation of *Currency Issuance*

Honest U, honest B and at most t malicious maintainers: $\mathcal{S}$ receives $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$ and initiates *Currency Issuance* protocol by emulating honest B.

1. Communication from B to U:
In the real-world $\mathcal{A}$ sees $(\mathtt{Send}, \mathsf{sid}, (\mathsf{B}, |v|), \mathsf{mid}_{\mathsf{B}})$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sra}}$. The simulator $\mathcal{S}$ has already known $|v|$ and thus sends the leakage to $\mathcal{A}$.
2. Communication from U to B:
Upon receiving $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$ the simulator emulates an honest user U. We 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.
In the real world, $\mathcal{A}$ sees $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, |\rho|, \mathsf{mid}_{\mathsf{U}})$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{ssa}}$. The simulator has already known $|\rho|$ and sends the leakage to $\mathcal{A}$.
3. Communication from B and U to $\mathbb{M}$:
Considering U and B's communications with $\mathbb{M}$, the adversary $\mathcal{A}$ respectively sees $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid}'_{\mathsf{U}})$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}}, \mathsf{mid}'_{\mathsf{B}})$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$. Once $\mathcal{S}$ receives $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{U}})$ from $\mathcal{A}$ it leaks the next leakage $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{j+1}, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid}'_{\mathsf{U}})$ and once it receives $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{B}})$ from $\mathcal{A}$ it leaks the next leakage $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \mathsf{M}_{j+1}, \mathsf{TI}_{\mathsf{B}}, \mathsf{mid}'_{\mathsf{B}})$.
Hence, $\mathcal{S}$ is supposed to simulate the view of dummy $\mathcal{A}$ with respect to the information real-world $\mathcal{A}$ sees without knowing the identity of U.
First of all, based on $\mathsf{PrepareBlindSign}$, $\mathcal{S}$ selects random values to compute $\mathsf{acc}^{\mathsf{new},\mathfrak{B}}$.
Then, computes $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ in a way described in Sec. 4.1. Afterwards, $\mathcal{S}$ computes a threshold encryption ψ on dummy values as plaintexts.
It computes T by randomly selecting $z \overset{\$}{\leftarrow} \mathbb{Z}_p$ and let $\mathsf{T} \leftarrow g^z$.
$\mathcal{S}$ sets $\mathtt{x} \leftarrow \big(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}\big)$. Emulating $\mathcal{F}_{\mathsf{NIZK}}$, the simulator sends $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$. The simulator receives $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$ and records $(\mathtt{x}, \pi)$.
$\mathcal{S}$ sends $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid}'_{\mathsf{U}})$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}}, \mathsf{mid}'_{\mathsf{B}})$ to $\mathcal{A}$ such that $\mathsf{TI}_{\mathsf{U}} = \big(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi\big)$ and $\mathsf{TI}_{\mathsf{B}} = \psi$ as explained above.
Finally, $\mathcal{A}$ (malicious maintainer) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid}'_{\mathsf{U}})$ and $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, \mathsf{TI}_{\mathsf{B}})$ from channel (emulated by $\mathcal{S}$) once dummy $\mathcal{A}$ sends $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{U}})$ and $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{B}})$ to $\mathcal{S}$ respectively.
4. Communication among honest and malicious maintainers:
$\mathcal{S}$ has already emulated the honest user and $\mathcal{F}_{\mathsf{NIZK}}$ (it has stored $(\mathtt{x}, \pi)$).
Hence, upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ via $\mathcal{A}$ (malicious maintainer), the simulator outputs $(\mathtt{Verification}, \mathsf{sid}, 1)$ to $\mathcal{A}$.

As soon as one honest maintainer receives both TI_U and TI_B, the simulator submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_\mathsf{id})$ to $\mathcal{F}_\mathsf{CBDC}$ where $t_\mathsf{id} = (\psi, \mathsf{T})$. The values of ψ and T are simulated by $\mathcal{S}$ as described above.

For each maintainer (either honest or malicious) who generates a valid signature on user's account $\mathcal{S}$ submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ to $\mathcal{F}_\mathsf{CBDC}$ where M_k is the identifier of that maintainer.

5. Communication from $\mathbb{M}$ to U:
$\mathcal{S}$ sends $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}'_\mathsf{U})$ to $\mathcal{A}$ as leakage of $\mathcal{F}_\mathsf{C}^\mathsf{sa}$ in which $\sigma_w^{\mathsf{new},\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ and $\sigma_t^{\mathsf{new},\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$ (malicious maintainer).

Malicious U, honest B and at most *t* malicious maintainers: In this case, $\mathcal{S}$ receives $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}, v)$ from $\mathcal{F}_\mathsf{CBDC}$. The simulator initiates *Currency Issuance* protocol on behalf of honest B to issue a digital currency worth of v for U.

1. Communication from B to U:
Similar to the case of honest U and honest B, $\mathcal{S}$ leaks to $\mathcal{A}$ the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, |v|, \mathsf{mid}_\mathsf{B})$ as leakage of $\mathcal{F}_\mathsf{C}^\mathsf{sra}$. In the real world, $\mathcal{A}$ (malicious U) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, v)$, $\mathcal{S}$ has already received $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}, v)$ from $\mathcal{F}_\mathsf{CBDC}$ thus it knows v and it sends $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, v)$ to the dummy (internally run) $\mathcal{A}$ once it receives the message $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$.
2. Communication from U to B:
Emulating $\mathcal{F}_\mathsf{C}^\mathsf{ssa}$, the simulator receives $\mathcal{A}$'s message of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \rho)$. Then, $\mathcal{S}$ leaks $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, |\rho|, \mathsf{mid}_\mathsf{U})$ to $\mathcal{A}$ as leakage of $\mathcal{F}_\mathsf{C}^\mathsf{ssa}$. Once $\mathcal{S}$ receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$ it proceeds emulating honest B.
3. Communication from B and U to $\mathbb{M}$:
Simulating the communication from B to $\mathbb{M}$ is similar to the case of honest U and honest B.
Regarding the communication from U to $\mathbb{M}$, the adversary calls $\mathcal{F}_\mathsf{C}^\mathsf{sa}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_\mathsf{U})$.
The simulator sends $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_\mathsf{U}, \mathsf{mid}'_\mathsf{U})$ to $\mathcal{A}$ as the leakage of $\mathcal{F}_\mathsf{C}^\mathsf{sa}$. Upon receiving $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}'_\mathsf{U})$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\mathtt{Continue}, \mathsf{sid})$ to $\mathcal{A}$ (malicious U).
$\mathcal{A}$ (malicious maintainer) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_\mathsf{U}, \mathsf{mid}'_\mathsf{U})$ from $\mathcal{S}$. For the simulator to do so, it uses $\mathcal{A}$'s sent information TI_U once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}'_\mathsf{U})$ from $\mathcal{A}$.
$\mathcal{S}$ submits $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, v)$ to $\mathcal{F}_\mathsf{CBDC}$ on behalf of malicious U.
4. Communication among honest and malicious maintainers:
$\mathcal{S}$ checks if ψ included in TI_U is generated using the randomness it received from $\mathcal{A}$ and value v. In other words, whether ψ equals to the threshold encryption that is generated by $\mathcal{S}$[13].
Also $\mathcal{S}$ checks if ψ is the first element of one of the t_id arrays saved in $\mathtt{List}_\mathtt{tid}$. If it is, then ignores.
Then, verifies whether T is the second element of one of the saved t_id arrays in $\mathtt{List}_\mathtt{tid}$. If not, checks whether $(\mathtt{x}, \pi)$ such that $\mathtt{x} = \left(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_\mathbb{M}^\mathsf{Rnd}, \mathsf{T}\right)$ is stored. Otherwise, sends $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$ as leakage of $\mathcal{F}_\mathsf{NIZK}$. Upon receiving the answer $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, checks $(\mathtt{x}, \mathtt{w}) \in \mathtt{R}$. If so stores $(\mathtt{x}, \pi)$. Else, ignore the message.
$\mathcal{S}$ saves $t_\mathsf{id} = (\psi, \mathsf{T})$ in $\mathtt{List}_\mathtt{tid}$.
$\mathcal{S}$ submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_\mathsf{id})$ to $\mathcal{F}_\mathsf{CBDC}$ where $t_\mathsf{id} = (\psi, \mathsf{T})$. The values of ψ is calculated by $\mathcal{S}$ (also it is given by $\mathcal{A}$) and T is given by $\mathcal{A}$ as described above.
As soon as one honest maintainer receives both TI_U and TI_B, the simulator submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_\mathsf{id})$ to $\mathcal{F}_\mathsf{CBDC}$ where $t_\mathsf{id} = (\psi, \mathsf{T})$. The values of ψ and T are received from $\mathcal{A}$ as described above.

[13] In this case, $\mathcal{S}$ has already received $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}, v)$ from $\mathcal{F}_\mathsf{CBDC}$ which means regulatory compliance and so on have already been verified by $\mathcal{F}_\mathsf{CBDC}$.

For each maintainer (either honest or malicious) who generates a valid signature on user's account $\mathcal{S}$ submits $(\texttt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ to $\mathcal{F}_{\mathsf{CBDC}}$ where M_k is the identifier of that maintainer.

5. Communication from $\mathbb{M}$ to U:
   For simulating the messages that are sent back to malicious U, $\mathcal{S}$ gives $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}'_{\mathsf{U}})$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ in which $\sigma_w^{\mathsf{new},\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ (as described in the sequences of games Sec. 4.1) and $\sigma_t^{\mathsf{new},\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$ (malicious maintainer).
   Also, once it receives $(\texttt{Ok.End}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{U}})$ from $\mathcal{A}$, it outputs $(\texttt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{new},\mathfrak{B}})$ to $\mathcal{A}$ as the message that malicious U receives.

Honest U, malicious B and at most t malicious maintainers: $\mathcal{A}$ on behalf of malicious B, initiates *Currency Issuance* protocol.

1. Communication from B to U:
   $\mathcal{A}$ initiates the protocol on behalf of B by calling $\mathcal{F}_{\mathsf{C}}^{\mathsf{sra}}$ with input $(\texttt{Send}, \mathsf{sid}, \mathsf{U}^*, v^*)$. Hence, emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{sra}}$ the simulator knows U^* and v^* and sends $(\texttt{Send}, \mathsf{sid}, \mathsf{B}, |v^*|, \mathsf{mid}_{\mathsf{B}})$ to $\mathcal{A}$ as the leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sra}}$. The simulator submits a currency issuance transaction to $\mathcal{F}_{\mathsf{CBDC}}$ with input $(\texttt{Iss}, \mathsf{sid}, \mathsf{U}^*, v^*)$ on behalf of malicious B. If $\mathcal{S}$ receives $(\texttt{AcceptIss}, \mathsf{sid}, \mathsf{pid})$, it concludes that sent values by $\mathcal{A}$, namely U^* and v^* are the same as corresponding values in honest U's message given to $\mathcal{F}_{\mathsf{CBDC}}$. In other words, $\mathsf{U}^* = \mathsf{U}$ and $v^* = v$ hold. Hence, it continues the protocol otherwise it ignores [14]
2. Communication from U to B:
   $\mathcal{S}$ emulates U and this emulation is similar to the case of honest U and honest B. In addition, emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{ssa}}$ the simulator sends $(\texttt{Received}, \mathsf{sid}, \mathsf{U}, \rho)$ to $\mathcal{A}$ in which ρ is chosen randomly by $\mathcal{S}$.
3. Communication from B and U to $\mathbb{M}$:
   The simulation of communication between U and $\mathbb{M}$ is similar to the case of honest U and honest B.
   $\mathcal{A}$ (malicious B) calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with input $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}})$. Emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$, the simulator leaks $(\texttt{Send}, \mathsf{sid}, \mathsf{B}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}}, \mathsf{mid}'_{\mathsf{B}})$ for to $\mathcal{A}$. The adversary (malicious maintainer) receives $(\texttt{Received}, \mathsf{sid}, \mathsf{B}, \mathsf{TI}_{\mathsf{B}})$ from $\mathcal{S}$ if $\mathcal{S}$ receives $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{B}})$ from $\mathcal{A}$.
   Upon receiving $(\texttt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{B}})$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\texttt{Continue}, \mathsf{sid})$ to $\mathcal{A}$ and leaks the next massage similarly.
4. Communication among honest and malicious maintainers:
   $\mathcal{S}$ (on behalf of honest maintainer) checks if $\mathsf{TI}_{\mathsf{B}} = \psi$ holds or not such that ψ is computed using the randomness chosen by itself and the value v. If it does not hold, $\mathcal{S}$ ignores. Other parts of simulation are similar to the case of honest U and honest B.
5. Communication from $\mathbb{M}$ to U:
   This simulation is similar to (the associated simulation of) the case of honest U and honest B.

Malicious U, malicious B and at most t malicious maintainers: In this case, exchanging information between U and B namely communication from B to U and communication from U to B is done by $\mathcal{A}$. If $\mathcal{A}$ uses communication channel functionalities to exchange information between U and B, the simulator leaks whatever real-world $\mathcal{A}$ sees as the leakage of channels to the dummy $\mathcal{A}$ similar to the associated simulations in the cases of malicious U and honest B, and honest U and malicious B described above.

[14] Doing so, $\mathcal{S}$ captures the fact that if B tries to issue a currency that breaks regulatory rules imposed to U the transaction will be failed. In the real world, U will not engage in a *Currency Issuance* protocol when it knows that doing so will break the rules (when U is malicious, after it engages in a transaction that breaks the regulatory rules, the transaction will be failed by maintainers as we will see in the next case).

1. Communication from B and U to $\mathbb{M}$:
   $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{U}})$ on behalf of U. Also, $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}})$ on behalf of B. The simulator leaks $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid}'_{\mathsf{U}})$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{B}, \mathsf{M}_j, \mathsf{TI}_{\mathsf{B}}, \mathsf{mid}'_{\mathsf{B}})$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ and $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ using the information received from $\mathcal{A}$.
   Once $\mathcal{S}$ receives $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{U}})$ from $\mathcal{A}$ it sends $(\mathtt{Continue}, \mathsf{sid})$ to $\mathcal{A}$. Similarly once $\mathcal{S}$ receives $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{B}})$ from $\mathcal{A}$ it sends $(\mathtt{Continue}, \mathsf{sid})$ to $\mathcal{A}$.
   The adversary (malicious maintainer) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_{\mathsf{U}}, \mathsf{mid}'_{\mathsf{U}})$ and $(\mathtt{Received}, \mathsf{sid}, \mathsf{B}, \mathsf{TI}_{\mathsf{B}})$ from $\mathcal{S}$. For the simulator to do so, it uses $\mathcal{A}$'s sent information TI_{U} and TI_{B} once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{U}})$ and $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}'_{\mathsf{B}})$ respectively.
2. Communication among honest and malicious maintainers:
   $\mathcal{S}$ checks if ψ in TI_{B} equals to ψ in TI_{U}. If it equals, $\mathcal{S}$ checks if ψ is the first element of one of the t_{id} arrays saved in $\mathtt{List_{tid}}$. If it is, then ignores. Else, $\mathcal{S}$ checks whether $(\mathtt{x}, \pi)$ such that $\mathtt{x} = \left(\psi, \mathsf{acc}^{\mathsf{new},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}\right)$ is stored. Otherwise sends $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{NIZK}}$. Upon receiving the answer $(\mathtt{Witness}, \mathsf{sid}, \mathtt{w})$ from $\mathcal{A}$, checks $(\mathtt{x}, \mathtt{w}) \in \mathtt{R}$. If so stores $(\mathtt{x}, \pi)$.
   Having the witness $\mathtt{w}$, the simulator submits a currency issuance transaction to $\mathcal{F}_{\mathsf{CBDC}}$ with input $(\mathtt{Iss}, \mathsf{sid}, \mathsf{U}, v)$ on behalf of malicious B. If it receives $(\mathtt{Iss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}, v)$ from $\mathcal{F}_{\mathsf{CBDC}}$ it saves $t_{\mathsf{id}} = (\psi, \mathsf{T})$ in $\mathtt{List_{tid}}$ else ignores.
   Other parts of simulation are similar to the case of malicious U and honest B.
3. Communication from $\mathbb{M}$ to U:
   This simulation is similar to the case of malicious U and honest B.

C.3 Simulation of *Payment*

Honest U_s, honest U_r and at most t malicious maintainers: $\mathcal{S}$ receives $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$ and initiates *Payment* protocol by emulating an honest sender U_s. We 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.

1. Communication from U_s to U_r:
   In the real-world $\mathcal{A}$ sees $(\mathtt{Send}, \mathsf{sid}, |(\rho_s, v)|, \mathsf{mid}_s)$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$. The simulator has already known $|(\rho_s, v)|$ and sends $(\mathtt{Send}, \mathsf{sid}, |(\rho_s, v)|, \mathsf{mid}_s)$ to $\mathcal{A}$ once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}_s)$ from $\mathcal{A}$.
2. Communication from U_r to U_s:
   Upon receiving $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$ (similar to what was explained for traced U_s above, if the receiver U_r has already been traced, $\mathcal{S}$ receives $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_r)$ from $\mathcal{F}_{\mathsf{CBDC}}$), the simulator emulates honest U_r.
   In the real world, $\mathcal{A}$ sees $(\mathtt{Send}, \mathsf{sid}, |\rho_r|, \mathsf{mid}_r)$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$. The simulator has already known $|\rho_r|$ and sends $(\mathtt{Send}, \mathsf{sid}, |\rho_r|, \mathsf{mid}_r)$ to $\mathcal{A}$ once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}_r)$ from $\mathcal{A}$. .
3. Communication from U_s and U_r to $\mathbb{M}$:
   Regarding U_s and U_r's communications with $\mathbb{M}$, the adversary respectively sees $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_s, \mathsf{mid}'_s)$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_r, \mathsf{mid}'_r)$ as leakages of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$.
   Hence, $\mathcal{S}$ is supposed to simulate the view of the dummy $\mathcal{A}$ with respect to the information real-world $\mathcal{A}$ sees. We note the simulation of U_s's communications with $\mathbb{M}$ and the simulation of U_r's communications with $\mathbb{M}$ in this step of the protocol is the same. Hence, in the following we describe simulation for U_s.
   Using $\mathsf{PrepareBlindSign}$ algorithm, $\mathcal{S}$ selects random values to compute $\mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}$. Then, computes $\sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}$ in a way described in the Sec. 4.1.
   Then, $\mathcal{S}$ computes a threshold encryption ψ on a dummy value as plaintext.
   It computes T by randomly selecting $z_s \overset{\$}{\leftarrow} \mathbb{Z}_p$ and let $\mathsf{T_s} \leftarrow g^{z_s}$.
   $\mathcal{S}$ sets $\mathtt{x}_s \leftarrow \left(\psi_s, \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s\right)$.

Emulating $\mathcal{F}_{\mathsf{SoK}}$, the simulator computes $\overline{\sigma_s}(\psi_r) \leftarrow \mathsf{Simsign}(\psi_r, \mathtt{x}_s)$. The simulator records the entry $(\psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$.
Therefore, U_s's transaction information TI_s is simulated by $\mathcal{S}$ which is of the form $\mathsf{TI}_s = \big(\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s\big)$.
The simulator gives $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_s, \mathsf{mid}'_s)$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$.
Finally, $\mathcal{A}$ (malicious maintainer) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_s, \mathsf{mid}'_s)$ from $\mathcal{S}$ once $\mathcal{S}$ receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}'_s)$ from $\mathcal{A}$. To do so, $\mathcal{S}$ uses the above simulated values for TI_s.

4. Communication among honest and malicious maintainers:
$\mathcal{S}$ has already emulated the honest sender U_s and honest receiver U_r, and $\mathcal{F}_{\mathsf{SoK}}$ (it has stored $(\psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$ and $(\psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s))$).
Hence, once $\mathcal{A}$ (malicious maintainer) calls $\mathcal{F}_{\mathsf{SoK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$ and $(\mathtt{Verify}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s))$, the simulator outputs $(\mathtt{Verified}, \mathsf{sid}, \psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r), 1)$ and $(\mathtt{Verified}, \mathsf{sid}, \psi_s, \mathtt{x}_r, \overline{\sigma_r}(\psi_s), 1)$ respectively to $\mathcal{A}$.
As soon as one honest maintainer receives both TI_s and TI_r, the simulator submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ to $\mathcal{F}_{\mathsf{CBDC}}$ where $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ (note that pid is unique per transaction hence $\mathcal{F}_{\mathsf{CBDC}}$ can distinguishes payment and issuance transactions based on tables it has generated with respect to pid). The values of $\psi_s, \psi_r, \mathsf{T}_s$ and T_r are simulated by $\mathcal{S}$ as described above.
For each maintainer (either honest or malicious) who generates a valid signature on sender and receiver's account $\mathcal{S}$ submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ to $\mathcal{F}_{\mathsf{CBDC}}$ where M_k is the identifier of that maintainer.
5. Communication from $\mathbb{M}$ to U_s and U_r:
$\mathcal{S}$ sends $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}'_s)$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_{r,j}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}'_r)$ to $\mathcal{A}$ as leakages of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ in which $\sigma_w^{\mathsf{new},\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ and $\sigma_t^{\mathsf{new},\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$ (malicious maintainer).

Malicious U_s, honest U_r and at most t malicious maintainers: $\mathcal{S}$ receives $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid})$ from $\mathcal{F}_{\mathsf{CBDC}}$ (similarly, if the sender U_s has already been traced, $\mathcal{S}$ receives $(\mathtt{AcceptIss}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s)$ from $\mathcal{F}_{\mathsf{CBDC}}$). $\mathcal{A}$ on behalf of malicious U_s, initiates *Payment* protocol.

1. Communication from U_s to U_r:
$\mathcal{A}$ initiates the protocol on behalf of U_s by calling $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}_r^*, (\rho_s, v^*))$.
Hence, emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ the simulator knows U_r^* and v^* and sends $(\mathtt{Send}, \mathsf{sid}, |\,(\rho_s, v^*)\,|, \mathsf{mid}_s)$ to $\mathcal{A}$ as the leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$.
The simulator submits a payment transaction to $\mathcal{F}_{\mathsf{CBDC}}$ with input $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r^*, v^*)$ on behalf of malicious U_s.
If $\mathcal{S}$ receives $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid})$ (or $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_r)$) from $\mathcal{F}_{\mathsf{CBDC}}$, it concludes that sent values by $\mathcal{A}$, namely U_r^* and v^* are the same as corresponding values in honest U_r's message which is $(\mathtt{GenTnxRcv}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s, v)$. In other words, $\mathsf{U}_r^* = \mathsf{U}_r$ and $v^* = v$ hold.
Hence, it continues the protocol otherwise it ignores[15].
2. Communication from U_r to U_s:
$\mathcal{S}$ emulates U_r and the simulation process is similar to the case of honest U_s and honest U_r.
In addition, emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ the simulator sends $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}_r, \rho_r)$ to $\mathcal{A}$ in which ρ_r is chosen randomly by $\mathcal{S}$ once $\mathcal{S}$ receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}_r)$ from $\mathcal{A}$.
3. Communication from U_s and U_r to $\mathbb{M}$:
The simulation of communications between U_r and $\mathbb{M}$ is similar to the case of honest U_s and honest U_r.

[15] Doing so, $\mathcal{S}$ captures the fact that if malicious U_s tries to make a payment that breaks regulatory rules related to the account of honest U_r the transaction will be failed. Because, in the real world, U_r will not engage in a *Payment* protocol when it knows that doing so will not be in compliant with system's rules.

$\mathcal{A}$ (malicious U_s) calls $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_s)$.
Emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$, the simulator leaks $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{TI}_s, \mathsf{mid}_s')$ to $\mathcal{A}$.
Upon receiving $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}_s')$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\mathtt{Continue}, \mathsf{sid})$ to $\mathcal{A}$ (malicious U_s).
Finally, $\mathcal{A}$ (malicious maintainer) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{TI}_s, \mathsf{mid}_s')$ from $\mathcal{S}$. To do so, $\mathcal{S}$ uses $\mathcal{A}$'s sent information TI_s once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}_s')$ from $\mathcal{A}$.

4. Communication among honest and malicious maintainers:
   Having $\mathsf{TI}_s = \big(\psi_s, \psi_r, \overline{\sigma_s}(\psi_r), \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s\big)$ generated by $\mathcal{A}$, the simulator checks if ψ_s and ψ_r equal the values that $\mathcal{S}$ has generated internally or not (using v and the random values exchanged between $\mathcal{A}$ and $\mathcal{S}$). If they are not the same, $\mathcal{S}$ ignores.
   Then, verifies whether T_s exists in one of the saved t_{id} arrays in $\mathtt{List_{tid}}$.
   If not checks whether $(\psi_r, \mathtt{x}_s, \sigma')$ such that $\mathtt{x}_s = \big(\psi_s, \mathsf{acc}_s^{\mathsf{new},\mathfrak{B}}, \sigma_{s,\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}_s\big)$ is stored for some σ' or not. Else emulating $\mathcal{F}_{\mathsf{SoK}}$, lets $\mathtt{w}_s \leftarrow \mathsf{Extract}(\psi_r, \mathtt{x}_s, \overline{\sigma_s}(\psi_r))$. Then, if $(\mathtt{x}_s, \mathtt{w}_s) \in \mathtt{R}$ proceeds as follows else ignores.
   $\mathcal{S}$ saves $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ in $\mathtt{List_{tid}}$.
   As soon as one honest maintainer receives both TI_s and TI_r, the simulator submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, t_{\mathsf{id}})$ to $\mathcal{F}_{\mathsf{CBDC}}$ where $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$. The values of ψ_s, ψ_r and T_r are simulated by $\mathcal{S}$ (the first two also are given by $\mathcal{A}$ to $\mathcal{S}$) and T_s is sent by $\mathcal{A}$ to $\mathcal{S}$.
   For each maintainer (either honest or malicious) who generates a valid signature on sender and receiver's account $\mathcal{S}$ submits $(\mathtt{GenTnx}, \mathsf{sid}, \mathsf{pid}, \mathsf{M}_k)$ to $\mathcal{F}_{\mathsf{CBDC}}$ where M_k is the identifier of that maintainer.
5. Communication from $\mathbb{M}$ to U_s and U_r:
   $\mathcal{S}$ sends $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}_s')$ and $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_{r,j}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}_r')$ to $\mathcal{A}$ as leakages of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ in which $\sigma_w^{\mathsf{new},\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ (as described in the sequences of games Sec. 4.1) and $\sigma_t^{\mathsf{new},\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$ (malicious maintainer).
   $\mathcal{S}$ also sends $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, \sigma_{s,j}^{\mathsf{new},\mathfrak{B}})$ to $\mathcal{A}$ (malicious U_s) once it receives $(\mathtt{Ok.End}, \mathsf{sid}, \mathsf{mid}_s')$ from $\mathcal{A}$.

Honest U_s, malicious U_r and at most t malicious maintainers: $\mathcal{S}$ on behalf of honest U_s, initiates *Payment* protocol. In this case, $\mathcal{S}$ receives $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s, \mathsf{U}_r, v)$ from $\mathcal{F}_{\mathsf{CBDC}}$.

1. Communication from U_s to U_r:
   $\mathcal{S}$ initiates the protocol on behalf of U_s. The simulator emulates $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ and sends $(\mathtt{Send}, \mathsf{sid}, |(\rho_s, v)|, \mathsf{mid}_s)$ to $\mathcal{A}$ as the leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$ such that ρ_s is chosen randomly by $\mathcal{S}$. The real world adversary also receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{U}_s, \rho_s, v)$ and $\mathcal{S}$ sends this message to $\mathcal{A}$ using the leaked information from $\mathcal{F}_{\mathsf{CBDC}}$ once it receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}_s)$ from $\mathcal{A}$.
2. Communication from U_r to U_s:
   Emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$, the simulator receives $\mathcal{A}$'s message of the form $(\mathtt{Send}, \mathsf{sid}, \mathsf{U}_s, \rho_r)$. Then, $\mathcal{S}$ leaks $(\mathtt{Send}, \mathsf{sid}, |\rho_r|, \mathsf{mid}_r)$ to $\mathcal{A}$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{fa}}$. Once $\mathcal{S}$ receives $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}_r)$ from $\mathcal{A}$ continues. Else, ignores.
3. Communication from U_s and U_r to $\mathbb{M}$:
   The simulation of communications between U_s and $\mathbb{M}$ is similar to the case of honest U_s and honest U_r.
   The simulation of communications between U_r and $\mathbb{M}$ is similar to the case of malicious U_s and honest U_r, however, for malicious U_r rather than malicious U_s.
4. Communication among honest and malicious maintainers:
   The simulation of this part is similar to the case of malicious U_s and honest U_r, however, for malicious U_r rather than malicious U_s.
5. Communication from $\mathbb{M}$ to U_s and U_r:
   The simulation of this part is similar to the case of malicious U_s and honest U_r, however, for malicious U_r rather than malicious U_s.

Malicious U_s, malicious U_r, and at most t malicious maintainers: In this case, exchanging information between U_s and U_r namely communication from U_s to U_r and communication from U_r to U_s is done by $\mathcal{A}$. If $\mathcal{A}$ uses communication channel functionalities to exchange information between U_s and U_r, the simulator leaks whatever real-world $\mathcal{A}$ sees as the leakage of channels to the dummy $\mathcal{A}$ similar to the associated simulations in the cases of malicious U_s and honest U_r, and honest U_s and malicious U_r described above.

1. Communication from U_s and U_r to $\mathbb{M}$:
   Communications from U_s to $\mathbb{M}$ is similar to the associated communications in case of malicious U_s and honest U_r, and communications from U_r to $\mathbb{M}$ is similar to the associated communications in the case of honest U_s and malicious U_r.
2. Communication among honest and malicious maintainers:
   $\mathcal{S}$ checks if (ψ_s, ψ_r) in TI_s equals to (ψ_s, ψ_r) in TI_r. If it is not, $\mathcal{S}$ ignores.
   After extracting the witnesses $\mathtt{w}_s$ and $\mathtt{w}_r$ (similar to what was described before), the simulator submits a payment $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{U}_r, v)$ to $\mathcal{F}_{\mathsf{CBDC}}$ on behalf of U_s.
   The rest of the simulation of this step of protocol is similar to the associated simulations in the cases of malicious U_s and honest U_r, and honest U_s and malicious U_r (e.g., upon receiving $(\mathtt{GenTnxSnd}, \mathsf{sid}, \mathsf{pid}, \mathsf{U}_s, \mathsf{U}_r, v)$ from $\mathcal{F}_{\mathsf{CBDC}}$ the simulator starts emulating $\mathbb{M}$ and so on).
3. Communication from $\mathbb{M}$ to U_s and U_r:
   This simulation is similar to the associated simulations in the cases of malicious U_s and honest U_r, and honest U_s and malicious U_r

C.4 Simulation of *Abort Transaction*

Honest U, and at most t malicious maintainers: In this case, $\mathcal{S}$ receives $(\mathtt{AbrTnx}, \mathsf{sid}, t_{\mathsf{id}})$ from $\mathcal{F}_{\mathsf{CBDC}}$.

1. Communication from U to $\mathbb{M}$:
   Considering U's communications with $\mathbb{M}$, the adversary $\mathcal{A}$ sees $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{AR}, \mathsf{mid})$ as leakage of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$. Once $\mathcal{S}$ receives $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$ it leaks the next leakage $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_{j+1}, \mathsf{AR}, \mathsf{mid}')$.
   Hence, $\mathcal{S}$ is supposed to simulate the view of dummy $\mathcal{A}$ with respect to the information real-world $\mathcal{A}$ sees without knowing the identity of U, however, by having the leaked T included in t_{id} given by $\mathcal{F}_{\mathsf{CBDC}}$.
   First of all, based on $\mathsf{PrepareBlindSign}$, $\mathcal{S}$ selects random values to compute $\mathsf{acc}^{\mathsf{r},\mathfrak{B}}$.
   Then, computes $\sigma_{\mathbb{M}}^{\mathsf{Rnd}}$ in a way described in Sec. 4.1.
   Using the leaked T included in t_{id}, $\mathcal{S}$ sets $\mathtt{x} \leftarrow \left(\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}\right)$.
   Emulating $\mathcal{F}_{\mathsf{NIZK}}$, the simulator sends $(\mathtt{Prove}, \mathsf{sid}, \mathtt{x})$ to $\mathcal{A}$.
   The simulator receives $(\mathtt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$ and records $(\mathtt{x}, \pi)$.
   $\mathcal{S}$ sends $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_j, \mathsf{AR}, \mathsf{mid})$ to $\mathcal{A}$ such that $\mathsf{AR} = \left(\mathsf{acc}^{\mathsf{r},\mathfrak{B}}, \sigma_{\mathbb{M}}^{\mathsf{Rnd}}, \mathsf{T}, \pi\right)$.
   Finally, $\mathcal{A}$ (malicious maintainer) receives $(\mathtt{Received}, \mathsf{sid}, \mathsf{AR}, \mathsf{mid})$ from the channel (emulated by $\mathcal{S}$) once dummy $\mathcal{A}$ sends $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ to $\mathcal{S}$.
2. Communication among honest and malicious maintainers:
   $\mathcal{S}$ has already emulated the honest user and $\mathcal{F}_{\mathsf{NIZK}}$ by storing $(\mathtt{x}, \pi)$. Hence, upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, \mathtt{x}, \pi)$ via $\mathcal{A}$ (malicious maintainer), the simulator outputs $(\mathtt{Verification}, \mathsf{sid}, 1)$ to $\mathcal{A}$.
   Emulating Byzantine Agreement functionality $\mathcal{F}_{\mathsf{BA}}$, $\mathcal{S}$ should leak $(\mathtt{Agree}, \mathsf{sid}, d_j, \mathsf{M}_j)$ to $\mathcal{A}$ where for malicious maintainers $\mathcal{S}$ uses d_t received by $\mathcal{A}$ (malicious maintainer M_t). On behalf of honest maintainers $\mathcal{S}$ checks its table to see whether it sees a t_{id} (for issuance or payment transaction) that contains T in $\mathtt{List}_{\mathtt{tid}}$ or not. In case it sees it sets the value of d_w to 1 (on behalf of honest maintainer M_w).
   $\mathcal{S}$ keeps track of adversary's blockage on the message each honest maintainer receives. As soon as one honest maintainer receives AR the simulator submits $(\mathtt{AbrTnx.Ok}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{F}_{\mathsf{CBDC}}$.
   The simulator simulates the rest of the protocol based on the output of $\mathcal{F}_{\mathsf{BA}}$ as follows:

3. If $Q = 1$:
   In this case, there exists at least one honest maintainer (e.g., M_w) who is emulated by $\mathcal{S}$.
   $\mathcal{S}$ leaks $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_i, \mathsf{M}_j, (\mathsf{TI}_s, \mathsf{TI}_r), \mathsf{mid})$ to $\mathcal{A}$.
   Also, it sends $(\texttt{Received}, \mathsf{sid}, \mathsf{M}_i, (\mathsf{TI}_s, \mathsf{TI}_r))$ to $\mathcal{A}$ (malicious maintainer) once it receives $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$.
   Emulating $\mathcal{F}_{\mathsf{SoK}}$, $\mathcal{S}$ outputs $(\texttt{Verified}, \mathsf{sid}, \psi, \mathtt{x}, \sigma(\psi), \mathsf{Verify}(\psi, \mathtt{x}, \sigma(\psi)))$ to $\mathcal{A}$ once $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{SoK}}$ with $(\texttt{Verify}, \mathsf{sid}, \psi, \mathtt{x}, \sigma(\psi))$.
   $\mathcal{S}$ saves $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$ in $\texttt{List}_{\texttt{tid}}$.
   (a) Communication from $\mathbb{M}$ to U:
   $\mathcal{S}$ sends $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_k, \sigma_{s,k}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}'_s)$ and $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_k, \sigma_{r,k}^{\mathsf{new},\mathfrak{B}}, \mathsf{mid}'_r)$ to $\mathcal{A}$ as leakages of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ in which mid'_s and mid'_r are included in TI_s and TI_r. $\sigma_k^{\mathsf{new},\mathfrak{B}}$ (for both U_s and U_r) parses as $\sigma_w^{\mathsf{new},\mathfrak{B}}$ and $\sigma_t^{\mathsf{new},\mathfrak{B}}$ (for both U_s and U_r). Also, $\sigma_w^{\mathsf{new},\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ and $\sigma_t^{\mathsf{new},\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$ (malicious maintainer).
4. If $Q = 0$:
   Emulating honest maintainers who have had $t_{\mathsf{id}} = (\psi_s, \psi_r, \mathsf{T}_s, \mathsf{T}_r)$, the simulator deletes the saved t_{id} entry.
   $\mathcal{S}$ saves $t_{\mathsf{id}} = (\mathsf{Aborted}, \mathsf{T})$ in $\texttt{List}_{\texttt{tid}}$.
   (a) Communication from $\mathbb{M}$ to U:
   $\mathcal{S}$ sends $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_j, \sigma_j^{\mathsf{r},\mathfrak{B}}, \mathsf{mid}'_s)$ to $\mathcal{A}$ as leakages of $\mathcal{F}_{\mathsf{C}}^{\mathsf{sa}}$ in which $\sigma_j^{\mathsf{r},\mathfrak{B}}$ parses as $\sigma_w^{\mathsf{r},\mathfrak{B}}$ and $\sigma_t^{\mathsf{r},\mathfrak{B}}$. Also, $\sigma_w^{\mathsf{r},\mathfrak{B}}$ for $w \in \mathcal{H}$ is simulated by $\mathcal{S}$ and $\sigma_t^{\mathsf{r},\mathfrak{B}}$ for $t \in \mathcal{C}$ is obtained from the $\mathcal{A}$ (malicious maintainer).

C.5 Simulation of *Privacy Revocation*

$\mathcal{S}$ receives $(\texttt{RvkAnm}, \mathsf{sid}, t_{\mathsf{id}}^j, \mathsf{M}_j)$ from $\mathcal{F}_{\mathsf{CBDC}}$ and starts emulating honest maintainers.

Honest U_s, honest U_r, and at most t malicious maintainers for both *Currency Issuance* and *Payment* protocols: In the following, for simplicity we describe $\mathcal{S}$ for payment transactions (issuance transactions are similar and more straightforward).
$\mathcal{S}$ uses its internally maintained list $\texttt{List}_{\texttt{tid}}$ to find out the associated ciphertext (ψ_s, ψ_r) of the received t_{id} from $\mathcal{F}_{\mathsf{CBDC}}$.
$\mathcal{S}$ submits $(\texttt{RvkAnm.Ok}, \mathsf{sid}, t_{\mathsf{id}})$ to $\mathcal{F}_{\mathsf{CBDC}}$ and waits for the message $(\texttt{AnmRevoked}, \mathsf{sid}, t_{\mathsf{id}}, \mathsf{U}_s, \mathsf{U}_r, v)$ from $\mathcal{F}_{\mathsf{CBDC}}$.
Once $\mathcal{S}$ receives that message it starts faking the threshold decryption of ciphertexts based on keys it has registered for honest U_s and honest U_r. Specifically, $\mathcal{S}$ computes shares of honest maintainers in a way that threshold decryption of (ψ_s, ψ_r) result in associated values received from $\mathcal{F}_{\mathsf{CBDC}}$, pk_s and pk_r registered for U_s and U_r and v as described in details in Sec. 4.1.
The real-world $\mathcal{A}$ sees the leakages of $\mathcal{F}_{\mathsf{NIZK}}$ when honest maintainer (e.g., M_w) generates proof which is $(\texttt{Prove}, \mathsf{sid}, \mathtt{x}_w)$ for $\mathtt{x}_w = \left(c_{s,1}, c_{r,1}, c_{s,1}^{\mathsf{sk}_{1,w}} c_{s,1}^{\mathsf{sk}_{2,w}}, c_{r,1}^{\mathsf{sk}_{1,w}}\right)$. In $\mathtt{x}_w$, the values $c_{s,1}$ and $c_{r,1}$ are from (ψ_s, ψ_r), however, the values of $c_{s,1}^{\mathsf{sk}_{1,w}} c_{s,1}^{\mathsf{sk}_{2,w}}$ and $c_{r,1}^{\mathsf{sk}_{1,w}}$ are computed as it is described detailedly in Sec. 4.1.
$\mathcal{S}$ outputs the leakage of authenticated channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ which is $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_w, \mathsf{M}_i, (\mathtt{x}_w, \pi_w), \mathsf{mid})$ to $\mathcal{A}$ which is related to the calls honest maintainers make.
Emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$, upon receiving the message $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_i, (\mathtt{x}_t, \pi_t))$ from $\mathcal{A}$ (malicious maintainer M_t), $\mathcal{S}$ outputs $(\texttt{Send}, \mathsf{sid}, \mathsf{M}_t, \mathsf{M}_i, (\mathtt{x}_t, \pi_t), \mathsf{mid}')$ to $\mathcal{A}$.
Upon receiving messages of the form $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid})$ and $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid}')$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\texttt{Received}, \mathsf{sid}, \mathsf{M}_j, (\mathtt{x}_j, \pi_j))$ to $\mathcal{A}$ (or malicious maintainer M_t) where M_j includes honest maintainer M_w and malicious maintainer M_t respectively.
Moreover, upon receiving $(\texttt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}')$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\texttt{Continue}, \mathsf{sid})$ to M_t.
$\mathcal{S}$ also leaks $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{pk}_s, \mathsf{M}_j)$ and $(\texttt{RetrieveID}, \mathsf{sid}, \mathsf{pk}_r, \mathsf{M}_j)$ to $\mathcal{A}$ upon receiving the associated calls from maintainer M_j.

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 the fact that there is no need for changing the shares. The reason is that in this case $\mathcal{S}$ knows the identities of participants U_s and U_r, and also transaction value v.
$\mathcal{S}$ on behalf of honest maintainers computes decryption shares and all participant maintainers in the *Privacy Revocation* protocol use their decryption shares to obtain the associated public keys and value as described in the construction details.

C.6 Simulation of *Tracing*

$\mathcal{S}$ receives $(\mathtt{Trace}, \mathsf{sid}, \mathsf{U}_j, \mathsf{M}_j)$ from $\mathcal{F}_{\mathsf{CBDC}}$ and emulates honest maintainers.

Honest U and at most t malicious maintainers for both *Currency Issuance* and *Payment* protocols: $\mathcal{S}$ submits $(\mathtt{Trace.Ok}, \mathsf{sid}, \mathsf{U})$ to $\mathcal{F}_{\mathsf{CBDC}}$ and upon receiving $(\mathtt{Traced}, \mathsf{sid}, \{t_{\mathsf{id}}^{\tau}, \mathsf{role}^{\tau}\}_{\tau=1}^{x})$ gets to know $\{t_{\mathsf{id}}^{\tau}, \mathsf{role}^{\tau}\}_{\tau=1}^{x}$ which are required for simulating honest maintainers' shares such that tracing tag computation results in tags (that are random values that were selected by $\mathcal{S}$ in issuance and payment transactions) associated to $\{t_{\mathsf{id}}^{\tau}\}_{\tau=1}^{x}$.
In the Simulation of *Currency Issuance* and Simulation of *Payment* we described that $\mathcal{S}$ randomly selects z. Hence, $\mathcal{S}$ should simulate the result of threshold tag computation to be consistent with values $\{g^{z_\tau}\}_{\tau=1}^{x}$. The simulator does so as described in the Sec. 4.1.
$\mathcal{S}$ uses its internally maintained list $\mathtt{List}_{\mathtt{UR}}$ to retrieve $\mathsf{UR} = (a_j, r_j, \mathsf{M}_j, \mathsf{U})$. Then, computes com_w (on behalf of the honest maintainer M_w). It outputs $(\mathtt{Prove}, \mathsf{sid}, \overline{\mathtt{x}}_w)$ to $\mathcal{A}$ where $\overline{\mathtt{x}}_w = (\mathsf{com}_w, \dot{g}^{a_w}, \dot{g})$ see Sec. 4.1 for details of computing $\dot{g}^{a_w}$.
$\mathcal{S}$ outputs the leakage of authenticated channel $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ which is $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_w, \mathsf{M}_i, (\overline{\mathtt{x}}_w, \overline{\pi}_w), \mathsf{mid})$ to $\mathcal{A}$ which is related to the calls honest maintainers make.
Emulating $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$, upon receiving the message $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_i, (\overline{\mathtt{x}}_t, \overline{\pi}_t))$ from $\mathcal{A}$ (malicious maintainer M_t), $\mathcal{S}$ outputs $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_t, \mathsf{M}_i, (\overline{\mathtt{x}}_t, \overline{\pi}_t), \mathsf{mid}')$ to $\mathcal{A}$.
Upon receiving messages of the form $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ and $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid}')$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\mathtt{Received}, \mathsf{sid}, \mathsf{M}_j, (\overline{\mathtt{x}}_j, \overline{\pi}_j))$ to $\mathcal{A}$ (or malicious maintainer M_t) where M_j includes honest maintainer M_w and malicious maintainer M_t respectively.
Moreover, upon receiving $(\mathtt{Ok.Snd}, \mathsf{sid}, \mathsf{mid}')$ from $\mathcal{A}$, $\mathcal{S}$ sends $(\mathtt{Continue}, \mathsf{sid})$ to M_t.
Emulation of $\mathcal{F}_{\mathsf{C}}^{\mathsf{ac}}$ for the rest of the protocol (e.g., calls with input $(\mathtt{Send}, \mathsf{sid}, \mathsf{M}_i, (0, \dot{g}^{a})))$ is similar to the calls above.

Malicious U and less than min (α, β) malicious maintainers for both *Currency Issuance* and *Payment* protocols: The simulation of this case is similar to the case above except the fact that there is no need for changing the shares of honest maintainers. $\mathcal{S}$ on behalf of honest maintainers participate at computing the tracing tags as described in the construction.

D Security Definitions of PEReDi's Building Blocks

D.1 d-sDDH Assumption

Definition 2. *We say that the d-strong Diffie-Hellman problem is hard relative to $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$ there exists a negligible function* negl(.) *such that:*

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

$\leq \mathsf{negl}_{\mathsf{d\text{-}sDDH}}(\lambda)$

where $(\mathbb{G}, p, g) \leftarrow \mathcal{G}\left(1^{\lambda}\right)$ *and the probabilities are taken over the choices of* $(x, x_1, \ldots, x_d) \xleftarrow{\$} \mathbb{Z}_p$.

D.2 Security Properties of Commitment Scheme

Let $\mathsf{com} = (\mathsf{com.Setup}, \mathsf{Commit}, \mathsf{com.Vrf})$ be a commitment scheme.

Definition 3. *For any PPT adversary* $\mathcal{A}$*, the hiding property is defined as the following security experiment between* $\mathcal{A}$ *and a challenger parameterized by a bit* $b \in \{0,1\}$*:*

$\mathsf{Hid\text{-}com}(\mathcal{A}, \lambda)$*:*

1. *The challenger runs* $\mathsf{PubPar} \xleftarrow{\$} \mathsf{com.Setup}(1^{\lambda})$ *and outputs* PubPar *to* $\mathcal{A}$.
2. $\mathcal{A}$ *gives two messages* (m_0, m_1) *such that* $m_0 \wedge m_1 \in \mathcal{M}$ *to the challenger.*
3. *The challenger computes* $(\mathsf{com}_b; r) = \mathsf{Commit}(m_b)$ *and outputs* com_b *to* $\mathcal{A}$.
4. $\mathcal{A}$ *outputs a bit* b' *to the challenger.*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hid\text{-}com}} = \left| \frac{1}{2} - \Pr[\mathsf{Hid\text{-}com}(\mathcal{A}, \lambda) \ s.t. \ b' = b] \right| \leq \mathsf{negl}_{\mathsf{com}}(\lambda)$$

We say that commitment scheme com *is perfectly hiding if* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hid\text{-}com}} = 0$.

Definition 4. *For any PPT adversary* $\mathcal{A}$*, the binding property is defined as the following security experiment between* $\mathcal{A}$ *and a challenger parameterized by a bit* $b \in \{0,1\}$*:*

$\mathsf{Bind\text{-}com}(\mathcal{A}, \lambda)$*:*

1. *The challenger runs* $\mathsf{PubPar} \xleftarrow{\$} \mathsf{com.Setup}(1^{\lambda})$ *and outputs* PubPar *to* $\mathcal{A}$.
2. $\mathcal{A}$ *outputs* $(\mathsf{com}, m_0, m_1, r_0, r_1)$.

$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind\text{-}com}} = \Pr[\mathsf{Bind\text{-}com}(\mathcal{A}, \lambda) \ s.t. \ \mathsf{com.Vrf}(\mathsf{com}, m_0, r_0) = 1 \ \wedge \ \mathsf{com.Vrf}(\mathsf{com}, m_1, r_1) = 1 \ \wedge \ m_0 \neq m_1] \leq \mathsf{negl}_{\mathsf{com}}(\lambda)$

We say that commitment scheme com *is perfectly binding if* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind\text{-}com}} = 0$.

D.3 CPA Security of Public Key Encryption Scheme

Definition 5. *Let* $\mathsf{PKE} = (\mathsf{PKE.Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public key encryption scheme. The following security experiment between PPT adversary* $\mathcal{A}$ *and a challenger is parameterized by a bit* $b \in \{0,1\}$*:*

$\mathsf{IND\text{-}CPA}^{b}_{\mathsf{PKE}}(\mathcal{A}, \lambda)$*:*

1. *The challenger runs* $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{PKE.Gen}(1^{\lambda})$ *and outputs* pk *to* $\mathcal{A}$.
2. $\mathcal{A}$ *gives two messages* (m_0, m_1) *such that* $|m_0| = |m_1|$ *to the challenger.*
3. *The challenger computes* $c_b = \mathsf{Enc}_{\mathsf{pk}}(m_b)$ *and outputs* c_b *to* $\mathcal{A}$.
4. $\mathcal{A}$ *outputs a bit* b' *to the challenger (if* $\mathcal{A}$ *aborts without giving any output, we set* $b' \leftarrow 0$*).*

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

D.4 Existential Unforgeability of Digital Signature Scheme

Definition 6. *Let* $\mathsf{DS} = (\mathsf{DS.Gen}, \mathsf{Sign}, \mathsf{Verify})$ *be a digital signature scheme. Existential Unforgeability under Chosen-Message Attack (EUF-CMA) is defined using the following game between PPT adversary* $\mathcal{A}$ *and the challenger:*

$\mathsf{EUF\text{-}CMA}_{\mathsf{DS}}(\mathcal{A}, \lambda)$*:*

1. *The challenger runs* $(\mathsf{vk}, \mathsf{sk}) \overset{\$}{\leftarrow} \mathsf{DS.Gen}(1^\lambda)$ *and gives the adversary* $\mathcal{A}$ *the resulting verification key* vk *and keeps the secret key* sk *to itself.*
2. *The adversary* $\mathcal{A}$ *submits signature queries for* $\{m_\tau\}_{\tau=1}^{q}$. *To each query* m_τ *the challenger responds by running* Sign *to generate a signature* σ_τ *of* m_τ *and sending* σ_τ *to the adversary* $\mathcal{A}$.
3. *The adversary* $\mathcal{A}$ *outputs a pair* (m, σ) *and wins if* σ *is a valid signature of* m *according to* Verify *and* (m, σ) *is not among the pairs* (m_τ, σ_τ) *generated during the query phase.*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} = \Pr[\mathsf{EUF\text{-}CMA}_{\mathsf{DS}}(\mathcal{A}, \lambda)] \leq \mathsf{negl}_{\mathsf{DS}}(\lambda)$$

D.5 Bilinear Maps

The threshold blind signature employed in PEReDi uses bilinear maps. Assuming that $(\mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t)$ are groups of prime order p, we define a map $e : \mathbb{G} \times \tilde{\mathbb{G}} \to \mathbb{G}_t$ with the following properties:

- Bilinearity: for all $g \in \mathbb{G}$, $\tilde{g} \in \tilde{\mathbb{G}}$, and $(x, y) \in \mathbb{F}_p^2$, $e(g^x, \tilde{g}^y) = e(g, \tilde{g})^{xy}$ holds.
- Non-degeneracy: for all generators $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$, $e(g, \tilde{g})$ generates $\mathbb{G}_t$.
- Efficiency: there exists an efficient algorithm $\mathcal{G}\left(1^\lambda\right)$ that outputs the pairing group setup $\left(p, \mathbb{G}, \tilde{\mathbb{G}}, \mathbb{G}_t, e, g, \tilde{g}\right)$ and an efficient algorithm to compute $e(g, \tilde{g})$ for any $g \in \mathbb{G}$ and $\tilde{g} \in \tilde{\mathbb{G}}$. In type 3 pairings, $\mathbb{G} \neq \tilde{\mathbb{G}}$ and there exists no efficiently computable homomorphism $f : \tilde{\mathbb{G}} \to \mathbb{G}$.