DART: Decentralized, Anonymous, and Regulation-friendly Tokenization

Amirreza Sarencheh[1], Hamidreza Khoshakhlagh[2], Alireza Kavousi[3], and Aggelos Kiayias[4]

[1] The University of Edinburgh, UK
amirreza.sarencheh@ed.ac.uk

[2] Partisia and Aarhus University, Denmark
hamidreza@cs.au.dk

[3] University College London, UK
a.kavousi@cs.ucl.ac.uk

[4] The University of Edinburgh and IOG, UK
aggelos.kiayias@ed.ac.uk

Abstract. We introduce DART, a *fully anonymous*, account-based payment system designed to address a comprehensive set of real-world considerations, including regulatory compliance, while achieving *constant transaction size*. DART supports *multiple asset types*, enabling users to issue on-chain assets such as tokenized real-world assets. It ensures confidentiality and anonymity by concealing asset types, transaction amounts, balances, and the identities of both senders and receivers, while guaranteeing unlinkability between transactions. Our design provides a mechanism for *asset-specific auditing*. Issuers can designate asset-specific auditors for the assets they issue, with the system preserving the privacy of the auditor's identity to achieve asset type privacy. Only the designated auditor is authorized to decrypt transactions related to their associated asset, and users efficiently prove the association between the (hidden) asset type and the (hidden) designated auditor in their transactions. DART supports *non-interactive payments*, allowing an online sender to submit a transaction even when the receiver is offline, while still incorporating a *receiver affirmation* mechanism that captures the real-world compliance consideration where the receiver must confirm (or deny) an incoming transaction. To the best of our knowledge, this is the first scheme of this kind in the permissionless setting. To accommodate all eventualities, DART also incorporates a *reversibility* mechanism, enabling senders to reclaim funds from pending transactions if the receiver's affirmation is not yet provided. Finally, it offers a privacy-preserving *proof of balance* (per asset type) mechanism. Our system achieves full anonymity while supporting concurrent incoming and outgoing transactions, resolving a common issue that plagues many account-based anonymous systems. We further demonstrate how our system supports *multi-party transactions*, allowing payment to multiple receivers in one transaction efficiently. We provide a full formal model in the Universal Composition (UC) setting, as well as a UC protocol realization.

Keywords: Digital Asset Transfer, Privacy, Regulatory Compliance, Real-World Assets, Tokenization, Anonymous Payments, Universal Composition.

Table of Contents

1 Introduction

A fully anonymous payment system ensures that users remain hidden within the entire set of system participants. Specifically, a payment transcript provides the counterparties of any transaction with an anonymity set as large as the total number of users in the system. Several existing blockchain systems, such as Zerocash [49] and Ouroboros Crypsinous [31], achieve this level of anonymity under varying assumptions. A common characteristic of these systems, however, is that they utilize coin-based bookkeeping, sometimes referred to as UTXO-based bookkeeping[5]. Intuitively, this means that during a payment protocol, users spend existing coins and create new coins (of specified amounts) to facilitate value transfer. An inherent consequence of this coin-based operation is that the token holdings of any individual account holder are dispersed across the various coins they control. It is important to note that this coin-based approach is not exclusive to "privacy coins" —Bitcoin [41], for example, also employs UTXO-based bookkeeping.

In contrast to the coin-based approach discussed above, the account-based approach employed by systems like Ethereum [10] offers a different method of managing transactions. In an account-based system, each account has a unique identifier which is publicly known, and senders transfer funds directly from their accounts by reducing their balances and crediting the recipient's account. A key advantage of this approach is that the system state has a straightforward representation, such as a table displaying each account's balance. Consequently, various account-related operations can be significantly simpler to perform compared to the UTXO model, where a user's balance is distributed.

The high-level idea behind confidential and anonymous payment schemes, whether UTXO-based or account-based, is for the sender to generate a Zero-Knowledge Proof (ZKP) that demonstrates the well-formedness of the transaction. The transaction conceals the transferred value within a commitment or encryption. In the ZKP, the sender proves the knowledge of secret values required for payment authorization, and proves the transferred value is positive and lies within the appropriate range as dictated by the input and output balances of the coins or accounts involved. Finally, double-spending is prevented by recording transaction-specific cryptographic objects, e.g., nullifiers, or by (homomorphically) reducing the sender's balance as soon as the transaction is received.

From a privacy perspective, the account-based approach poses significant challenges in achieving anonymity. Systems like QuisQuis [24][6], Zether [8], Anonymous Zether [19], and PriDe CT [28] restrict the sender's anonymity set to k out of n, where $k \ll n$ is a security parameter, and their transaction size is $\mathcal{O}(k)$.[7] The primary challenge arises from the requirement that, for full anonymity, every transaction must interact with the complete ledger state. If certain accounts

[5] UTXO stands for "unspent transaction output".

[6] Note that QuisQuis is an account-based cryptocurrency that also uses UTXOs.

[7] For instance, k is 64 for QuisQuis and 256 for anonymous Zether. For an anonymity set of 16, the transaction size for QuisQuis is 26 KB, whereas for anonymous Zether, it is 6 KB. Note that, the maximum block size is assumed to be 100 KB [34].

are excluded from being updated during a transaction, the anonymity of that transaction is compromised, as a small k opens up the possibility of privacy attacks (e.g., see [33,40]). The limitation on the anonymity set is inherent to these systems, as the value of k is constrained by the maximum transaction size that a block can hold. Another challenge is the need to carefully choose the accounts that are part of the anonymity set as the level of anonymity can be greatly diminished depending on the choice of these accounts.

Standard digital currency transactions (e.g., in Bitcoin [41]) allow the receiver to obtain the funds immediately upon blockchain validators processing the transaction. Moving beyond simple payments, the concept of *receiver affirmation* is a well-known regulatory requirement in broader financial transactions such as securities transfers, emphasized by the US Securities and Exchange Commission (SEC) [55], and others [52], [44], [53], [45], [6], [54]. This mechanism grants receivers the authority to accept or reject incoming payments before their accounts are updated. Receiver affirmation empowers receivers to manage their obligations proactively and is particularly important for transactions with legal or tax implications, where receivers assume specific responsibilities upon receiving an asset.

Accommodating receiver affirmation introduces additional challenges, such as facilitating *proof of balance* (PoB). PoB allows any verifier to request the balance of a specific asset under a specific public key or address. A related concept is proof of assets (PoA) [15], which enables a prover to convince a verifier that they possess *at least* a certain amount of assets. Unlike PoB, which requires revealing the *exact* balance in the user's account, PoA focuses solely on proving sufficiency. With receiver affirmation, PoB becomes non-trivial as the sender's balance in their account is split while waiting for the receiver to come online. Depending on the receiver's decision (affirming or rejecting), the funds can either be reversed (added back to the sender's balance) or transferred to the receiver's account balance. Also, receiver affirmation introduces another challenge that addresses scenarios in which the receiver does not affirm the transaction. The system should support *reversibility*, ensuring that if a receiver does not affirm, the sender can reclaim the funds. By allowing the reclamation of unaffirmed funds, reversibility prevents assets from being lost. Moreover, maintaining *simultaneous transactions* allows senders to engage in multiple transactions simultaneously with different parties, without waiting for pending transactions to be affirmed or reversed. This ensures high throughput in the payment system.

In the anonymous payment literature, specifically account-based models, a well-known issue referred to as *concurrency issues* may arise, given that the ZKP of users depends on their current account state [1, 2, 8, 34]. To illustrate the challenge consider that proving a sender's sufficient balance for an outgoing transfer requires referencing a valid account state. However, if the account state changes (for instance, the account is a recipient of another user's transaction which modifies its cryptographic form, e.g., by rerandomizing it with zero——a step necessary to achieve anonymity in some schemes), the sender's ZKP becomes inadmissible. Consequently, the sender's transaction has to be rejected,

and the user may also lose the associated transaction fees. We call a system that circumvents this problem as satisfying concurrency of incoming/outgoing transactions (CIO). To address this problem, a periodic *pending transfers* mechanism can be introduced, where incoming transfers are held in a pending state and then rolled into accounts periodically at the end of fixed length epochs (time periods consisting of a set number of blocks). By requiring users to publish transactions at the beginning of an epoch and ensuring that pending transfers are processed only at epoch boundaries, the design preserves ZKP validity and prevents conflicts caused by account state changes during transaction processing. At the beginning of each epoch, the sender must query the blockchain to obtain their most updated account state before generating their ZKP. To mitigate double spending, each user is restricted to generating at most *one* transaction per epoch (note that the identity of the sender is hidden inside the anonymity set). While longer epochs slow down transaction generation, shorter epochs increase the likelihood of ZKP invalidation if another user's transaction rerandomizes the sender's account state.

For a payment system, it is important to support *non-interactivity*, allowing senders to initiate payments without requiring the receiver to be online at the same time. This ensures that transactions are not hindered by the receiver's immediate presence. Additionally, the ability of a sender to send to multiple receivers and a receiver to receive (in our context affirm) from multiple senders in a single transaction (referred to as a multi-party transaction (MPT), or batchability [28]) becomes particularly important in privacy-preserving payments. Since the cost of each transaction is generally higher than in a non-privacy-preserving one, supporting MPT increases efficiency. Furthermore, multiple assets support refers to the capability of a protocol to manage, process, and facilitate transactions involving various types of digital assets within a system. Ensuring privacy for the asset type being transferred is also an important consideration.

Regulatory compliance issues arise with full privacy, as auditors may lack access to necessary information when all data is encrypted. While ZKPs can enforce compliance on-chain without the need for encrypting information under an auditor's public key, decisions involving human discretion, such as judicial rulings, are subjective and challenging (or may be even impossible in some cases) to formalize mathematically. Thus, auditors may require direct access to the underlying transaction data. Furthermore, verifying certain regulatory policies that rely heavily on off-chain data can be costly and impractical to facilitate entirely on-chain. In a system with multiple asset types (potentially spanning different jurisdictions), auditing necessitates careful considerations, such as addressing *asset-specific auditing* mechanisms to ensure that only the designated auditors associated with each asset are granted access to perform their audits.

Despite significant advances in anonymous payment systems [4], current solutions face critical challenges in achieving a practical balance between full anonymity, addressing real-world considerations, and regulatory compliance. Reconciling the need for receiver affirmation, PoB, non-interactivity, MPT, support for multiple asset types, circumventing concurrency issues, and asset-specific

auditing in a privacy-preserving manner, preferably in a permissionless setting remains an open area of investigation. Addressing these limitations altogether requires a novel approach that draws from the best features of existing systems while overcoming their inherent drawbacks, leading to the design of a versatile, anonymous, and permissionless payment system as we do in this paper.

1.1 Our results

We design a **D**ecentralized, **A**nonymous, and **R**egulation-friendly **T**okenization system, called DART, which is suitable for a permissionless setting. Our system, DART, not only achieves full anonymity but also addresses a number of critical considerations, making it suitable for real-world applications. DART supports the issuance of on-chain representations of real-world assets (i.e., tokenization) as well as the creation of on-chain native assets. Users can act as issuers, minting and distributing digital assets. DART exhibits strong privacy-preserving properties by concealing sensitive information such as transaction values, user balances, public keys, and asset types, while ensuring unlinkability between transactions, thus hiding the transaction graph. Despite these privacy guarantees, DART facilitates publicly verifiable state transitions by settling transactions on a public ledger maintained permissionlessly.

Our system builds upon the *account-based* model of PEReDi [47] and Platypus [56] as well as the *unlinkability* techniques of Zerocash [49] while overcoming their inherent drawbacks in our setting. In PEReDi and Platypus each user holds an account and for each transaction, they update their account and efficiently prove in zero-knowledge that the update was performed correctly. However, these account state transitions are authorized by (possibly distributed) validator using a blind re-randomizable signature. Importantly, the public key of the validator must be *fixed* and known to all network participants. In Zerocash, on the other hand, the idea for achieving anonymity is to generate an anonymity set that includes all coins ever introduced into the system and efficiently prove knowledge of a (fresh) UTXO as a member of this anonymity set, without revealing it. While this aligns well with permissionless operation it appears to be inherently dependent on a coin/UTXO oriented mode of operation.

A natural crossover of the two approaches is to generate an anonymity set comprising all *account* states with transactions of constant size. Whenever a user wants to perform an account state transition, they prove knowledge of an unused (old) account state in the ledger and submit their new account state. The validators can verify that the old account is not being reused (e.g., preventing double-spending) without seeing the old account itself, allowing the user's new account to be added to the ledger. Importantly, the old account remains hidden from validators to ensure unlinkability. To prevent double-spending, a deterministic *nullifier* (also known as a tag) is derived and revealed from the account state. This nullifier is recorded on the ledger, ensuring that each account *state* can be used only once. The nullifier set on-chain grows linearly with respect to the number of transactions. Note, however, that this account update process requires both the sender and receiver to be *online* to update their respective

accounts at the same time, as only the users themselves have the necessary ZKP witness to make the account updates.

In our construction DART, we adopt this crossover approach but obviate the need for interactivity. This is achieved as follows: when the sender submits a transaction, an ephemeral *account-update-record* (AUR) is generated for the (possibly offline) receiver, who can claim it anytime later when they will update their account. The AUR remains pending (as "funds in transit") until the receiver affirms it or the sender reclaims it using *reversibility* mechanism. Importantly, the receiver affirmation step (which claims an AUR) does not reveal any details about the receiver. It is worth highlighting that the notion of *ephemeral* AUR in DART crucially differs from a coin or UTXO based approach. Particularly, an AUR itself is *not* spendable by the receiver and may be reclaimed by the sender at any time, prior to the receiver affirms and incorporates it into their account.

To prevent double-spending, the sender's balance is debited so that the sender can initiate other transactions while any previous ones are pending. Additionally, the sender can initiate an *efficient* AUR reversion protocol *anytime* before the receiver affirms AUR, thereby preventing indefinite resource locking. Note that once the sender reclaims the funds the receiver becomes permanently unable to claim them as their own. Additionally, a malicious sender cannot reclaim an AUR more than once. Note that, *by definition*, to achieve receiver affirmation there is a need for the receiver to submit a transaction/affirmation on-chain. Hence, for a DART transaction to be considered finalized, two transactions are necessary: one from the sender and one from the receiver. Importantly, in DART, these two transactions do not need to be concurrent.

To address regulatory compliance, in our construction, asset issuers can designate *asset-specific* auditors for their assets. When transacting with a specific asset type, users must include the designated auditor (if any) in the transaction, ensuring that the auditor can decrypt the required data. Only the auditor registered for that asset type can decrypt the transaction data, while other auditors remain oblivious to the transaction details. Achieving asset-type privacy in such a regulated framework presents challenges. Crucially, to maintain asset type privacy, it is necessary to also conceal the public key of the auditor involved in the transaction, as it could reveal the underlying asset type due to the existence of a publicly known mapping between asset types and their associated auditors. Thus, the challenge is twofold: (i) to hide both the asset type and the corresponding auditor's public key[8], (ii) to enable the sender to *efficiently* prove that the hidden public key belongs to the correct auditor for the asset being transferred.

We note that the sender and receiver could, in principle, agree *off-chain* by privately communicating to update their account states and subsequently submitting a proof on-chain to prove that the state transition was executed correctly. However, our system intentionally avoids this approach for several reasons: (i)

[8] We need *key-privacy* [3], as CPA-security alone is insufficient for the problem at hand. Without key-privacy, the encryptions under auditors' public keys could reveal the public keys, thereby compromising privacy.

We do not assume the existence of an off-chain communication channel between the sender and receiver at the time of value transfer, although such an assumption is common in the literature (see e.g., [29]). Note that, with this assumption, the receiver in DART can operate more efficiently, as they no longer need to scan the ledger to identify the receipt of AUR. (ii) This approach would necessitate interactive payments to avoid concurrency issues (as discussed before) as there is no way for on-chain validators to verify account state transitions anonymously. Therefore, we adopt the approach of requiring two on-chain submissions for each payment. This is crucial for maintaining asynchrony between sender and receiver as well as avoiding concurrency issues.

Supporting multiple assets in users' accounts requires careful consideration, particularly in maintaining the hiding properties. The subtlety lies in minimizing the overhead on transaction generation for the sender's ZKP. For instance, if all asset types were to be included within a single account, the witness for the sender's proof in every account state transition would need to encompass all asset types associated with that account. This approach would result in witnesses proportional to the total number of assets supported by the system. Although SNARKs can provide succinct proof sizes, the proving time would remain considerably high since the proof generation complexity scales linearly with the witness size[9]. To address this, we adopt the use of asset-specific accounts. This approach confines the witness for transaction generation to only the relevant asset type, thereby reducing the computational overhead involved in proof generation[10].

DART efficiently facilitates *multi-party transactions* (MPT), namely, transfer of a specific asset to multiple receivers in a single transaction, and further receipt/affirm from multiple senders within a single transaction, thereby enhancing efficiency (also known as batchability [28]). Moreover, DART supports a proof of balance (PoB) mechanism, which allows any party to request the balance from some user given an account identifier. Users register asset-specific accounts, and the ledger maintains mappings between account identifiers and their associated asset types. A user can prove their balance for a particular asset without revealing any additional details. It is crucial to ensure that a malicious user cannot bypass the PoB protocol. For instance, a malicious sender could initiate a transaction reducing their *spendable* finalized balance, then engage with the PoB verifier and subsequently reverse the pending transaction without the PoB verifier being aware of the pending transaction. To prevent such an exploit, the system enforces the sender to update their (non-spendable) pending balance

[9] Notably, while the witness size grows linearly with the number of supported assets, most witness elements are likely zero for typical users who interact with only a subset of assets. This raises an intriguing direction for future work: leveraging witness sparsity. Employing recent techniques such as sparse polynomial commitments [50, 51], it is worth exploring proof systems whose efficiency—particularly in proof generation—depends mainly on the number of non-zero witness elements.

[10] Having asset-specific accounts leads to linear on-chain storage growth proportional to the number of assets linked to each key. In contrast, consolidating all assets within a single account can reduce storage requirements. However, our system prioritizes sender proof efficiency over on-chain storage minimization in this trade-off.

within their account (hidden) data. This ensures that information about the pending balance is also available to the PoB verifier, allowing them to account for the possibility of potential reversion in the future. Moreover, our reversibility functionality provides an incentive for the receiver to claim their AUR promptly, as the sender retains the ability to reverse the transaction at any time. Not only the sender has financial incentives to do so but also it is preferable for the sender to have no pending transactions for privacy reasons (we elaborate on this later). This creates another incentive for the sender to reverse any pending transactions which in turn results in the receiver losing access to the funds.

Summary of our contributions. With DART, we make the following contributions:

- *Full anonymity and confidentiality in an account-based model with constant transaction size.* We formalize the notion of a payment system that ensures privacy of balances, transaction values, and the identities/public keys of both senders and receivers, while also guaranteeing unlinkability between transactions. Moreover, payments in DART obscure the underlying asset types and the identities of auditors. Our system operates under an account-based bookkeeping model that differs from other account-based systems with respect to anonymity guarantees in that they are: (i) *partially anonymous*, such as QuisQuis [24], Zether [8], Anonymous Zether [19], and PriDe CT [28]. (ii) *permissioned*, like PEReDi [47] and PARScoin [48]. (iii) *centralized* like Platypus [56]. (iv) *inherently inefficient* where the validator has complexity proportional to the account set for each transaction and have *concurrency issues* like PriFHEte [34].
- *Support for receiver affirmation, proof of balance, reversibility, and simultaneous transactions.* We formalize the concept of *receiver affirmation* in digital currencies, enabling receivers to affirm incoming payments before account updates, thus aligning blockchain-based payments with well-known regulatory requirements in financial transactions and empowering receivers to manage obligations, particularly in scenarios involving legal or tax implications. We address the challenges introduced by incorporating receiver affirmation, specifically by: (i) introducing *proof of balance* (PoB) protocol to enable verifiers to check exact balances, while accommodating the complexities of split balances during receiver affirmation. (ii) ensuring *reversibility*, allowing senders to reclaim unaffirmed funds to prevent indefinite asset locking. (iii) supporting *simultaneous transactions*, enabling senders to engage in multiple transactions concurrently, while waiting for previous transactions' receiver affirmations.
- *Support for non-interactivity, multi-party transaction, multiple asset-types, and asset-specific auditing while avoiding concurrency issues.* We support *non-interactivity* for offline receivers, *multi-party transactions* (MPT) to enable batch transactions for improved efficiency in privacy-preserving settings, and *multiple asset-types* to handle a diverse set of digital assets while ensuring privacy. We address regulatory compliance challenges in privacy-preserving systems by incorporating encrypted transaction data with au-

ditor access when necessary, enabling *asset-specific auditing* to ensure jurisdictional alignment and granting designated auditors access. Finally, our design does not suffer by the *concurrency issues* common in account-based anonymous payments as discussed above.

- *Universal composition (UC) security and efficient instantiation.* We formalize decentralized and fully anonymous payments with all the properties described above in a Universal Composition (UC) [13] framework via our ideal functionality $\mathcal{F}_{\mathsf{DART}}$ and provide an efficient realization of $\mathcal{F}_{\mathsf{DART}}$ via our construction Π_{DART}. This ensures that Π_{DART} can be securely integrated with other decentralized finance (DeFi) systems or traditional finance (TradFi) legacy systems. In our construction Π_{DART}, we employ well-established cryptographic schemes and we utilize two non-interactive zero-knowledge proof systems: Σ-protocols and Bulletproofs [9], to efficiently verify statements on algebraically encoded data.

Overview of the related works. Early research highlighted that Bitcoin [41] provides only pseudonymity rather than anonymity [35, 36]. This limitation spurred significant academic and practical efforts to address the issue, leading to the development of a rich body of privacy-preserving payment systems each operating under distinct assumptions, settings, and properties. These systems can be broadly categorized into two classes: UTXO-based systems and account-based systems. In the UTXO-based category, notable examples include Zerocoin [39], Zerocash [49], and Monero [43]. On the other hand, examples of account-based systems include Zether [8], Anonymous Zether [19], Platypus [56], PEReDi [47], PriDe CT [28], PARScoin [48], and PriFHEte [34]. We present an overview of these systems as well as how they compare to DART in Table 1; more details are provided in Section 2.

2 Related works

Among the first pioneering approaches to privacy enhanced payments was Zerocash [49], which builts upon the earlier protocol Zerocoin [39]. These works introduced the concept of *decentralized anonymous payments*, proposing a payment system that guarantees strong confidentiality, anonymity, and public verifiability. Zerocash, a UTXO-based fully anonymous payment system, employs commitments and zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs) to conceal transaction details, including the transaction value and the identities or public keys of both the sender and the receiver. Each transaction in Zerocash includes the value of a cryptographic accumulator that aggregates all coins ever introduced into the system, thereby forming the anonymity set. The sender proves ownership of a coin within this set by generating a zk-SNARK proof, without explicitly revealing the specific coin being spent. Notably, the size of each transaction remains constant independent of the size of the anonymity set different from majority of (partially) anonymous account-based systems. To prevent double-spending, each coin is associated with a unique nullifier, which

	Full anonymity	Proof of balance	Concurrency	Receiver affirmation	Non-interactivity	Multi party transaction	Transaction size
Zerocash [49]	✓	✗	✓	✗	✓	✗	$\mathcal{O}(1)$
Monero [43]	✗	✗	✓	✗	✓	✓	$\mathcal{O}(k)$
zkLedger [42]	✓*	✓	✓	✗	✓	✓	$\mathcal{O}(k)$
QuisQuis [24]	✗	✗	✗	✗	✓	✓	$\mathcal{O}(k)$
Zether [8]	✗	△	✗	✗	✓	✗	$\mathcal{O}(k)$
Ano. Zether [19]	✗	△	✗	✗	✓	✓	$\mathcal{O}(k)$
Veksel [11]	✗	✗	✓	✗	✓	✗	$\mathcal{O}(1)$
Platypus [56]	✓	△	✓	✓*	✗	✗	$\mathcal{O}(1)$
PEReDi [47]	✓	△	✓	✓*	✗	✗	$\mathcal{O}(1)$
PriDe CT [28]	✗	△	✗	✗	✓	✓	$\mathcal{O}(k)$
PARScoin [48]	✓	✗	✓	✗	✓	✗	$\mathcal{O}(1)$
Ocash [29]	✓	✗	✓	✗	✓	✓	$\mathcal{O}(1)$
PriFHEte [34]	✓	△	✗	✗	✓	✗	$\mathcal{O}(1)$
DART	✓	✓	✓	✓	✓	✓	$\mathcal{O}(1)$

Table 1: Overview of privacy-preserving payment systems and their properties.
Full anonymity (✓*: The system only supports a small number of users.)
Proof of balance (△: The scheme does not offer the property, but it can be achieved with reasonable modifications to the construction.)
Concurrency: concurrency of incoming/outgoing transactions
Receiver affirmation and reversibility (✓*: The system has interactive transactions, hence receiver affirmation is trivially addressed.)
*Note: additional properties such as **permissionlessness**, **multiple assets support**, and **asset specific auditing** are not included in this table to maintain clarity and conciseness. For a more detailed comparison, refer to Section 2.*

is publicly revealed upon spending as in Chaum's original blind signature-based e-cash [16].

One of the main distinctions between DART and Zerocash [49] is that PoB is not supported by Zerocash, as the user's balance in Zerocash is distributed across (potentially many) UTXOs, which a malicious receiver may selectively omit during a PoB procedure. Note that once a sender submits their transaction, the ownership of the funds is transferred to the receiver. In contrast, in DART, ownership is *not* transferred to the receiver until they claim AUR and update their account balance accordingly (i.e., AURs are not spendable). Until that moment, the sender retains the ability to reclaim AUR at any time via reversibility operation. Furthermore, the PoB prover has to disclose any pending funds. The authors of Zerocash assume two inputs and two outputs for sim-

plicity, but their scheme can be extended to n inputs and m outputs. As such, Zerocash can in principle support MPT. However, for n UTXOs consumed as inputs, the user must provide n accumulator membership proofs, which constitute the computationally costly part of the user's ZKP. In contrast, our approach allows a receiver to claim multiple AURs within a single account state transition, requiring only *one* accumulator membership proof via a *single* transaction. Similarly, a sender can provide a single accumulator membership proof while simultaneously creating multiple AURs intended for different receivers within a single transaction. Note that the number of AURs is upper-bounded by the block size. For more details on our approach to MPT, see Section 5.1.

Monero [43] employs a UTXO-based model to ensure transaction anonymity. It leverages ring signatures, where the anonymity set consists of a group of public keys included in the signature's ring and a transaction grows proportionally to that. In QuisQuis [24], each user maintains an account, and transactions utilize UTXO inputs rather than direct account debits. To preserve anonymity, Quisquis incorporates anonymity sets with updatable keys that provide unlinkability.

To support auditing functionalities, zkLedger [42] introduced a ledger representation using a table-based structure, where each row corresponds to a transaction with entries for all participants in the system. Each transfer is recorded as a transaction containing commitments to values, depending on whether the amount is being debited (commitment to a negative value) or credited (commitment to a positive value). This design allows auditors to efficiently verify the balance of each user. However, zkLedger has a significant limitation: it is designed to support anonymity for only a small number of users as the transaction size grows linearly in the number of users unlike DART where transaction size is of constant size. Moreover, zkLedger performs over a permissioned system and requires participants to have out-of-band communications.

Zether [8] and Anonymous Zether [19] are anonymous payment system designs that hide the user balance, transaction value, and sender and receiver identities. These systems are account-based, where each public key holder is associated with an ElGamal encryption of their balance under their own public key. To generate a transaction, the sender encrypts the transaction value under the receiver's public key and the negative transaction value under their own public key. Additionally, the sender encrypts zero under some randomly chosen (dummy) public keys. These dummy public keys, along with the sender's and receiver's public keys, form an anonymity set (a ring), concealing the identities of both the sender and the receiver. The sender then proves in zero knowledge that all encryptions are well-formed and that the transaction has been constructed correctly. The system restricts each sender to at most *one* transaction per epoch, where an epoch consists of k consecutive blocks. At the beginning of each epoch, the sender must query the blockchain to obtain their most updated state before generating their ZKP. This update is necessary because, at the end of each epoch, the blockchain rolls over all pending states to permanent ones. If other users include the sender's public key in their anonymity sets, the sender's state will be updated accordingly at the end of the epoch. Anonymous Zether makes

ZKPs of Zether more efficient. These works only support k-anonymity and have concurrency issues, with a transaction size of $\mathcal{O}(k)$.

PriFHEte [34], while achieving full anonymity in an account-based model with a constant transaction size of $\mathcal{O}(1)$, similarly suffers from concurrency issues (further discussion on PriFHEte follows). In contrast, in DART, no one updates another user's account state except the user themselves, allowing the sender to submit transactions at any time and avoiding such concurrency issues. Moreover, receiver affirmation, multiple assets, and asset-type-specific auditing are not supported in [8, 19, 34]. We note, however, that unlike DART, the nullifier set in Zether, Anonymous Zether, and PriFHEte does not grow, as it is reset to empty at the end of each epoch. PriDe CT [28] is a follow-up work on Anonymous Zether, inheriting some of its properties such as being account-based, achieving k-anonymity, and having a transaction size of $\mathcal{O}(k)$, while also introducing support for MPT.

Veksel [11] is an anonymous account-based payment system that shares similarities with Zerocash [49] in that it supports an arbitrarily large anonymity set. Payments in Veksel are facilitated by having the sender generate a coin commitment along with its encryption under the receiver's public key. During the collection process, a random spending tag (i.e., nullifier) is revealed. While the system ensures unlinkability for each transaction to its sender and receiver and obscures the transferred value, it does not hide the identities of the involved parties (e.g. balance updates are known), thus lack the anonymity.

PEReDi [47] and Platypus [56] are fully anonymous, account-based Central Bank Digital Currency (CBDC) payment systems. In Platypus, a *central* entity (the central bank) maintains the state of the system. PEReDi employs a distributed architecture, where cryptographically thresholdized maintainers manage the state of the system in a distributed (also referred to as *permissioned*) and asynchronous manner. In both schemes, the sender interacts with the receiver (hence, these are *interactive* systems[11]) to generate a transaction, proving in zero-knowledge that their account state transitions are correct. Each account state transition leaves a unique nullifier as a footprint, enabling the state maintainer(s) to detect double-spending. Additionally, for each state transition, users receive a signature from the state maintainer(s), confirming the validity of the operation. Payments in PEReDi's distributed setting do not require Byzantine Agreement or Byzantine Broadcast during the optimistic path of a transaction. This leads to efficient state transitions whenever transaction counterparties behave honestly. Maintainers independently sign each user's account state transition using a threshold blind and randomizable signature without communicating with one another. Users then collect signature shares from maintainers to consolidate a full signature on their updated account state. PEReDi introduces an approach for tracing potentially malicious users through the collaboration of a threshold number of maintainers. The revealed double-spending prevention unique tags also serve as a means to trace users. Multiple assets and asset-type-

[11] A limitation of interactive systems is that a sender cannot send funds if the receiver is not online and engaged in the transaction submission.

specific auditing are not supported in PEReDi and Platypus. In Platypus, the entire system operates under a centralized auditor, while in PEReDi, it operates under a cryptographically thresholdized set of auditors (the auditors are not asset-specific). These systems do not have concurrency issues and have a transaction size of $\mathcal{O}(1)$. Due to their interactive nature, they trivially support receiver affirmation by default.

PARScoin [48] is a fiat currency tokenization follow-up work on PEReDi that similarly has a fully anonymous, asynchronous, account-based and distributed model with no concurrency issues and transaction size of $\mathcal{O}(1)$. Note that fiat-backed stablecoins are a specific use case of our general solution. In the case of PARScoin, the real-world asset is specifically a fiat currency, and as a result of tokenization, the on-chain asset is referred to as a stablecoin—representing the tokenized form of the fiat currency[12]. PARScoin extends the interactive payment model of PEReDi to a non-interactive one. It supports privacy-preserving auditability using homomorphic encryptions ensuring that the stablecoin in circulation is backed by sufficient off-chain funds, which is important for price stability. In PARScoin, if the receiver is unable to claim the funds for any reason (e.g., unwillingness to affirm the transaction due to tax liabilities, loss of access to their wallet, etc.), the sender's funds are locked forever. In contrast, a DART sender is able to reclaim funds if the receiver does not affirm the transaction. Our reversibility solution incentivizes the receiver to affirm the transaction and update their account balance as soon as possible to guarantee the ownership transfer. This differs from PARScoin's setting, in which ownership is immediately transferred to the receiver as soon as the sender submits their transaction. In PARScoin, the receiver must update their account balance by claiming the funds sent by the sender to be able to spend them. However, the receiver can perform this action at any time. For example, consider a balance upper-bound requirement, which is a common regulatory requirement [47,48,56]. A potentially malicious receiver could delay claiming the incoming funds until after sending funds to others (reducing their balance), ensuring that their balance remains within the permissible range to accept new funds.

Recently, OCash [29] introduced a novel approach to anonymous payment systems in the account-based model, specifically designed to accommodate light clients. At a high level, the payer conducts payments by placing coins on the ledger, which can later be collected by the payees in a privacy-preserving manner. The system offers two levels of anonymity. In the weaker variant, the payer is able to observe when a payee claims the coin. However, the proposed design is limited by two assumptions that constrain its practical usability. First, it assumes the existence of a private channel between the payer and the payee. Second, the system requires an anonymizer service that maintains a private state and regularly posts messages on the ledger. The anonymity guarantees of OCash heavily depend on this second assumption, as the anonymizer functions

[12] Since DART supports multiple asset types, one of these asset types can be a stablecoin.

as the core of an Oblivious RAM-like structure [26], with the ledger acting as the database containing the committed or encrypted coins.

PriFHEte [34] is a novel approach to ensuring anonymity and confidentiality in account-based cryptocurrency transactions. The proposed protocol achieves full anonymity while having $\mathcal{O}(1)$ transaction size. This work leverages a property known as wrong-key decryption (WKE). The authors combine this property with fully homomorphic encryption (FHE) to design a payment system capable of facilitating "compact" transactions, which, with the assistance of chain validators, simultaneously update all accounts in the system for each transaction. The protocol employs a key-private public key encryption scheme with the WKE property to encrypt transactions, and an FHE scheme to encrypt user balances stored on-chain. When Alice sends an amount v to Bob, she creates a transaction that encrypts v using both her and Bob's WKE public keys. She also generates a corresponding NIZK proof to verify the transaction's validity. User balances are stored on-chain as ciphertexts, encrypted under their respective FHE public keys. Chain validators, equipped with the WKE public key, the FHE public key, and an encryption of the WKE secret key under the user's FHE public key, facilitate account state transitions. Validators must maintain this information for all users (accounts). After validating the sender's proof, validators execute the following steps for each account commitment recorded on-chain: Using the ciphertext of the WKE secret key encrypted under the user's FHE public key and the sender's transaction (which contains the encryption of v under both the sender's and receiver's WKE public keys), they generate encryptions with a plaintext value of zero for all non-counterparty users. For the actual sender and receiver, the plaintext value is v. Next, the FHE scheme is used to homomorphically deduct the amount v from the sender's balance and add it to the receiver's balance. The balances of all other users remain unchanged, as their old balances are updated by zero. The approach leverages the WKE property to avoid requiring the sender to generate zero-value encryptions for dummy public keys. Instead, validators utilize the WKE property to blindly generate these zero-value encryptions for all non-counterparty users. In addition to concurrency issues (discussed above), PriFHEte [34] needs homomorphic updates to *all* account commitments recorded on-chain for *each* transaction by blockchain validators (the validator complexity is $\mathcal{O}(n)$ for each transaction), raising concerns about its practicality in terms of both *transaction fees* and *computational efficiency*[13]. Biccer and Tschudin [5] demonstrated a double-spending attack on PriFHEte. In response, the authors proposed a mitigation technique involving two commitments to address this vulnerability.

[13] As the number of accounts in the system grows, the fees associated with each transaction would increase, since validators must update more on-chain entries, including the sender's. This dynamic seems counterintuitive, as network effects typically reduce user costs.

3 Formal modeling

We define $\mathcal{F}_{\mathsf{DART}}$ as an ideal functionality for anonymous and regulation-friendly tokenization, capturing the desired properties. A party P can take the roles of issuer, sender P^{s}, receiver P^{r}, and auditor A. The adversary is $\mathcal{A}$, and the session identifier as sid is selected by the environment $\mathcal{Z}$. $\mathcal{F}_{\mathsf{DART}}$ is parameterized by an upper bound on transaction value $\mathsf{v}_{\mathsf{max}}$. $\mathcal{A}$ can delay the delivery of messages or prevent a message from being delivered to the intended recipient. This models real-world scenarios where $\mathcal{A}$ has control over the communication channel. The ideal functionality $\mathcal{F}_{\mathsf{DART}}$ is composed of the following interfaces:

Address generation. A party P invokes $\mathcal{F}_{\mathsf{DART}}$, which in turn queries the adversary $\mathcal{A}$ for an address $\mathsf{addr}_{\mathsf{pk}}$. Upon receiving it from $\mathcal{A}$, $\mathcal{F}_{\mathsf{DART}}$ verifies whether it is fresh. If so, it records the party identifier P along with the address and outputs $\mathsf{addr}_{\mathsf{pk}}$ to P.

Asset issuance. A party P invokes this interface to issue a new asset by providing the asset type (identifier) $\mathsf{a.t}$ and the associated auditor A assigned to $\mathsf{a.t}$. $\mathcal{F}_{\mathsf{DART}}$ first verifies whether P and A have already generated their addresses and checks whether $\mathsf{a.t}$ is fresh, as it must be unique. $\mathcal{F}_{\mathsf{DART}}$ does not enforce uniqueness for the auditor A assigned to $\mathsf{a.t}$, allowing a single auditor A to serve as the auditor for more than one $\mathsf{a.t}$. $\mathcal{F}_{\mathsf{DART}}$ registers the issuer along with $\mathsf{a.t}$ in the list $\mathtt{IA}$ and registers the associated auditor A along with $\mathsf{a.t}$ in the list $\mathtt{AA}$. $\mathcal{F}_{\mathsf{DART}}$ leaks $(\mathsf{P}, \mathsf{a.t}, \mathsf{A})$ to $\mathcal{A}$, as there is no privacy preservation at the issuance.

Account registration. By invoking this interface, the party P registers an asset-specific account. $\mathcal{F}_{\mathsf{DART}}$ verifies whether P has generated an address and checks whether an account with the same $\mathsf{a.t}$ has already been registered for P via checking $\mathbb{R}(\mathsf{P}, \mathsf{a.t})$, which is initially set to $\perp$. $\mathcal{F}_{\mathsf{DART}}$ also checks whether $\mathsf{a.t}$ is a valid asset using $\mathtt{IA}$. $\mathcal{F}_{\mathsf{DART}}$ then updates $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) = (\mathsf{f.b}, \mathsf{p.b}) = (0, 0)$, where the initial finalized (spendable) balance $\mathsf{f.b}$ and pending (non-spendable) balance $\mathsf{p.b}$ are set to zero.

Increase asset supply. The issuer P of an asset $\mathsf{a.t}$ invokes this interface to increase the supply of $\mathsf{a.t}$ by a value v. $\mathcal{F}_{\mathsf{DART}}$ checks whether P has initialized their account and verifies that they are indeed the associated issuer of $\mathsf{a.t}$. $\mathcal{F}_{\mathsf{DART}}$ records $(c, \mathsf{P}, \mathsf{a.t}, v)$ to prevent replay attacks by verifying c, which is initially set to 0. $\mathcal{F}_{\mathsf{DART}}$ then waits for confirmation from $\mathcal{A}$ to finalize the supply increase by increasing $\mathsf{f.b}$ of the issuer's account by v and setting c to 1.

Functionality $\mathcal{F}_{\mathsf{DART}}$ part I

The functionality is parameterized by $\mathsf{v}_{\mathsf{max}}$, and the lists $\mathtt{IA}$ and $\mathtt{AA}$ are initialized as empty.

Address generation.

- Upon receiving $(\mathtt{Gen.Addr}, \mathsf{sid})$ from P, send $(\mathtt{Gen.Addr}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{A}$.

- Upon receiving $(\texttt{Gen.Addr.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{addr}_{\mathsf{pk}})$ from $\mathcal{A}$, ignore if $(\cdot, \mathsf{addr}_{\mathsf{pk}})$ is already recorded. Else, record $(\mathsf{P}, \mathsf{addr}_{\mathsf{pk}})$. Output $(\texttt{Addr.Gened}, \mathsf{sid}, \mathsf{addr}_{\mathsf{pk}})$ to P.

Asset issuance.

- Upon receiving $(\texttt{Issue}, \mathsf{sid}, \mathsf{a.t}, \mathsf{A})$ from P, ignore if $(\mathsf{P}, \cdot)$ or $(\mathsf{A}, \cdot)$ is not recorded or $(\cdot, \mathsf{a.t}) \in \texttt{IA}$. Else, send $(\texttt{Issue}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{A})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Issue.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{A})$ from $\mathcal{A}$, set $\texttt{IA} \leftarrow \texttt{IA} \cup \{(\mathsf{P}, \mathsf{a.t})\}$ and $\texttt{AA} \leftarrow \texttt{AA} \cup \{(\mathsf{A}, \mathsf{a.t})\}$. Output $(\texttt{Issued}, \mathsf{sid}, \mathsf{a.t})$ to P via public-delayed output.

Account registration.

- Upon receiving $(\texttt{Register}, \mathsf{sid}, \mathsf{a.t})$ from P, ignore if $(\mathsf{P}, \cdot)$ is not recorded, or $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) \neq \bot$, or $(\cdot, \mathsf{a.t}) \notin \texttt{IA}$. Else, send $(\texttt{Register}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Register.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t})$ from $\mathcal{A}$, if $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) = \bot$, set $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) \leftarrow (0, 0)$ and send $(\texttt{Registered}, \mathsf{sid}, \mathsf{a.t})$ to P via public-delayed output.

Increase asset supply.

- Upon receiving $(\texttt{Increase}, \mathsf{sid}, \mathsf{a.t}, v)$ from P, ignore if $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) = \bot$, or $(\mathsf{P}, \mathsf{a.t}) \notin \texttt{IA}$, or $v \leq 0$. Else, record $(c, \mathsf{P}, \mathsf{a.t}, v)$ for $c = 0$. Send $(\texttt{Increase}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, v)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Increase.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, v)$ from $\mathcal{A}$, ignore if $(0, \mathsf{P}, \mathsf{a.t}, v)$ is not recorded. Else, retrieve $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) = (\mathsf{f.b}, \mathsf{p.b})$. Set $\mathsf{f.b} \leftarrow \mathsf{f.b} + v$ and $c \leftarrow 1$. Output $(\texttt{Increased}, \mathsf{sid}, \mathsf{a.t}, v)$ to user P via public-delayed output.

Sender transaction. A sender P^{s} invokes this interface by providing the receiver's identifier P^{r} and specifying the value v as the transfer amount of asset type $\mathsf{a.t}$. $\mathcal{F}_{\mathsf{DART}}$ then verifies several conditions, such as whether the sender's finalized balance contains sufficient funds. In cases where the receiver P^{r} or $\mathsf{a.t}$'s auditor A is malicious, $\mathcal{F}_{\mathsf{DART}}$ leaks $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$ to $\mathcal{A}$. Otherwise, $\mathcal{A}$ only observes a random identifier β, while $\mathcal{F}_{\mathsf{DART}}$ internally maintains a mapping list $L(\cdot)$ to associate β with the transaction details. Upon receiving confirmation from $\mathcal{A}$, $\mathcal{F}_{\mathsf{DART}}$ decreases the sender's finalized balance and increases their pending balance. Note that a *unique* identifier TNX is submitted by $\mathcal{A}$ to $\mathcal{F}_{\mathsf{DART}}$, which uses it to uniquely identify each transaction (rejecting if TNX is not unique). α is an associated flag to TNX representing the status of the transaction. It is necessary to check the balance of the sender again, ensuring $\mathsf{f.b}^{\mathsf{s}} < v$ before the balance update, as the sender may have already initiated other transactions. $L(\beta)$ is set to $\bot$ after updating the balance to prevent replay attacks.

Receiver transaction. A receiver P^{r} invokes this interface by providing the unique identifier of a transaction TNX. $\mathcal{F}_{\mathsf{DART}}$ verifies whether a payment associated with TNX exists, belongs to P^{r}, and has not yet been affirmed (i.e., $\alpha =$

0). No information is leaked to the adversary, except TNX. In case $\mathsf{P^s}$ is malicious $\mathcal{F}_{\mathsf{DART}}$ leaks $(\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t}, \mathsf{TNX})$ to $\mathcal{A}$. Upon receiving $\mathcal{A}$'s confirmation, $\mathcal{F}_{\mathsf{DART}}$ outputs $(\mathsf{P^s}, v, \mathsf{a.t}, \mathsf{TNX})$ to $\mathsf{P^r}$. If $\mathsf{P^r}$ affirms the transaction by sending back $(\mathtt{Affirm}, \mathsf{sid}, \mathsf{TNX})$, $\mathcal{F}_{\mathsf{DART}}$ increases $\mathsf{P^r}$ finalized balance $\mathsf{f.b^r} + v$, and decreases the associated pending balance for $\mathsf{a.t}$ of $\mathsf{P^s}$. $\mathcal{F}_{\mathsf{DART}}$ ensures the funds in this transaction cannot be claimed again or reversed by setting $\alpha \leftarrow 1$.

Reversion. Using this interface, a sender $\mathsf{P^s}$ provides as input a unique identifier TNX of a transaction. $\mathcal{F}_{\mathsf{DART}}$ then verifies whether $\mathsf{P^s}$ is indeed the sender of TNX and checks the status α associated with TNX to determine if the transaction has already been affirmed by the receiver. Similar to receiving, $\mathcal{F}_{\mathsf{DART}}$ leaks only TNX to $\mathcal{A}$, and upon confirmation, reverses the transaction by updating the sender's finalized and pending balances. $\mathcal{F}_{\mathsf{DART}}$ ensures that the receiver cannot affirm this transaction in the future by setting α to 2 (recall that affirmation requires $\alpha = 0$) and prevents the sender from reversing the transaction again (reversion also requires $\alpha = 0$).

Functionality $\mathcal{F}_{\mathsf{DART}}$ part II

Sender transaction.

- Upon receiving $(\mathtt{Send}, \mathsf{sid}, \mathsf{P^r}, v, \mathsf{a.t})$ from $\mathsf{P^s}$, ignore if $\mathbb{R}(\mathsf{P^s}, \mathsf{a.t}) = \bot$, or $v \leq 0$, or $v > \mathsf{v_{max}}$. Else, retrieve $\mathbb{R}(\mathsf{P^s}, \mathsf{a.t}) = (\mathsf{f.b^s}, \mathsf{p.b^s})$ and ignore if $\mathsf{f.b^s} < v$. Else, sample a new β and set $L(\beta) \leftarrow (\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t})$. Retrieve $(\mathsf{A}, \mathsf{a.t})$ from $\mathtt{AA}$. If A or $\mathsf{P^r}$ is malicious, overwrite $\beta \leftarrow (\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t})$. Send $(\mathtt{Send}, \mathsf{sid}, \beta)$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Send.Ok}, \mathsf{sid}, \beta, \mathsf{TNX})$ from $\mathcal{A}$, ignore if $L(\beta) = \bot$, or $(\cdot, \cdot, \cdot, \cdot, \cdot, \mathsf{TNX})$ is already recorded. Else, retrieve $L(\beta) = (\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t})$ and then $\mathbb{R}(\mathsf{P^s}, \mathsf{a.t}) = (\mathsf{f.b^s}, \mathsf{p.b^s})$. Ignore if $\mathsf{f.b^s} < v$. Else, set $\mathbb{R}(\mathsf{P^s}, \mathsf{a.t}) \leftarrow (\mathsf{f.b^s} - v, \mathsf{p.b^s} + v)$, and $L(\beta) \leftarrow \bot$. Record $(\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t}, \alpha, \mathsf{TNX})$, where $\alpha = 0$. Output $(\mathtt{Sent}, \mathsf{sid}, , \mathsf{P^r}, v, \mathsf{a.t}, \mathsf{TNX})$ to $\mathsf{P^s}$ via private-delayed output.

Receiver transaction.

- Upon receiving $(\mathtt{Receive}, \mathsf{sid}, \mathsf{TNX})$ from $\mathsf{P^r}$, ignore if $(\cdot, \mathsf{P^r}, \cdot, \mathsf{a.t}, 0, \mathsf{TNX})$ for some $\mathsf{a.t}$ is not recorded. Retrieve $\mathbb{R}(\mathsf{P^r}, \mathsf{a.t})$ using $\mathsf{a.t}$, and ignore if it equals $\bot$. Else, retrieve $(\mathsf{A}, \mathsf{a.t})$ from $\mathtt{AA}$ and retrieve $(\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t}, \alpha, \mathsf{TNX})$. If $\mathsf{P^s}$ is malicious send $(\mathtt{Receive}, \mathsf{sid}, \mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t}, \mathsf{TNX})$ to $\mathcal{A}$. Else, send $(\mathtt{Receive}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Receive.Ok}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{A}$, retrieve $(\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t}, \alpha, \mathsf{TNX})$. Ignore if $\alpha \neq 0$. Output $(\mathtt{Receiving}, \mathsf{sid}, \mathsf{P^s}, v, \mathsf{a.t}, \mathsf{TNX})$ to $\mathsf{P^r}$ via private-delayed output.
- Upon receiving $(\mathtt{Affirm}, \mathsf{sid}, \mathsf{TNX})$ from $\mathsf{P^r}$, ignore if $(\mathtt{Received}, \mathsf{sid}, \mathsf{P^s}, v, \mathsf{a.t}, \mathsf{TNX})$ has not been output to $\mathsf{P^r}$ yet or $(\cdot, \cdot, \cdot, \cdot, 1, \mathsf{TNX})$ is recorded. Otherwise, send $(\mathtt{Affirmed}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{A}$. Upon receiving $(\mathtt{Affirmed.Ok}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{A}$, retrieve $\mathbb{R}(\mathsf{P^r}, \mathsf{a.t}) = (\mathsf{f.b^r}, \mathsf{p.b^r})$, and

$\mathbb{R}(\mathsf{P^s}, \mathsf{a.t}) = (\mathsf{f.b^s}, \mathsf{p.b^s})$. Update $\mathsf{f.b^r} \leftarrow \mathsf{f.b^r} + v$, $\mathsf{p.b^s} \leftarrow \mathsf{p.b^s} - v$, and $\alpha \leftarrow 1$. Output $(\texttt{Received}, \mathsf{sid}, \mathsf{TNX})$ to $\mathsf{P^r}$ via public-delayed output.

Reversion.

- Upon receiving $(\texttt{Reverse}, \mathsf{sid}, \mathsf{TNX})$ from $\mathsf{P^s}$, ignore if $(\mathsf{P^s}, \cdot, \cdot, \mathsf{a.t}, 0, \mathsf{TNX})$ for some $\mathsf{a.t}$ is not recorded. Retrieve $\mathbb{R}(\mathsf{P^s}, \mathsf{a.t})$ using $\mathsf{a.t}$, and ignore if it equals $\perp$. Else, send $(\texttt{Reverse}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Reverse.Ok}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{A}$, retrieve $(\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t}, \alpha, \mathsf{TNX})$. Ignore if $\alpha \neq 0$. Else, retrieve $\mathbb{R}(\mathsf{P^s}, \mathsf{a.t}) = (\mathsf{f.b^s}, \mathsf{p.b^s})$. Set $\mathsf{f.b^s} \leftarrow \mathsf{f.b^s} + v$, $\mathsf{p.b^s} \leftarrow \mathsf{p.b^s} - v$, and $\alpha \leftarrow 2$. Output $(\texttt{Reversed}, \mathsf{sid}, \mathsf{TNX})$ to $\mathsf{P^s}$.

Proof of balance. The user provides $\mathsf{a.t}$ to $\mathcal{F}_{\mathsf{DART}}$, which in turn outputs a fresh random identifier γ to $\mathcal{A}$. If $\mathsf{a.t}$ has the auditor A who is corrupted, then $\mathcal{F}_{\mathsf{DART}}$ overwrites γ with $(\mathsf{P}, \mathsf{a.t})$. Upon receiving $\mathcal{A}$'s message $\mathcal{F}_{\mathsf{DART}}$ outputs $(\mathsf{P}, \mathsf{a.t}, \mathsf{f.b}, \mathsf{p.b})$ to the auditor.

Auditor operation. The auditor A invokes this interface by providing the unique identifier of a transaction TNX. $\mathcal{F}_{\mathsf{DART}}$ checks whether a transaction exists that is recorded with the specified TNX and verifies whether $\mathsf{a.t}$ associated with TNX has the caller as its auditor. If the auditor is indeed the associated auditor, $\mathcal{F}_{\mathsf{DART}}$, upon $\mathcal{A}$'s confirmation, outputs the transaction metadata to A.

Functionality $\mathcal{F}_{\mathsf{DART}}$ part III

Proof of balance.

- Upon receiving $(\texttt{Balance}, \mathsf{sid}, \mathsf{a.t})$ from P, ignore if $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) = \perp$. Else, given $\mathsf{a.t}$, retrieve $(\mathsf{A}, \mathsf{a.t})$ from $\texttt{AA}$. Generate a new γ, and set $L(\gamma) \leftarrow (\mathsf{P}, \mathsf{a.t}, \mathsf{A})$. If A is malicious, overwrite $\gamma \leftarrow (\mathsf{P}, \mathsf{a.t})$. Send $(\texttt{Balance}, \mathsf{sid}, \gamma)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Balance.Ok}, \mathsf{sid}, \gamma)$ from $\mathcal{A}$, ignore if $L(\gamma) = \perp$. Else, retrieve $L(\gamma) = (\mathsf{P}, \mathsf{a.t}, \mathsf{A})$. Retrieve $\mathbb{R}(\mathsf{P}, \mathsf{a.t}) = (\mathsf{f.b}, \mathsf{p.b})$. Output $(\texttt{Balance}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{f.b}, \mathsf{p.b})$ to A via private-delayed output and set $L(\gamma) \leftarrow \perp$.

Auditor operation.

- Upon receiving $(\texttt{Audit}, \mathsf{sid}, \mathsf{TNX})$ from an auditor A, ignore if $(\cdot, \cdot, \cdot, \mathsf{a.t}, \cdot, \mathsf{TNX})$ for some $\mathsf{a.t}$ is not recorded. Else, ignore if $(\mathsf{A}, \mathsf{a.t}) \notin \texttt{AA}$. Else, sample a new η, and set $L(\eta) \leftarrow (\mathsf{A}, \mathsf{TNX})$. Send $(\texttt{Audit}, \mathsf{sid}, \eta)$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Audit.Ok}, \mathsf{sid}, \eta)$ from $\mathcal{A}$, ignore if $L(\eta) = \perp$. Else, retrieve $L(\eta) = (\mathsf{A}, \mathsf{TNX})$, and then retrieve $(\mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t}, \cdot, \mathsf{TNX})$. Output $(\texttt{Audited}, \mathsf{sid}, \mathsf{P^s}, \mathsf{P^r}, v, \mathsf{a.t})$ to A via private-delayed output, and set $L(\eta) \leftarrow \perp$.

4 Our construction DART

As previously discussed, users maintain accounts where old account $\mathsf{acct}_{\mathsf{old}}$ transitions into a new account $\mathsf{acct}_{\mathsf{new}}$ as a result of a transaction. To achieve unlinkability, users do not reveal their $\mathsf{acct}_{\mathsf{old}}$ and only reveal their $\mathsf{acct}_{\mathsf{new}}$, and to achieve confidentiality the *commitment* to the new account $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}$ is revealed. Users prove in zero-knowledge that $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}$ is a valid member of the blockchain's state $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}} \in \mathsf{state}$ and that it has not been used. This is done by revealing the nullifier $\mathsf{nul} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho)$, which is a pseudorandom function (PRF) (Definition 11) output. It is proved that nul is well-formed with respect to $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}$ for which the membership proof is provided. The ledger then checks if nul has already appeared in the ledger's state. Moreover, the user proves that $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}$, which is revealed, is consistent with $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}$, according to the *type* of the transaction. For instance, in the case transaction type is $\mathtt{Send}$, $\mathsf{tnx}_{\mathtt{Send}}$, the sender proves that $\mathsf{f.b}$ in $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}$ has been correctly reduced by the transaction value v, i.e., $\mathsf{f.b}^{\mathsf{s}}_{\mathsf{new}} = \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} - v$ holds. Once the transaction tnx is validated, the ledger records $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}$ nullifier $\mathsf{nul}_{\mathsf{old}}$, and adds $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}$ to state.

In our construction Π_{DART}, we employ the following ideal functionalities: the key generation $\mathcal{F}_{\mathsf{KeyReg}}$ (Appendix B.1), the non-interactive zero-knowledge $\mathcal{F}_{\mathsf{NIZK}}$ (Appendix B.2), the ledger $\mathcal{F}_{\mathsf{Ledger}}$ (Appendix B.3), and the communication channel $\mathcal{F}_{\mathsf{ch}}$ (Appendix B.4). Additionally, Π_{DART} incorporates several cryptographic primitives, specifically a public-key encryption scheme (Definition 1), a commitment scheme (Definition 6), a pseudorandom function (PRF) (Definition 11), and an accumulator scheme (Definition 14). See Appendix C for formal definitions and security properties. See Section 6 for their instantiations.

User account acct is defined as a tuple $\mathsf{acct} := (\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}, \rho, s)$ containing an account secret key $\mathsf{sk}_{\mathsf{acct}}$, finalized balance $\mathsf{f.b}$ that is spendable, pending balance $\mathsf{p.b}$ that is not spendable, an asset type $\mathsf{a.t}$, and randomness values ρ and s, respectively for deriving nullifier nul and generating account commitment $\mathsf{acct}^{\mathsf{cm}}$. We denote the NIZK statement, witness, relation, and proof by x, w, $\mathcal{R}$, and π, respectively. Similar to $\mathcal{F}_{\mathsf{DART}}$, Π_{DART} is parameterized by $\mathsf{v}_{\mathsf{max}}$, ensuring that both $\mathsf{f.b}$ and $\mathsf{p.b}$ will not overflow. $\mathcal{F}_{\mathsf{Ledger}}$ maintains the state of the ledger $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$. state parses into all transactions submitted to the chain state', the most recent accumulator value $\mathsf{v}_{\mathsf{ACCU}}$, and *most recent* state of three lists $(\mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}})$, where $\mathbb{L}_{\mathsf{NUL}}$ is an append-only list of account nullifiers updated by consensus and $\mathbb{L}_{\mathsf{IAA}}$ and $\mathbb{L}_{\mathsf{KA}}$ will be described in the following sections. $\mathcal{F}_{\mathsf{Ledger}}$ is parameterized by two functions: (i) (0 or 1) $\leftarrow$ VALIDATE($\mathsf{state}, \mathsf{tnx}$), and (ii) $\mathsf{state}^{\mathsf{new}} \leftarrow$ UPDATE($\mathsf{state}^{\mathsf{old}}, \mathsf{tnx}$), both taking as input blockchain state and a transaction tnx. Each of these functions includes several sub-algorithms tailored to different types of transactions as we will see later. Each consensus member—whether part of a permissioned or permissionless blockchain—reads the blockchain state state and verify the transaction by executing VALIDATE($\mathsf{state}, \mathsf{tnx}$). If the transaction is verified, state is updated accordingly by executing UPDATE($\mathsf{state}, \mathsf{tnx}$).

4.1 Algorithms

Figure 1 illustrates the sender's transaction status $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} \in \{0, 1, 2\}$. Each transition between these values is triggered by a specific transaction. By submitting $\mathsf{tnx}_{\mathtt{Send}}$ to the ledger, the sender's finalized balance decreases by the transaction value $\mathsf{f.b}^{\mathsf{s}}_{\mathsf{new}} = \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} - v$ and pending balance $\mathsf{p.b}^{\mathsf{s}}$ increases by the same value $\mathsf{p.b}^{\mathsf{s}}_{\mathsf{new}} = \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} + v$. Without handling $\mathsf{p.b}^{\mathsf{s}}$, a malicious sender could circumvent PoB as discussed in Section 1. As Figure 1 illustrates, from $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 0$ two

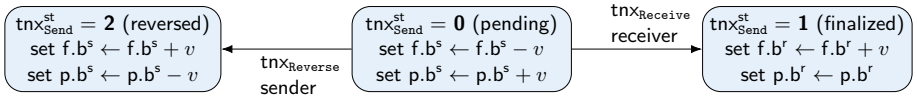

Fig. 1: Transaction status $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}}$.

possible transitions can occur. A transition to $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$ when the receiver affirms by submitting $\mathsf{tnx}_{\mathtt{Receive}}$. Or, a transition to $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 2$ when the sender reverses by submitting $\mathsf{tnx}_{\mathtt{Reverse}}$. A transition from $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$ to $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 2$ is impossible; once the receiver affirms, the sender can no longer reverse. Similarly, a transition from $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 2$ to $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$ is not possible; once the sender reverses, the receiver cannot affirm. Whenever the receiver submits $\mathsf{tnx}_{\mathtt{Receive}}$, their finalized balance increases $\mathsf{f.b}^{\mathsf{r}}_{\mathsf{new}} = \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}} + v$. Once $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$ the transfer of fund ownership from the sender's account to the receiver's account is completed. When the sender submits $\mathsf{tnx}_{\mathtt{Reverse}}$, they perform the exact opposite of what they did in the initial transaction $\mathsf{tnx}_{\mathtt{Send}}$. After key generation, each party ignores environment $\mathcal{Z}$'s instruction if $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ is not recorded but for simplicity we avoid explicitly addressing it.

4.1.1 Address generation. Each user calls $\mathcal{F}_{\mathsf{KeyReg}}$ to receive two pairs of cryptographic keys: one pair $(\mathsf{pk}_{\mathsf{en}}, \mathsf{sk}_{\mathsf{en}})$ for PKE and another pair $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}})$ for their account. The algorithm is provided in Figure 2.

▷ Upon receiving $(\mathtt{Gen.Addr}, \mathsf{sid})$ from $\mathcal{Z}$ call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Gen.Key}, \mathsf{sid})$.
▷ Upon receiving $(\mathtt{Gen.Key}, \mathsf{sid}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ from $\mathcal{F}_{\mathsf{KeyReg}}$, record $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$.
▷ Output $(\mathtt{Addr.Gened}, \mathsf{sid}, \mathsf{addr}_{\mathsf{pk}})$ to $\mathcal{Z}$.

Fig. 2: Address generation

4.1.2 Asset issuance. There is a public list called the Issuer-Asset-Auditor list, denoted by $\mathbb{L}_{\mathsf{IAA}}$, which is maintained by consensus (on-chain) and initially set to empty. When a party P issues an asset $\mathsf{a.t}$, $\mathbb{L}_{\mathsf{IAA}}$ is updated accordingly.

$\mathbb{L}_{\mathsf{IAA}}$ contains tuples of the form $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$, where $\mathsf{pk}_{\mathsf{acct}}$ is issuer's account public key and $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$ is the auditor's public key encryption[14]. In our NIZK relations, we may need a concise representation of $\mathbb{L}_{\mathsf{IAA}}$, which only contains pairs of $(\mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$. We will therefore abuse the notation when describing proof relations so that for an index i, we write $(\mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}) = \mathbb{L}_{\mathsf{IAA}}[i]$ to indicate that $(\mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$ is the i-th entry in $\mathbb{L}_{\mathsf{IAA}}$, providing the association of $\mathsf{a.t}$ and $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$. In Section 5.3, we discuss how to address situations where it becomes necessary to revoke an auditor. The algorithm is provided in Figure 3.
The NIZK witness, statement, and relation are defined as follows.

- $w_{\mathtt{Issue}} = \mathsf{sk}_{\mathsf{acct}}$
- $x_{\mathtt{Issue}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$
- The relation $\mathcal{R}_{\mathtt{Issue}}(x_{\mathtt{Issue}}, w_{\mathtt{Issue}})$ is defined as:

$$\mathcal{R}_{\mathtt{Issue}} = \Big\{ (\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}}) \in \mathsf{Acc.KeyGen}(1^{\lambda}) \Big\}.$$

$\mathsf{tnx}_{\mathtt{Issue}}$ Generation

▷ Upon receiving $(\mathtt{Issue}, \mathsf{sid}, \mathsf{a.t}, \mathsf{A})$ from $\mathcal{Z}$: ignore if $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ is not recorded.
▷ Else, call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$. Upon receiving $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ Ignore if $(\cdot, \mathsf{a.t}, \cdot) \in \mathbb{L}_{\mathsf{IAA}}$.
▷ Else, parse $\mathsf{addr}_{\mathsf{pk}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{pk}_{\mathsf{en}})$, and $\mathsf{addr}_{\mathsf{sk}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{en}})$.
▷ Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{A})$ and upon receiving $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{A}, \mathsf{addr}^{\mathsf{A}}_{\mathsf{pk}})$ retrieve $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$ from $\mathsf{addr}^{\mathsf{A}}_{\mathsf{pk}}$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, x_{\mathtt{Issue}}, w_{\mathtt{Issue}})$ and receive $(\mathtt{Proof}, \mathsf{sid}, \pi_{\mathtt{Issue}})$.
▷ Set $\mathsf{tnx}_{\mathtt{Issue}} \leftarrow (x_{\mathtt{Issue}}, \pi_{\mathtt{Issue}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Append}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Issue}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$; upon receiving back $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$, if $\mathsf{tnx}_{\mathtt{Issue}} \in \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$ output $(\mathtt{Issued}, \mathsf{sid}, \mathsf{a.t})$ to $\mathcal{Z}$.

$\mathsf{tnx}_{\mathtt{Issue}}$ Verification

▷ Execute $\textsc{Validate}_{\mathtt{Issue}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Issue}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Issue}} = (x_{\mathtt{Issue}}, \pi_{\mathtt{Issue}})$, and $x_{\mathtt{Issue}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$.
- ignore if any of the following conditions hold:
  - $(\cdot, \mathsf{a.t}, \cdot) \in \mathbb{L}_{\mathsf{IAA}}$
  - upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x_{\mathtt{Issue}}, \pi_{\mathtt{Issue}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.
- else, call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Rtrv.id}, \mathsf{sid}, \mathsf{pk})$ for $\mathsf{pk} = \mathsf{pk}_{\mathsf{acct}}$, and $\mathsf{pk} = \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$.
- upon receiving $(\mathtt{Rtrved.id}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{F}_{\mathsf{KeyReg}}$ for $\mathsf{P} = \mathsf{P}^*$ and $\mathsf{P} = \mathsf{A}$ output 1.

[14] We assume a single auditor A in our formal analysis. We believe our approach can be straightforwardly extended to accommodate a group of auditors.

▷ Execute $\textsc{Update}_{\texttt{Issue}}(\mathsf{state}, \mathsf{tnx}_{\texttt{Issue}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\texttt{Issue}} = (x_{\texttt{Issue}}, \pi_{\texttt{Issue}})$, and $x_{\texttt{Issue}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$.
- set $\mathbb{L}_{\mathsf{IAA}} \leftarrow \mathbb{L}_{\mathsf{IAA}} \cup \{(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})\}$.
- set $\mathsf{state} \leftarrow \mathsf{state} \cup \{(\cdot, \mathbb{L}_{\mathsf{IAA}}, \cdot, \cdot, \cdot)\}$.
- output state.

Fig. 3: Asset issuance transaction $\mathsf{tnx}_{\texttt{Issue}}$

4.1.3 Account registration. There is a public list called the Key-Asset list, denoted by $\mathbb{L}_{\mathsf{KA}}$, which is maintained by consensus. The list is initially set to empty and is updated when a party P registers an asset-specific account acct. The $\mathbb{L}_{\mathsf{KA}}$ list stores tuples of the form $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t})$, where each $\mathsf{pk}_{\mathsf{acct}}$ can register one account for a given $\mathsf{a.t}$. P proves that account commitment $\mathsf{acct}^{\mathsf{cm}}$ is well-formed, and that they possess knowledge of the associated secret key of $\mathsf{pk}_{\mathsf{acct}}$. The algorithm is provided in Figure 4.
The NIZK witness, statement, and relation are defined as follows.

- $w_{\texttt{Register}} = (\mathsf{sk}_{\mathsf{acct}}, \rho, s)$
- $x_{\texttt{Register}} = (\mathsf{acct}^{\mathsf{cm}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t})$
- The relation $\mathcal{R}_{\texttt{Register}}(x_{\texttt{Register}}, w_{\texttt{Register}})$ is defined as:

$$\mathcal{R}_{\texttt{Register}} = \Big\{ \mathsf{acct}^{\mathsf{cm}} = \mathsf{Com}(\mathsf{sk}_{\mathsf{acct}}, 0, 0, \mathsf{a.t}, \rho; s) \\ \wedge (\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}}) \in \mathsf{Acc.KeyGen}(1^{\lambda}) \Big\}.$$

$\mathsf{tnx}_{\texttt{Register}}$ Generation

▷ Upon receiving $(\texttt{Register}, \mathsf{sid}, \mathsf{a.t})$ from $\mathcal{Z}$, ignore if $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ is not recorded.
▷ Else, parse $\mathsf{addr}_{\mathsf{pk}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{pk}_{\mathsf{en}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathsf{Read}, \mathsf{sid})$, and upon receiving back $(\mathsf{Read}, \mathsf{sid}, \mathsf{state})$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ Ignore if any of the following conditions hold:
- $(\cdot, \mathsf{a.t}, \cdot) \notin \mathbb{L}_{\mathsf{IAA}}$
- $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}) \in \mathbb{L}_{\mathsf{KA}}$

▷ Else, sample $\rho \xleftarrow{\$} \mathbb{Z}_q$, and $s \xleftarrow{\$} \mathbb{Z}_q$.
▷ Parse $\mathsf{addr}_{\mathsf{sk}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{en}})$.
▷ Set $\mathsf{acct} \leftarrow (\mathsf{sk}_{\mathsf{acct}}, 0, 0, \mathsf{a.t}, \rho, s)$.
▷ Compute $\mathsf{acct}^{\mathsf{cm}} \leftarrow \mathsf{Com}(\mathsf{sk}_{\mathsf{acct}}, 0, 0, \mathsf{a.t}, \rho; s)$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Prove}, \mathsf{sid}, x_{\texttt{Register}}, w_{\texttt{Register}})$ and receive $(\texttt{Proof}, \mathsf{sid}, \pi_{\texttt{Register}})$.
▷ Set $\mathsf{tnx}_{\texttt{Register}} \leftarrow (x_{\texttt{Register}}, \pi_{\texttt{Register}})$.

▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Append}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Register}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$, and upon receiving back $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$, if $\mathsf{tnx}_{\mathtt{Register}} \in \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$, record $(\mathsf{acct}^{\mathsf{cm}}, \mathsf{acct})$.
▷ Output $(\mathtt{Registered}, \mathsf{sid}, \mathsf{a.t})$ to $\mathcal{Z}$.

$\mathsf{tnx}_{\mathtt{Register}}$ Verification

▷ Execute $\textsc{Validate}_{\mathtt{Register}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Register}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Register}} = (x_{\mathtt{Register}}, \pi_{\mathtt{Register}})$, $x_{\mathtt{Register}} = (\mathsf{acct}^{\mathsf{cm}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t})$.
- ignore if any of the following conditions hold:
  - $(\cdot, \mathsf{a.t}, \cdot) \notin \mathbb{L}_{\mathsf{IAA}}$
  - $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}) \in \mathbb{L}_{\mathsf{KA}}$
  - upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x_{\mathtt{Register}}, \pi_{\mathtt{Register}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.
- else, call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Rtrv.id}, \mathsf{sid}, \mathsf{pk}_{\mathsf{acct}})$ and upon receiving back $(\mathtt{Rtrved.id}, \mathsf{sid}, \mathsf{P})$ output 1.

▷ Execute $\textsc{Update}_{\mathtt{Register}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Register}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Register}} = (x_{\mathtt{Register}}, \pi_{\mathtt{Register}})$, $x_{\mathtt{Register}} = (\mathsf{acct}^{\mathsf{cm}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t})$.
- call $\mathsf{v}_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm}})$.
- set $\mathbb{L}_{\mathsf{KA}} \leftarrow \mathbb{L}_{\mathsf{KA}} \cup \{(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t})\}$.
- set $\mathsf{state} \leftarrow \mathsf{state} \cup \{(\cdot, \cdot, \mathbb{L}_{\mathsf{KA}}, \cdot, \mathsf{v}_{\mathsf{ACCU}})\}$.
- output state.

Fig. 4: Account registration transaction $\mathsf{tnx}_{\mathtt{Register}}$

4.1.4 Increase asset supply. The asset issuer can increase the supply of the asset they have previously issued. The ledger verifies whether the issuer is the legitimate issuer of the specified asset based on $\mathbb{L}_{\mathsf{IAA}}$. The issuer's finalized balance increases $\mathsf{f.b}_{\mathsf{new}} = \mathsf{f.b}_{\mathsf{old}} + v$. Note that in this algorithm, there are no privacy-preserving operations, and $\mathsf{pk}_{\mathsf{acct}}$ is part of the public values. However, $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}$ is kept private, as revealing it would link the issuer's previous, potentially anonymous transaction (e.g., a send transaction) to this one, thereby compromising the privacy of those transactions. The issuer reveals their new account commitment $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}$ along with the nullifier $\mathsf{nul}_{\mathsf{old}}$ of $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}$ for which accumulator membership proof is computed. The issuer proves the knowledge of $\mathsf{sk}_{\mathsf{acct}}$ and discloses $\mathsf{a.t}$ and v. Furthermore, the well-formedness of $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}$ is also proved. The algorithm is provided in Figure 5.
The NIZK witness, statement, and relation are defined as follows.

- $w_{\mathtt{Increase}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}_{\mathsf{old}}, \mathsf{p.b}_{\mathsf{old}}, \rho_{\mathsf{old}}, s_{\mathsf{old}}, \rho_{\mathsf{new}}, s_{\mathsf{new}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}, \pi_{\mathsf{ACCU}})$
- $x_{\mathtt{Increase}} = (\mathsf{nul}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, v)$

- The relation $\mathcal{R}_{\texttt{Increase}}(x_{\texttt{Increase}}, w_{\texttt{Increase}})$ is defined as:

$$\mathcal{R}_{\texttt{Increase}} = \Big\{ \mathsf{ACCU.VfyMem}(\mathsf{pp_{ACCU}}, \mathsf{v_{ACCU}}, \mathsf{acct^{cm}_{old}}, \pi_{\mathsf{ACCU}}) = 1 \\ \wedge \mathsf{acct^{cm}_{new}} = \mathsf{Com}(\mathsf{sk_{acct}}, \mathsf{f.b_{old}} + v, \mathsf{p.b_{old}}, \mathsf{a.t}, \rho_{\mathsf{new}}; s_{\mathsf{new}}) \\ \wedge \mathsf{acct^{cm}_{old}} = \mathsf{Com}(\mathsf{sk_{acct}}, \mathsf{f.b_{old}}, \mathsf{p.b_{old}}, \mathsf{a.t}, \rho_{\mathsf{old}}; s_{\mathsf{old}}) \\ \wedge \mathsf{nul_{old}} = \mathsf{PRF}_{\mathsf{sk_{acct}}}(\rho_{\mathsf{old}}) \\ \wedge (\mathsf{pk_{acct}}, \mathsf{sk_{acct}}) \in \mathsf{Acc.KeyGen}(1^\lambda) \Big\}.$$

$\mathsf{tnx}_{\texttt{Increase}}$ Generation

▷ Upon receiving $(\texttt{Increase}, \mathsf{sid}, \mathsf{a.t}, v)$ from $\mathcal{Z}$, ignore if any of the following conditions hold:
- $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$ is not recorded.
- $v \leq 0$

▷ Else, parse $\mathsf{addr_{pk}} = (\mathsf{pk_{acct}}, \mathsf{pk_{en}})$.
▷ Retrieve the recorded entry $(\mathsf{acct^{cm}_{old}}, \mathsf{acct_{old}})$ where $\mathsf{acct_{old}} = (\mathsf{sk_{acct}}, \mathsf{f.b_{old}}, \mathsf{p.b_{old}}, \mathsf{a.t}, \rho_{\mathsf{old}}, s_{\mathsf{old}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\texttt{Read}, \mathsf{sid})$. Upon receiving $(\texttt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v_{ACCU}})$.
▷ Ignore if $(\mathsf{pk_{acct}}, \mathsf{a.t}, \cdot) \notin \mathbb{L}_{\mathsf{IAA}}$.
▷ Else, compute $\mathsf{nul_{old}} \leftarrow \mathsf{PRF}_{\mathsf{sk_{acct}}}(\rho_{\mathsf{old}})$.
▷ Sample $\rho_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$, and $s_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$.
▷ Set $\mathsf{acct_{new}} \leftarrow (\mathsf{sk_{acct}}, \mathsf{f.b_{old}} + v, \mathsf{p.b_{old}}, \mathsf{a.t}, \rho_{\mathsf{new}}, s_{\mathsf{new}})$.
▷ Compute $\mathsf{acct^{cm}_{new}} \leftarrow \mathsf{Com}(\mathsf{sk_{acct}}, \mathsf{f.b_{old}} + v, \mathsf{p.b_{old}}, \mathsf{a.t}, \rho_{\mathsf{new}}; s_{\mathsf{new}})$.
▷ Compute $\pi_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.PrvMem}(\mathsf{pp_{ACCU}}, \mathsf{state}', \mathsf{acct^{cm}_{old}})$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Prove}, \mathsf{sid}, x_{\texttt{Increase}}, w_{\texttt{Increase}})$ and receive $(\texttt{Proof}, \mathsf{sid}, \pi_{\texttt{Increase}})$.
▷ Set $\mathsf{tnx}_{\texttt{Increase}} \leftarrow (x_{\texttt{Increase}}, \pi_{\texttt{Increase}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\texttt{Append}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Increase}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\texttt{Read}, \mathsf{sid})$, upon receiving $(\texttt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, if $\mathsf{tnx}_{\texttt{Increase}} \in \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$, set $(\mathsf{acct^{cm}_{old}}, \mathsf{acct_{old}}) \leftarrow (\mathsf{acct^{cm}_{new}}, \mathsf{acct_{new}})$.
▷ Output $(\texttt{Increased}, \mathsf{sid}, \mathsf{a.t}, v)$ to $\mathcal{Z}$.

$\mathsf{tnx}_{\texttt{Increase}}$ Verification

▷ Execute $\textsc{Validate}_{\texttt{Increase}}(\mathsf{state}, \mathsf{tnx}_{\texttt{Increase}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v_{ACCU}})$, $\mathsf{tnx}_{\texttt{Increase}} = (x_{\texttt{Increase}}, \pi_{\texttt{Increase}})$, and $x_{\texttt{Increase}} = (\mathsf{nul_{old}}, \mathsf{acct^{cm}_{new}}, \mathsf{pk_{acct}}, \mathsf{a.t}, v)$.
- ignore if any of the following conditions hold:
  - $v \leq 0$
  - $\mathsf{nul_{old}} \in \mathbb{L}_{\mathsf{NUL}}$
  - $(\mathsf{pk_{acct}}, \mathsf{a.t}, \cdot) \notin \mathbb{L}_{\mathsf{IAA}}$

– calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x_{\mathtt{Increase}}, \pi_{\mathtt{Increase}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.

• else, output 1.

▷ Execute $\textsc{Update}_{\mathtt{Increase}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Increase}})$:

• parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Increase}} = (x_{\mathtt{Increase}}, \pi_{\mathtt{Increase}})$, and $x_{\mathtt{Increase}} = (\mathsf{nul}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{a.t}, v)$.

• call $\mathsf{v}_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}})$.

• set $\mathbb{L}_{\mathsf{NUL}} \leftarrow \mathbb{L}_{\mathsf{NUL}} \cup \{\mathsf{nul}_{\mathsf{old}}\}$.

• set $\mathsf{state} \leftarrow \mathsf{state} \cup \{(\cdot, \cdot, \cdot, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})\}$.

• output state.

Fig. 5: Increase asset supply transaction $\mathsf{tnx}_{\mathtt{Increase}}$

4.1.5 Sender transaction. The transaction occurs between a sender P^{s} and a receiver P^{r}, transferring an amount v of asset type $\mathsf{a.t}$ from the sender's account $\mathsf{acct}^{\mathsf{s}}$ to the receiver's account $\mathsf{acct}^{\mathsf{r}}$ via the help of ephemeral account-update-record AUR. P^{s} deducts v from $\mathsf{f.b}^{\mathsf{s}}$ and adds it to $\mathsf{p.b}^{\mathsf{s}}$, they also generate AUR for P^{r} with a value of v. AUR includes $(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t}; \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}})$. P^{s} encrypts $(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t})$ under $\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}$ using a key-private encryption generating $\mathsf{AUR}_{\mathsf{en}}$. This enables P^{r} to later scan the ledger using $\mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}$ to identify whether $\mathsf{AUR}_{\mathsf{en}}$ belongs to them and to claim it by providing a proof of knowledge of $\mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}$. The sender proves the well-formedness of AUR with respect to their account. Upon receiving a (valid) sender transaction $\mathsf{tnx}_{\mathtt{Send}}$, the ledger assigns transaction status $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}}$ to $\mathsf{tnx}_{\mathtt{Send}}$ with initial value of 0. As long as P^{r} has not claimed $\mathsf{AUR}_{\mathsf{en}}$, $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 0$, indicating that P^{s} can still reclaim the $\mathsf{AUR}_{\mathsf{en}}$ using the reversion algorithm (as we will discuss later). Once P^{r} claims $\mathsf{AUR}_{\mathsf{en}}$, $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}}$ is updated to 1, indicating that $\mathsf{AUR}_{\mathsf{en}}$ can no longer be reversed and also that P^{r} cannot claim it again.
P^{s} locally maintains a list $\mathsf{L}_{\mathsf{AUR}}$ (which is initially empty) that contains tuples of the form $(\mathsf{AUR}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}})$ where $\mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}}$ is a unique transaction identifier assigned by the ledger. $\mathsf{L}_{\mathsf{AUR}}$ is used when P^{s} needs to reverse a transaction (as described later). P^{s} proves ownership of fresh $\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}$ by providing a ZK proof of membership π_{ACCU} in the accumulator. P^{s} reveals $\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}$ of $\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}$ and proves the well-formedness of $\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}$. P^{s} also proves the well-formedness of $\mathsf{AUR}_{\mathsf{en}}$ and ψ, proving that the correct values are included in the plaintext, and that ψ is created under the public key of the auditor $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$ associated with the relevant asset type $\mathsf{a.t}$. Note that $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$ and $\mathsf{a.t}$ are part of the witness; and the link between the two is defined by $(\mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}) = \mathbb{L}_{\mathsf{IAA}}[i]$, where the index i is also part of the witness. The algorithm is provided in Figure 6.
The NIZK witness, statement, and relation are defined as follows.

- $w_{\mathtt{Send}} = (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}}, \rho^{\mathsf{s}}_{\mathsf{old}}, s^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \rho^{\mathsf{s}}_{\mathsf{new}}, s^{\mathsf{s}}_{\mathsf{new}}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t}, \pi_{\mathsf{ACCU}}, i)$
- $x_{\mathtt{Send}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi)$

- The relation $\mathcal{R}_{\mathtt{Send}}(x_{\mathtt{Send}}, w_{\mathtt{Send}})$ is defined as:

$$\begin{aligned}\mathcal{R}_{\mathtt{Send}} = \Big\{ & \mathsf{ACCU.VfyMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \pi_{\mathsf{ACCU}}) = 1 \\ & \wedge \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}} = \mathsf{Com}(\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} - v, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} + v, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{new}}; s^{\mathsf{s}}_{\mathsf{new}}) \\ & \wedge \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}} = \mathsf{Com}(\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{old}}; s^{\mathsf{s}}_{\mathsf{old}}) \\ & \wedge 0 < v \leq \mathsf{v}_{\mathsf{max}} \\ & \wedge \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} \geq v \\ & \wedge \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}} = \mathsf{PRF}_{\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}}(\rho^{\mathsf{s}}_{\mathsf{old}}) \\ & \wedge (\mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}) = \mathbb{L}_{\mathsf{IAA}}[i] \\ & \wedge (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}) \in \mathsf{Acc.KeyGen}(1^{\lambda}) \\ & \wedge \mathsf{AUR}_{\mathsf{en}} = \mathsf{Enc}_{\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t}) \\ & \wedge \psi = \mathsf{Enc}_{\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t}) \Big\}. \end{aligned}$$

$\mathsf{tnx}_{\mathtt{Send}}$ Generation

▷ Upon receiving $(\mathtt{Send}, \mathsf{sid}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$ from $\mathcal{Z}$ proceed as follows.
▷ Ignore if $v \leq 0$ or $v > \mathsf{v}_{\mathsf{max}}$. Else, parse $\mathsf{addr}^{\mathsf{s}}_{\mathsf{pk}} = (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{s}}_{\mathsf{en}})$, and $\mathsf{addr}^{\mathsf{s}}_{\mathsf{sk}} = (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{sk}^{\mathsf{s}}_{\mathsf{en}})$.
▷ Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{P}^{\mathsf{r}})$; upon receiving back $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{P}^{\mathsf{r}}, \mathsf{addr}^{\mathsf{r}}_{\mathsf{pk}})$ retrieve $\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}$.
▷ Retrieve the recorded entry $(\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{s}}_{\mathsf{old}})$ where $\mathsf{acct}^{\mathsf{s}}_{\mathsf{old}} = (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{old}}, s^{\mathsf{s}}_{\mathsf{old}})$.
▷ Compute $\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}} \leftarrow \mathsf{PRF}_{\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}}(\rho^{\mathsf{s}}_{\mathsf{old}})$.
▷ Sample $\rho^{\mathsf{s}}_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$, and $s^{\mathsf{s}}_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$.
▷ Set $\mathsf{acct}^{\mathsf{s}}_{\mathsf{new}} \leftarrow (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} - v, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} + v, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{new}}, s^{\mathsf{s}}_{\mathsf{new}})$.
▷ Compute $\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}} \leftarrow \mathsf{Com}(\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} - v, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} + v, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{new}}; s^{\mathsf{s}}_{\mathsf{new}})$.
▷ Set $\mathsf{AUR} \leftarrow (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t}; \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}})$.
▷ Compute $\mathsf{AUR}_{\mathsf{en}} \leftarrow \mathsf{Enc}_{\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$, upon receiving back $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ Using $\mathsf{a.t}$, retrieve $(\mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}) = \mathbb{L}_{\mathsf{IAA}}[i]$.
▷ Compute $\psi \leftarrow \mathsf{Enc}_{\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t})$.
▷ Compute $\pi_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.PrvMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{state}', \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}})$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, x_{\mathtt{Send}}, w_{\mathtt{Send}})$ and receive $(\mathtt{Proof}, \mathsf{sid}, \pi_{\mathtt{Send}})$.
▷ Set $\mathsf{tnx}_{\mathtt{Send}} \leftarrow (x_{\mathtt{Send}}, \pi_{\mathtt{Send}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Append}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Send}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$; upon receiving back $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$, if $\mathsf{TNX} \in \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$ where $\mathsf{tnx}_{\mathtt{Send}} \in \mathsf{TNX}$, set $(\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{s}}_{\mathsf{old}}) \leftarrow (\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{acct}^{\mathsf{s}}_{\mathsf{new}})$. Else, ignore.
▷ Parse $\mathsf{TNX} = (\mathsf{tnx}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}})$, and set $\mathsf{L}_{\mathsf{AUR}} \leftarrow \mathsf{L}_{\mathsf{AUR}} \cup \{(\mathsf{AUR}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}})\}$.
▷ Output $(\mathtt{Sent}, \mathsf{sid}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX})$ to $\mathcal{Z}$.

$\mathsf{tnx}_{\mathtt{Send}}$ Verification

▷ Execute $\textsc{Validate}_{\mathtt{Send}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Send}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Send}} = (x_{\mathtt{Send}}, \pi_{\mathtt{Send}})$, and $x_{\mathtt{Send}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi)$.
- Ignore if any of the following conditions hold:
  - $\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}} \in \mathbb{L}_{\mathsf{NUL}}$
  - upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x_{\mathtt{Send}}, \pi_{\mathtt{Send}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.
- else, output 1.

▷ Execute $\textsc{Update}_{\mathtt{Send}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Send}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Send}} = (x_{\mathtt{Send}}, \pi_{\mathtt{Send}})$, and $x_{\mathtt{Send}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi)$.
- sample a fresh $\mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}}$, and set $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} \leftarrow 0$, $\mathsf{TNX} \leftarrow (\mathsf{tnx}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}})$, and $\mathsf{state}' \leftarrow \mathsf{state}' \cup \{\mathsf{TNX}\}$.
- call $\mathsf{v}_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}})$.
- set $\mathbb{L}_{\mathsf{NUL}} \leftarrow \mathbb{L}_{\mathsf{NUL}} \cup \{\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}\}$.
- set $\mathsf{state} \leftarrow \mathsf{state} \cup \{(\mathsf{state}', \cdot, \cdot, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})\}$.
- output state.

Fig. 6: Sender transaction $\mathsf{tnx}_{\mathtt{Send}}$

4.1.6 Receiver transaction. Given TNX via $\mathcal{Z}$, the receiver P^{r} identifies whether $\mathsf{AUR}_{\mathsf{en}} \in \mathsf{TNX}$ is intended for them. P^{r} proves knowledge of the secret key $\mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}$ that can successfully decrypt $\mathsf{AUR}_{\mathsf{en}}$. Following this, P^{r} performs their account transition, proving that their new account is correctly updated with respect to the value v and asset type $\mathsf{a.t}$ contained within $\mathsf{AUR}_{\mathsf{en}}$. The algorithm is provided in Figure 7.
The NIZK witness, statement, and relation are defined as follows.

- $w_{\mathtt{Receive}} = (\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}}, \rho^{\mathsf{r}}_{\mathsf{old}}, s^{\mathsf{r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{old}}, \pi_{\mathsf{ACCU}}, \mathsf{p.b}^{\mathsf{r}}_{\mathsf{old}}, \rho^{\mathsf{r}}_{\mathsf{new}}, s^{\mathsf{r}}_{\mathsf{new}}, v, \mathsf{a.t}, \mathsf{sk}^{\mathsf{r}}_{\mathsf{en}})$
- $x_{\mathtt{Receive}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}})$
- The relation $\mathcal{R}_{\mathtt{Receive}}(x_{\mathtt{Receive}}, w_{\mathtt{Receive}})$ is defined as:

$$\begin{aligned}\mathcal{R}_{\mathtt{Receive}} = \Big\{ & \mathsf{ACCU.VfyMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{old}}, \pi_{\mathsf{ACCU}}) = 1 \\ & \wedge \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{new}} = \mathsf{Com}(\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}} + v, \mathsf{p.b}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{r}}_{\mathsf{new}}; s^{\mathsf{r}}_{\mathsf{new}}) \\ & \wedge \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{old}} = \mathsf{Com}(\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{r}}_{\mathsf{old}}; s^{\mathsf{r}}_{\mathsf{old}}) \\ & \wedge (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t}) = \mathsf{Dec}_{\mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{AUR}_{\mathsf{en}}) \\ & \wedge \mathsf{nul}^{\mathsf{r}}_{\mathsf{old}} = \mathsf{PRF}_{\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}}(\rho^{\mathsf{r}}_{\mathsf{old}}) \Big\}.\end{aligned}$$

$\mathsf{tnx}_{\texttt{Receive}}$ Generation

▷ Upon receiving $(\texttt{Receive}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{Z}$, call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\texttt{Read}, \mathsf{sid})$, upon receiving back $(\texttt{Read}, \mathsf{sid}, \mathsf{state})$, ignore if $\mathsf{TNX} \notin \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$.
▷ Else, parse $\mathsf{addr}^{\mathsf{r}}_{\mathsf{sk}} = (\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{sk}^{\mathsf{r}}_{\mathsf{en}})$, $\mathsf{TNX} = (\mathsf{tnx}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}})$, $\mathsf{tnx}_{\mathtt{Send}} = (x_{\mathtt{Send}}, \pi_{\mathtt{Send}})$, and $x_{\mathtt{Send}} = (\mathsf{v}'_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi)$. Ignore if $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} \neq 0$.
▷ Compute $o \leftarrow \mathsf{Dec}_{\mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{AUR}_{\mathsf{en}})$. Ignore if $o = \perp$.
▷ Else, parse $o = (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t})$, and retrieve the recorded entry $(\mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{r}}_{\mathsf{old}})$ where $\mathsf{acct}^{\mathsf{r}}_{\mathsf{old}} = (\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{r}}_{\mathsf{old}}, s^{\mathsf{r}}_{\mathsf{old}})$.
▷ Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\texttt{Rtrv.id}, \mathsf{sid}, \mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}})$, upon receiving back $(\texttt{Rtrved.id}, \mathsf{sid}, \mathsf{P}^{\mathsf{s}})$ proceed.
▷ Output $(\texttt{Receiving}, \mathsf{sid}, \mathsf{P}^{\mathsf{s}}, v, \mathsf{a.t}, \mathsf{TNX})$ to $\mathcal{Z}$.
▷ Parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ Compute $\mathsf{nul}^{\mathsf{r}}_{\mathsf{old}} \leftarrow \mathsf{PRF}_{\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}}(\rho^{\mathsf{r}}_{\mathsf{old}})$.
▷ Sample $\rho^{\mathsf{r}}_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$, and $s^{\mathsf{r}}_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$.
▷ Set $\mathsf{acct}^{\mathsf{r}}_{\mathsf{new}} = (\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}} + v, \mathsf{p.b}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{r}}_{\mathsf{new}}, s^{\mathsf{r}}_{\mathsf{new}})$.
▷ Compute $\mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{new}} \leftarrow \mathsf{Com}(\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}} + v, \mathsf{p.b}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{r}}_{\mathsf{new}}; s^{\mathsf{r}}_{\mathsf{new}})$.
▷ Compute $\pi_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.PrvMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{state}', \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{old}})$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Prove}, \mathsf{sid}, x_{\mathtt{Receive}}, w_{\mathtt{Receive}})$ and receive $(\texttt{Proof}, \mathsf{sid}, \pi_{\mathtt{Receive}})$.
▷ Set $\mathsf{tnx}_{\mathtt{Receive}} \leftarrow (x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$.
▷ If $(\texttt{Affirm}, \mathsf{sid}, \mathsf{TNX})$ has been received from $\mathcal{Z}$, call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\texttt{Append}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Receive}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\texttt{Read}, \mathsf{sid})$; upon receiving back $(\texttt{Read}, \mathsf{sid}, \mathsf{state})$, if $\mathsf{tnx}_{\mathtt{Receive}} \in \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$, set $(\mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{r}}_{\mathsf{old}}) \leftarrow (\mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{new}}, \mathsf{acct}^{\mathsf{r}}_{\mathsf{new}})$
▷ Output $(\texttt{Received}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{Z}$.

$\mathsf{tnx}_{\texttt{Receive}}$ Verification

▷ Execute $\textsc{Validate}_{\texttt{Receive}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Receive}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Receive}} = (x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$, and $x_{\mathtt{Receive}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}})$.
- ignore if any of the following conditions hold:
  - given $\mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}} \in x_{\mathtt{Receive}}$, retrieve the associated $\mathsf{AUR}^{*}_{\mathsf{en}} \in \mathsf{state}'$ and $\mathsf{AUR}^{*}_{\mathsf{en}} \neq \mathsf{AUR}_{\mathsf{en}}$.
  - $\mathsf{nul}^{\mathsf{r}}_{\mathsf{old}} \in \mathbb{L}_{\mathsf{NUL}}$
  - upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\texttt{Verify}, \mathsf{sid}, x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$, $(\texttt{Vrfed}, \mathsf{sid}, 0)$ is returned.
  - $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} \neq 0$
- Else, output 1.

▷ Execute $\textsc{Update}_{\texttt{Receive}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Receive}})$:

- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Receive}} = (x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$, and $x_{\mathtt{Receive}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}})$.
- given $\mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}$, retrieve $\mathsf{TNX} = (\mathsf{tnx}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}})$.
- set $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} \leftarrow 1$, $\mathsf{TNX} \leftarrow (\cdot, \cdot, \mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}})$, and $\mathsf{state}' \leftarrow \mathsf{state}' \cup \{\mathsf{TNX}\}$.
- call $\mathsf{v}_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{new}})$.
- set $\mathbb{L}_{\mathsf{NUL}} \leftarrow \mathbb{L}_{\mathsf{NUL}} \cup \{\mathsf{nul}^{\mathsf{r}}_{\mathsf{old}}\}$.
- set $\mathsf{state} \leftarrow \mathsf{state} \cup \{(\mathsf{state}', \cdot, \cdot, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})\}$.
- output state.

Fig. 7: Receiver transaction $\mathsf{tnx}_{\mathtt{Receive}}$

4.1.7 Reversion. The $\mathsf{AUR}_{\mathsf{en}}$ is reversible by the sender P^{s} as long as the corresponding receiver P^{r} has not claimed it. P^{s} proves that they are the creator of $\mathsf{AUR}_{\mathsf{en}}$ by proving knowledge of the associated secret key $\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}$, thereby proving their role as its creator. If $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$ (see Figure 1), P^{s} can reverse/reclaim $\mathsf{AUR}_{\mathsf{en}}$. Given the value v of $\mathsf{AUR}_{\mathsf{en}}$, P^{s} reverses the transaction by increasing their finalized balance $\mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} + v$ and decreasing their pending balance $\mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} - v$ (recall, $\mathsf{p.b}^{\mathsf{s}}$ is increased by v when P^{s} initially generated $\mathsf{AUR}_{\mathsf{en}}$). The algorithm is provided in the Figure 8.
The NIZK witness, statement, and relation are defined as follows.

- $w_{\mathtt{Reverse}} = (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}}, \rho^{\mathsf{s}}_{\mathsf{old}}, s^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}}, \rho^{\mathsf{s}}_{\mathsf{new}}, s^{\mathsf{s}}_{\mathsf{new}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t}, \pi_{\mathsf{ACCU}})$
- $x_{\mathtt{Reverse}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}})$
- The relation $\mathcal{R}_{\mathtt{Reverse}}(x_{\mathtt{Reverse}}, w_{\mathtt{Reverse}})$ is defined as:

$$\begin{aligned}\mathcal{R}_{\mathtt{Reverse}} = \Big\{ & \mathsf{ACCU.VfyMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \pi_{\mathsf{ACCU}}) = 1 \\ & \wedge \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}} = \mathsf{PRF}_{\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}}(\rho^{\mathsf{s}}_{\mathsf{old}}) \\ & \wedge \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}} = \mathsf{Com}(\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} + v, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} - v, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{new}}; s^{\mathsf{s}}_{\mathsf{new}}) \\ & \wedge \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}} = \mathsf{Com}(\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{old}}; s^{\mathsf{s}}_{\mathsf{old}}) \\ & \wedge \mathsf{AUR}_{\mathsf{en}} = \mathsf{Enc}_{\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t}) \\ & \wedge (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}) \in \mathsf{Acc.KeyGen}(1^{\lambda}) \Big\}.\end{aligned}$$

$\mathsf{tnx}_{\mathtt{Reverse}}$ Generation

▷ Upon receiving $(\mathtt{Reverse}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{Z}$, call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$, upon receiving $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, ignore if $\mathsf{TNX} \notin \mathsf{state}$.
▷ Else, parse $\mathsf{addr}^{\mathsf{s}}_{\mathsf{sk}} = (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{sk}^{\mathsf{s}}_{\mathsf{en}})$, $\mathsf{TNX} = (\mathsf{tnx}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}})$, $\mathsf{tnx}_{\mathsf{Send}} = (x_{\mathsf{Send}}, \pi_{\mathsf{Send}})$, and $x_{\mathsf{Send}} = (\mathsf{v}'_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi)$. Ignore if $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} \neq 0$.
▷ Else, retrieve the entry $(\cdot, \cdot, \mathsf{tnx}^{*\mathsf{id}}_{\mathsf{Send}})$, from $\mathsf{L}_{\mathsf{AUR}}$ where $\mathsf{tnx}^{*\mathsf{id}}_{\mathsf{Send}} = \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}$ and parse $\mathsf{AUR} = (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t}; \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}})$. Ignore if no such entry exists.

▷ Else, retrieve the recorded entry $(\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{s}}_{\mathsf{old}})$ where $\mathsf{acct}^{\mathsf{s}}_{\mathsf{old}} = (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{old}}, s^{\mathsf{s}}_{\mathsf{old}})$.
▷ Compute $\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}} \leftarrow \mathsf{PRF}_{\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}}(\rho^{\mathsf{s}}_{\mathsf{old}})$.
▷ Sample $\rho^{\mathsf{s}}_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$, and $s^{\mathsf{s}}_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$.
▷ Set $\mathsf{acct}^{\mathsf{s}}_{\mathsf{new}} \leftarrow (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} + v, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} - v, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{new}}, s^{\mathsf{s}}_{\mathsf{new}})$.
▷ Compute $\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}} \leftarrow \mathsf{Com}(\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}} + v, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}} - v, \mathsf{a.t}, \rho^{\mathsf{s}}_{\mathsf{new}}; s^{\mathsf{s}}_{\mathsf{new}})$.
▷ Parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ Compute $\pi_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.PrvMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{state}', \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}})$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, x_{\mathtt{Reverse}}, w_{\mathtt{Reverse}})$ and receive $(\mathtt{Proof}, \mathsf{sid}, \pi_{\mathtt{Reverse}})$.
▷ Set $\mathsf{tnx}_{\mathtt{Reverse}} \leftarrow (x_{\mathtt{Reverse}}, \pi_{\mathtt{Reverse}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Append}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Reverse}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$; upon receiving $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, if $\mathsf{tnx}_{\mathtt{Reverse}} \in \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$, set $(\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{s}}_{\mathsf{old}}) \leftarrow (\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{acct}^{\mathsf{s}}_{\mathsf{new}})$, and set $\mathsf{L}_{\mathsf{AUR}} \leftarrow \mathsf{L}_{\mathsf{AUR}} \setminus \{(\mathsf{AUR}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}})\}$.
▷ Output $(\mathtt{Reversed}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{Z}$.

$\mathsf{tnx}_{\mathtt{Reverse}}$ Verification

▷ Execute $\textsc{Validate}_{\mathtt{Reverse}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Reverse}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Reverse}} = (x_{\mathtt{Reverse}}, \pi_{\mathtt{Reverse}})$, and $x_{\mathtt{Reverse}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}})$.
- ignore if any of the following conditions hold:
  - given $\mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}} \in x_{\mathtt{Reverse}}$, retrieve the associated $\mathsf{AUR}^{*}_{\mathsf{en}} \in \mathsf{state}'$ and $\mathsf{AUR}^{*}_{\mathsf{en}} \neq \mathsf{AUR}_{\mathsf{en}}$.
  - $\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}} \in \mathbb{L}_{\mathsf{NUL}}$
  - upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x_{\mathtt{Reverse}}, \pi_{\mathtt{Reverse}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.
  - $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} \neq 0$
- else, output 1.

▷ Execute $\textsc{Update}_{\mathtt{Reverse}}(\mathsf{state}, \mathsf{tnx}_{\mathtt{Reverse}})$:
- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathtt{Reverse}} = (x_{\mathtt{Reverse}}, \pi_{\mathtt{Reverse}})$, and $x_{\mathtt{Reverse}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}})$.
- given $\mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}$, retrieve $\mathsf{TNX} = (\mathsf{tnx}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}})$.
- set $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} \leftarrow 2$, $\mathsf{TNX} \leftarrow (\cdot, \cdot, \mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}})$, and $\mathsf{state}' \leftarrow \mathsf{state}' \cup \{\mathsf{TNX}\}$.
- call $\mathsf{v}_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}})$.
- set $\mathbb{L}_{\mathsf{NUL}} \leftarrow \mathbb{L}_{\mathsf{NUL}} \cup \{\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}\}$.
- set $\mathsf{state} \leftarrow \mathsf{state} \cup \{(\mathsf{state}', \cdot, \cdot, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})\}$.
- output state.

Fig. 8: Reversion transaction $\mathsf{tnx}_{\mathtt{Reverse}}$

4.1.8 Proof of balance. We present two approaches for proof of balance (PoB). In the first approach, the party P interacts with the associated auditor A (for the asset type $\mathsf{a.t}$ whose balance is requested). A outputs $\mathsf{f.b}$ and $\mathsf{p.b}$ to the environment $\mathcal{Z}$. In the second approach, outlined as a generic solution in Appendix A, P proves both their $\mathsf{f.b}$ and $\mathsf{p.b}$ directly to any PoB verifier, without relying on A. The first approach is provided in Figure 9, where P reveals their $\mathsf{f.b}$ and $\mathsf{p.b}$ from their associated account and proves the correctness of the revealed values. There is no account state transition in PoB; however, P must reveal the nullifier nul to enable A to verify that P is disclosing $\mathsf{f.b}$ and $\mathsf{p.b}$ corresponding to their most *recent* account. P also provides pointers to all transactions with $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$ and $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$ to justify the value of $\mathsf{p.b}$. The values $\mathsf{f.b}$ and $\mathsf{p.b} - V^{(1)}$ are revealed to $\mathcal{Z}$, where $V^{(1)}$ is the sum of the values of P's transactions that have been finalized (with $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$) and in which P has been the sender.
The NIZK witness, statement, and relation are defined as follows.

- $w^{\mathsf{A}}_{\mathrm{PoB}} = (\mathsf{sk}_{\mathsf{acct}}, \rho, s)$
- $x^{\mathsf{A}}_{\mathrm{PoB}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{nul}, \mathsf{acct}^{\mathsf{cm}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t})$
- The relation $\mathcal{R}_{\mathrm{PoB}}(x^{\mathsf{A}}_{\mathrm{PoB}}, w^{\mathsf{A}}_{\mathrm{PoB}})$ is defined as:

$$\mathcal{R}_{\mathrm{PoB}} = \Big\{ \begin{aligned} &\mathsf{acct}^{\mathsf{cm}} = \mathsf{Com}(\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}, \rho; s) \\ &\wedge (\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}}) \in \mathsf{Acc.KeyGen}(1^{\lambda}) \\ &\wedge \mathsf{nul} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho) \Big\}. \end{aligned}$$

$\mathsf{tnx}_{\mathbf{PoB}}$ Generation executed by P

▷ Upon receiving $(\mathtt{Balance}, \mathsf{sid}, \mathsf{a.t})$ from $\mathcal{Z}$, parse $\mathsf{addr}_{\mathsf{pk}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{pk}_{\mathsf{en}})$, and $\mathsf{addr}_{\mathsf{sk}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{en}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$, upon receiving $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ Retrieve the recorded entry $(\mathsf{acct}^{\mathsf{cm}}, \mathsf{acct})$ where $\mathsf{acct} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}, \rho, s)$.
▷ Compute $\mathsf{nul} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho)$.
▷ Call $\mathcal{F}^{\dagger}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, x^{\mathsf{A}}_{\mathrm{PoB}}, w^{\mathsf{A}}_{\mathrm{PoB}})$ and receive $(\mathtt{Proof}, \mathsf{sid}, \pi^{\mathsf{A}}_{\mathrm{PoB}})$.
▷ Retrieve all entries $(\mathsf{AUR}^i, \mathsf{AUR}^i_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}_i}_{\mathsf{Send}})$ in $\mathsf{L}_{\mathsf{AUR}}$ where, upon parsing $\mathsf{AUR}^i = (\mathsf{pk}^{\mathsf{s}_i}_{\mathsf{acct}}, v^{(i)}, \mathsf{a.t}^i; \mathsf{pk}^{\mathsf{r}_i}_{\mathsf{en}})$ the conditions $\mathsf{a.t}^i = \mathsf{a.t}$ and $\mathsf{pk}^{\mathsf{s}_i}_{\mathsf{acct}} = \mathsf{pk}_{\mathsf{acct}}$ hold.
▷ Define two lists $\overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}$ and $\overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)}$, both initially set to $\varnothing$.
▷ Retrieve all $\mathsf{TNX}^j \in \mathsf{state}'$ where $\mathsf{TNX}^j = (\mathsf{tnx}^j_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{id}_j}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{st}_j}_{\mathsf{Send}})$.
▷ For each $\mathsf{tnx}^{\mathsf{id}_i}_{\mathsf{Send}}$ retrieved from $\mathsf{L}_{\mathsf{AUR}}$:
  - add $\mathsf{tnx}^{\mathsf{id}_i}_{\mathsf{Send}}$ to $\overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}$ if $\mathsf{tnx}^{\mathsf{id}_i}_{\mathsf{Send}} = \mathsf{tnx}^{\mathsf{id}_j}_{\mathsf{Send}}$ & $\mathsf{tnx}^{\mathsf{st}_j}_{\mathsf{Send}} = 1$ for some $\mathsf{TNX}^j = (\mathsf{tnx}^j_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{id}_j}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{st}_j}_{\mathsf{Send}})$.

- add $\mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}}$ to $\overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)}$ if $\mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}} = \mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}}$ & $\mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}} = 0$ for some $\mathsf{TNX}^j = (\mathsf{tnx}^j_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}})$.

▷ Using $\mathsf{a.t}$, retrieve $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$ from $\mathbb{L}_{\mathsf{IAA}}$.
▷ Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Rtrv.id}, \mathsf{sid}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$, upon receiving $(\mathtt{Rtrved.id}, \mathsf{sid}, \mathsf{A})$ proceed.
▷ Set $\mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}} \leftarrow (x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)})$, and call $\mathcal{F}_{\mathsf{ch}}$ with $(\mathtt{Send}, \mathsf{sid}, \mathsf{A}, \mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}})$.

PoB output executed by A

▷ Upon receiving $(\mathtt{Received}, \mathsf{sid}, \mathsf{P}, \mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}})$ from $\mathcal{F}_{\mathsf{ch}}$, parse $\mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}} = (x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)})$, and $x^{\mathsf{A}}_{\mathrm{PoB}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{nul}, \mathsf{acct}^{\mathsf{cm}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t})$.
▷ Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{P})$. Ignore if $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{P}, (\mathsf{pk}'_{\mathsf{acct}}, \cdot))$ is received from $\mathcal{F}_{\mathsf{KeyReg}}$ where $\mathsf{pk}'_{\mathsf{acct}} \neq \mathsf{pk}_{\mathsf{acct}}$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$, upon receiving $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ Ignore if any of the following conditions hold:
- $\mathsf{nul} \in \mathbb{L}_{\mathsf{NUL}}$
- $\mathsf{acct}^{\mathsf{cm}} \notin \mathsf{state}'$
- upon calling $\mathcal{F}^{\dagger}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.

▷ For all $\mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}} \in \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}$:
- ignore if there is no $\mathsf{TNX}^i \in \mathsf{state}'$ where $\mathsf{TNX}^i = (\mathsf{tnx}^i_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}_i}_{\mathtt{Send}})$ with $\mathsf{tnx}^{\mathsf{st}_i}_{\mathtt{Send}} = 1$.
- parse $\mathsf{tnx}^i_{\mathtt{Send}} = (x^i_{\mathtt{Send}}, \pi^i_{\mathtt{Send}})$, and $x^i_{\mathtt{Send}} = (\mathsf{v}^i_{\mathsf{ACCU}}, \mathsf{nul}^i, \mathsf{acct}^{\mathsf{cm},i}, \mathsf{AUR}^i_{\mathsf{en}}, \psi^i)$.
- compute decryption $o^i \leftarrow \mathsf{Dec}_{\mathsf{sk}^{\mathsf{A}}_{\mathsf{en}}}(\psi^i)$. Ignore if $o^i = \perp$. Else, parse $o^i = (\mathsf{pk}^{(i)}_{\mathsf{acct}}, \cdot, v^{(i)}, \mathsf{a.t}^{(i)})$, and ignore if $\mathsf{pk}^{(i)}_{\mathsf{acct}} \neq \mathsf{pk}_{\mathsf{acct}}$, or $\mathsf{a.t}^{(i)} \neq \mathsf{a.t}$. Else, set $V^{(1)} \leftarrow V^{(1)} + v^{(i)}$ where initially $V^{(1)} = 0$.

▷ For all $\mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}} \in \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)}$:
- ignore if there is no $\mathsf{TNX}^i \in \mathsf{state}'$ where $\mathsf{TNX}^i = (\mathsf{tnx}^i_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}_i}_{\mathtt{Send}})$ with $\mathsf{tnx}^{\mathsf{st}_i}_{\mathtt{Send}} = 0$.
- parse $\mathsf{tnx}^i_{\mathtt{Send}} = (x^i_{\mathtt{Send}}, \pi^i_{\mathtt{Send}})$, and $x^i_{\mathtt{Send}} = (\mathsf{v}^i_{\mathsf{ACCU}}, \mathsf{nul}^i, \mathsf{acct}^{\mathsf{cm},i}, \mathsf{AUR}^i_{\mathsf{en}}, \psi^i)$.
- compute decryption $o^i \leftarrow \mathsf{Dec}_{\mathsf{sk}^{\mathsf{A}}_{\mathsf{en}}}(\psi^i)$. Ignore if $o^i = \perp$. Else, parse $o^i = (\mathsf{pk}^{(i)}_{\mathsf{acct}}, \cdot, v^{(i)}, \mathsf{a.t}^{(i)})$, and ignore if $\mathsf{pk}^{(i)}_{\mathsf{acct}} \neq \mathsf{pk}_{\mathsf{acct}}$, or $\mathsf{a.t}^{(i)} \neq \mathsf{a.t}$. Else, set $V^{(0)} \leftarrow V^{(0)} + v^{(i)}$ where initially $V^{(0)} = 0$.

▷ Ignore if $V^{(1)} + V^{(0)} \neq \mathsf{p.b}$.
▷ Else, output $(\mathtt{Balance}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{f.b}, V^{(0)})$ to $\mathcal{Z}$.

Fig. 9: Proof of balance transaction $\mathsf{tnx}_{\mathrm{PoB}}$ – 1st Approach

4.1.9 Auditor operation. The auditor can access the hidden information of a particular transaction where its underlying asset type corresponds to them. The algorithm is provided in Figure 10.

▷ Upon receiving $(\texttt{Audit}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{Z}$, call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\texttt{Read}, \mathsf{sid})$, upon receiving $(\texttt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, ignore if $\mathsf{TNX} \notin \mathsf{state}$.
▷ Else, parse $\mathsf{addr}^{\mathsf{A}}_{\mathsf{sk}} = (\mathsf{sk}^{\mathsf{A}}_{\mathsf{acct}}, \mathsf{sk}^{\mathsf{A}}_{\mathsf{en}})$, $\mathsf{TNX} = (\mathsf{tnx}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}})$, $\mathsf{tnx}_{\mathsf{Send}} = (x_{\mathsf{Send}}, \pi_{\mathsf{Send}})$, and $x_{\mathsf{Send}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi)$.
▷ Compute $o \leftarrow \mathsf{Dec}_{\mathsf{sk}^{\mathsf{A}}_{\mathsf{en}}}(\psi)$. Ignore if $o = \perp$.
▷ Else, parse $o = (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t})$.
▷ Call $\mathcal{F}_{\mathsf{KeyReg}}$ with $(\texttt{Rtrv.id}, \mathsf{sid}, \mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}})$ and $(\texttt{Rtrv.id}, \mathsf{sid}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}})$.
▷ Upon receiving $(\texttt{Rtrved.id}, \mathsf{sid}, \mathsf{P}^{\mathsf{s}})$ and $(\texttt{Rtrved.id}, \mathsf{sid}, \mathsf{P}^{\mathsf{r}})$ from $\mathcal{F}_{\mathsf{KeyReg}}$ respectively output $(\texttt{Audited}, \mathsf{sid}, \mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$ to $\mathcal{Z}$.

Fig. 10: Auditor algorithm (off-chain operation)

5 Discussion

5.1 Multi-party transactions

DART facilitates efficient multi-party transactions (MPTs). Specifically, a single sender can, with one account state transition (i.e., one accumulator proof), send funds to multiple receivers. Similarly, a single receiver can, with one account state transition, affirm (receive) funds from multiple independent senders who have sent funds in separate transactions. The process works as follows: a sender generates multiple $\mathsf{AUR}_{\mathsf{en}}$ and proves, in zero-knowledge, that the total value deducted from their account equals the sum of all values encapsulated within these $\mathsf{AUR}_{\mathsf{en}}$. The sender then submits a transaction such that each associated receiver can subsequently claim their respective $\mathsf{AUR}_{\mathsf{en}}$. The ledger updates the status of each $\mathsf{AUR}_{\mathsf{en}}$ individually, determining whether the corresponding receiver has affirmed it so that the sender can reverse any $\mathsf{AUR}_{\mathsf{en}}$ independently. Similarly, when a receiver wishes to affirm (claim) multiple $\mathsf{AUR}_{\mathsf{en}}$, they prove that they can successfully decrypt all associated $\mathsf{AUR}_{\mathsf{en}}$ and that the total value added to their account equals the sum of the affirmed $\mathsf{AUR}_{\mathsf{en}}$. In this way, DART enables efficient MPTs through a *single* account state transition. Note that in MPTs, ephemeral anonymous linkability occurs. An anonymous sender in one transaction includes multiple $\mathsf{AUR}_{\mathsf{en}}$ to anonymous receivers, or an anonymous receiver references multiple $\mathsf{AUR}_{\mathsf{en}}$ generated by anonymous senders. However, each transaction is *unlinkable* to any previous or subsequent transactions. This ensures that an adversary cannot link the transactions of an entity involved in a MPT (whether as a sender or receiver) to their prior or future transactions, thereby preserving unlinkability.

Aggregatable transactions for efficient account updates. With the same mechanism described above for MPT, a user (sender) can batch a group of

transactions and reverse them all via a single account transition in one $\mathsf{tnx}_{\texttt{Reverse}}$, rather than submitting separate transactions individually. The user references multiple transactions and proves that they are the creator of those transactions. Additionally, the user proves that the value being added to their finalized balance and deducted from their pending balance equals the sum of the values within all the transactions they intend to reverse.

5.2 Fee payments

In our formal modeling, we do not explicitly address transaction fee payments to maintain focus on our primary contribution. However, we outline here how fee payments can be incorporated into the system. Each user account could include an additional element representing the balance in the system's currency dedicated to fee payment, from which transaction fees are deducted. This extension can be easily integrated into the zero-knowledge relations. The NIZK proof would enable efficient verification, ensuring that the correct transaction fee has been deducted from the balance of the fee payment asset. Each transaction type could have a predefined fee (reflecting differences in their verification costs). Users would then prove, in zero-knowledge, that the appropriate fee has been deducted from their balance based on the transaction type.

5.3 Maintenance of $\mathbb{L}_{\mathsf{IAA}}$

The list $\mathbb{L}_{\mathsf{IAA}}$ with entries of the form $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$ is publicly maintained on-chain and can be updated arbitrarily. For instance, in practice, an issuer may choose to change the associated auditor A for their asset due to regulations. This could occur, for example, if the originally assigned auditor becomes unavailable or is revoked, prompting the issuer to update the auditor's address $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$ to another one. In such a case, the list will undergo an update. The issuer proves that they are the legitimate associated issuer for the specific $\mathbb{L}_{\mathsf{IAA}}$ entry to authorize this update. Consequently, the list can be updated arbitrarily, and this could invalidate a user's zero-knowledge proof if it was generated based on a previous state of the list that has since changed. To mitigate this issue, we propose introducing structured update intervals; specifically, updates to lists can be restricted to predefined times. Users can generate their zero-knowledge proofs with the assurance that the list will remain stable within a defined period.
Moreover, we do not accumulate elements of $\mathbb{L}_{\mathsf{IAA}}$, e.g., using a cryptographic accumulator as we do for accumulating account commitments. We note that it is straightforward to accumulate the list $\mathbb{L}_{\mathsf{IAA}}$ so that chain validators do not need to read the whole list for each transaction. Instead, they read the updated list at the beginning of an interval and compute the accumulator value for it. Later, until the end of the interval, they only check a user's proof against a single accumulator value. Users also generate their proof based on the accumulator, not the list itself, similar to how they do so for their account commitments.
In implementation, $\mathbb{L}_{\mathsf{IAA}}$ could bear other attributes such as an asset description to provide more information about the asset, or the total value issued of $\mathsf{a.t}$

for auditability purposes (e.g., checking the balance of the off-chain real-world assets (RWAs) against their on-chain representations as in PARScoin [48]).

6 Implementation details

In Section 4, we described the components of our system Π_{DART} in terms of cryptographic building blocks. In this section, we elaborate on the instantiation of these elements by specifying the concrete schemes used.

Building blocks. Our system leverages cryptographic schemes such as ElGamal encryption [23] (Definition 5) for $\mathsf{AUR}_{\mathsf{en}}$ and ψ, Pedersen commitments [46] (Definition 10) for $\mathsf{acct}^{\mathsf{cm}}$, and the Dodis-Yampolskiy pseudorandom function (PRF) [21] (Definition 13) for $\mathsf{nul} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho)$. These schemes are algebraic and hence integrate efficiently with the algebraic-friendly non-interactive zero-knowledge proofs. A critical component of our design is the use of cryptographic accumulators. Our accumulator scheme is instantiated using Curve Trees [12], which are transparent and do not require a trusted setup. A key feature of Curve Trees is that they support efficient updates to the accumulator value without necessitating access to the entire set, analogous to the behavior of Merkle Trees. Curve Trees relays on the commit-and-prove capabilities of Bulletproofs—to prove accumulator membership in zero-knowledge.

NIZK proofs. In our implementation, we use two types of NIZK proofs: Σ- protocols and Bulletproofs [9], both made non-interactive through the Fiat-Shamir transform [25]. Both have transparent setups and are computationally sound in the random oracle model under the discrete logarithm assumption. While neither of these protocols in their standard forms provides UC security, transformations like the one in [32] show how they can be adapted to achieve UC security. We refer the reader to [18] for further details about such transformations.
Some of the statements we have, such as proving the well-formedness of commitments, encryptions, or decryption computations, can be handled straightforwardly with Σ-protocols. Here, we focus on the more complex statements. Range checks are handled using Bulletproof range proofs. To prove the correct mapping between asset types and their corresponding auditors, we assume that the public list $\mathbb{L}_{\mathsf{IAA}} = \{(\mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})\}$ is represented as two vectors $\mathbb{L}^{(1)}_{\mathsf{IAA}} := \{\mathsf{a.t}_1, \ldots, \mathsf{a.t}_n\}$ and $\mathbb{L}^{(2)}_{\mathsf{IAA}} := \{\mathsf{pk}^{\mathsf{A}_1}_{\mathsf{en}}, \ldots, \mathsf{pk}^{\mathsf{A}_n}_{\mathsf{en}}\}$, respectively storing all registered asset types and public keys of their associated auditors. At any point in time when n assets are registered, the prover can demonstrate the correct mapping by committing to a binary vector V_I of size n with exactly one non-zero entry, and then showing that the asset type and auditor's public key involved in the statement correspond to the inner products of $\langle V_I, \mathbb{L}^{(1)}_{\mathsf{IAA}} \rangle$ and $\langle V_I, \mathbb{L}^{(2)}_{\mathsf{IAA}} \rangle$, respectively. This again can be accomplished using Bulletproof's inner-product argument [9].
To prove accumulator membership in zero-knowledge, we use Curve Trees [12], which relies on the commit-and-prove capabilities of Bulletproofs. While other candidates for instantiating our accumulator scheme exist (e.g., Merkle trees), we choose an algebraic accumulator as it better integrates with our design and

makes the arithmetization of Bulletproofs required for converting accumulator statements into R1CS more efficient.

Handling historical accumulators for proof verification. The blockchain validators cannot just maintain the most recent accumulator value, because, in that case, a proof generated using an old accumulator will no longer pass verification. To overcome this issue, the chain stores a list of the last several accumulator values. This means one can reference an older accumulator value when generating their proof. As long as that accumulator value is still within the list of stored historical values, the user's NIZK proof is validated.

Avoiding exhaustive search. We use "ElGamal in the exponent" for encrypting field elements in $\mathsf{AUR}_{\mathsf{en}} = \mathsf{Enc}_{\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t})$ and $\psi = \mathsf{Enc}_{\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t})$. This results in an inefficient decryption, where the decryptor needs to extract x from g^x. To address this, we use a method that integrates an additional encryption scheme, denoted by $\mathsf{ENC}^{\mathsf{cnst}}$, into the construction while ensuring constant-time decryption. In our formal model, $\mathsf{ENC}^{\mathsf{cnst}}$ is not explicitly incorporated and remains an implementation-level enhancement that can be *added* alongside the existing ElGamal encryption. Also, the original encryption scheme cannot be replaced by $\mathsf{ENC}^{\mathsf{cnst}}$, as the ability to efficiently prove properties about plaintexts in zero-knowledge proofs must be preserved. $\mathsf{ENC}^{\mathsf{cnst}}$ is employed to recover field element plaintexts in constant time, and the result is verified against their group encoding, which is computed from ElGamal decryption (recall that the sender proves the well-formedness of the ElGamal ciphertext). If, for example, the decryption of $\mathsf{AUR}_{\mathsf{en}}$ (resulted in g^v) and the $\mathsf{ENC}^{\mathsf{cnst}}$ ciphertext (resulted in v') are inconsistent $g^v \neq g^{v'}$, the receiver may simply ignore the transaction[15] and the sender can reverse the transaction, requiring them to pay additional fees to do so. Hence, the potentially malicious sender has an incentive to generate the $\mathsf{ENC}^{\mathsf{cnst}}$ ciphertext correctly. Moreover, any claims to the funds rely solely on the ElGamal ciphertext (for which efficient zero-knowledge proof of well-formedness has already been submitted) and not on $\mathsf{ENC}^{\mathsf{cnst}}$.
For completeness, the construction of $\mathsf{ENC}^{\mathsf{cnst}}$ is specified below. The key generation process mirrors the procedure outlined in Definition 5 for ElGamal encryption. The constant-time encryption and decryption algorithms $\mathsf{Enc}^{\mathsf{cnst}}$ and $\mathsf{Dec}^{\mathsf{cnst}}$ are defined as follows:

- $\mathsf{Enc}^{\mathsf{cnst}}(\mathsf{pk}, m)$: Given a public key pk and a message $m \in M$, sample $r, q \xleftarrow{\$} \mathbb{Z}_p$ uniformly at random. Compute:

$$\psi_1 = g^r, \quad \psi_2 = \mathsf{pk}^r \cdot h^q, \quad \psi_3 = \mathsf{H}(h^q) \oplus m,$$

  and output the ciphertext $C = (\psi_1, \psi_2, \psi_3)$.
- $\mathsf{Dec}^{\mathsf{cnst}}(\mathsf{sk}, C)$: Parse $C = (\psi_1, \psi_2, \psi_3)$. Compute: $h^q = \frac{\psi_2}{\psi_1^{\mathsf{sk}}}$, and recover the plaintext message as $m = \mathsf{H}(h^q) \oplus \psi_3$.

[15] However, in principle, the receiver could still brute-force v in $\mathsf{AUR}_{\mathsf{en}}$ and claim the funds.

Account recovery. For account recovery, we adopt a similar approach to PARScoin [48], which relies on standard recovery methods. Values such as nullifier randomness ρ or commitment randomness s can be pseudorandomly derived from a secret key by inputting a counter (e.g., starting at zero) and the secret key into a pseudorandom function (PRF). To recover the account, the user regenerates nullifier nul values up to the latest one recorded on the blockchain, retrieving the most recent randomness values. For account balance, once the randomness is recovered, balances can be found either via brute-force or by decrypting public ciphertexts—encrypted by the user per transaction. Similarly, randomness for encryptions $\mathsf{AUR}_{\mathsf{en}}$ and ψ can be regenerated via a pseudorandom generator, with associated values brute-forced or decrypted using the user's long-term secret key (requiring the user to generate separate encryptions per transaction).

7 DART security proof

In this section, we prove the security of our construction Π_{DART}. The following is our main theorem:

Theorem 1. *Assuming that the public key encryption scheme (Definition 1) is (i) unconditionally correct (Definition 2), (ii) computationally CPA secure (Definition 3), and (iii) computationally key private (Definition 4); the commitment scheme (Definition 6) is (i) unconditionally correct (Definition 7), (ii) unconditionally hiding (Definition 8) and (iii) computationally binding (Definition 9); the pseudorandom function (PRF) scheme (Definition 11) is (i) unconditionally correct, and (ii) computationally pseudorandom (Definition 12); the accumulator scheme (Definition 14) is (i) unconditionally correct (Definition 15), and (ii) computationally sound (Definition 16); no PPT environment $\mathcal{Z}$ can distinguish the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{DART}},\mathcal{A},\mathcal{Z}}$ from the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{DART}},\mathcal{S},\mathcal{Z}}$ in the $\{\mathcal{F}_{\mathsf{NIZK}}, \mathcal{F}^{\dagger}_{\mathsf{NIZK}}, \mathcal{F}_{\mathsf{Ledger}}, \mathcal{F}_{\mathsf{ch}}, \mathcal{F}_{\mathsf{KeyReg}}\}$-hybrid model with static corruptions in the presence of arbitrary number of malicious parties (including all entities within the system issuers, senders, receivers, and auditors) with advantage better than:*

$$\mathsf{Adv}^{\mathsf{Bind}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{KeyPriv}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{PRF.Pseudo}}_{\mathcal{A}} + \mathsf{Adv}^{\mathsf{Sound}}_{\mathcal{A}}$$

7.1 Simulation

In the following, we present our simulator $\mathcal{S}$ description, which ensures that the real-world execution $\mathsf{EXEC}_{\Pi_{\mathsf{DART}},\mathcal{A},\mathcal{Z}}$ is indistinguishable from the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{DART}},\mathcal{S},\mathcal{Z}}$ for any PPT environment $\mathcal{Z}$. The environment $\mathcal{Z}$ selects the session identifier sid. The simulator $\mathcal{S}$ operates by internally running an instance of Π_{DART} and the real-world adversary $\mathcal{A}$, while simulating the actions of honest users. Its purpose is to ensure that the internally simulated view of $\mathcal{A}$ in the ideal-world is indistinguishable from its real-world view. $\mathcal{S}$ extracts necessary information (from adversary's messages) and instructs the functionality to produce outputs that are indistinguishable for honest users.

At the start of execution, the environment $\mathcal{Z}$ directs the adversary (simulator in the ideal-world) to corrupt specific parties through a message of the form $(\texttt{Corrupt}, \mathsf{sid}, \mathsf{P})$, where P represents a network entity. Upon receiving corruption messages, the simulator $\mathcal{S}$ informs $\mathcal{F}_{\mathsf{DART}}$ of the corrupted parties by sending the message $(\texttt{Corrupt}, \mathsf{sid}, \mathsf{P})$ and records these corrupted parties for future interactions. The adversary $\mathcal{A}$ has control over the corrupted parties, and $\mathcal{S}$ interacts with $\mathcal{F}_{\mathsf{DART}}$ on their behalf. During the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{DART}},\mathcal{S},\mathcal{Z}}$, honest dummy parties directly transmit their inputs from $\mathcal{Z}$ to $\mathcal{F}_{\mathsf{DART}}$. The simulator $\mathcal{S}$ is designed to reproduce the adversary $\mathcal{A}$'s view in the real-world execution for the dummy internally run adversary in the ideal world, while simulating the behavior of honest parties. $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}^{\dagger}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{Ledger}}$, $\mathcal{F}_{\mathsf{ch}}$, and $\mathcal{F}_{\mathsf{KeyReg}}$, so whenever the adversary, on behalf of a malicious network entity, calls a functionality, the simulator sees the identifier (e.g., P). Emulation of these ideal functionalities involves maintaining specific lists associated with each functionality. However, for simplicity and without loss of generality, we assume that $\mathcal{S}$ manages the states of these functionalities without explicitly elaborating on all such list maintenance.
Throughout the following, we assume that the simulator $\mathcal{S}$ adheres to the real-world algorithms described in Section 4.1 whenever emulating an honest entity or an ideal functionality, without explicitly detailing steps of the algorithms. However, in instances where the simulator diverges from these real-world procedures—such as simulating cryptographic elements based on information disclosed by the ideal functionality $\mathcal{F}_{\mathsf{DART}}$ or extracting a zero-knowledge witness from the adversary's proof—we explicitly highlight such deviations.

Address generation.
(i) Honest user P:

- Upon receiving $(\texttt{Gen.Addr}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{F}_{\mathsf{DART}}$, initiate honest user P emulation.
- Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, generate key pairs $(\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ and update the state of $\mathcal{F}_{\mathsf{KeyReg}}$.
- Send $(\texttt{Gen.Addr.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{addr}_{\mathsf{pk}})$ to $\mathcal{F}_{\mathsf{DART}}$.

(ii) Malicious user P:

- Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, upon receiving $(\texttt{Gen.Key}, \mathsf{sid})$ from the adversary (malicious party P), submit $(\texttt{Gen.Addr}, \mathsf{sid})$ on behalf of P to $\mathcal{F}_{\mathsf{DART}}$.
- Upon receiving $(\texttt{Gen.Addr}, \mathsf{sid}, \mathsf{P})$ from $\mathcal{F}_{\mathsf{DART}}$, if emulating $\mathcal{F}_{\mathsf{KeyReg}}$ $(\mathsf{P}, \mathsf{addr}_{\mathsf{pk}}, \cdot)$ has been recorded, output $(\texttt{Gen.Addr.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{addr}_{\mathsf{pk}})$ to $\mathcal{F}_{\mathsf{DART}}$.

Asset issuance.
(i) Honest user P:

- Upon receiving $(\texttt{Issue}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{A})$ from $\mathcal{F}_{\mathsf{DART}}$, start honest user P emulation.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\texttt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.

- Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, using P and A leaked by $\mathcal{F}_{\mathsf{DART}}$, retrieve $(\mathsf{P}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ and $(\mathsf{A}, \mathsf{addr}^{\mathsf{A}}_{\mathsf{pk}}, \mathsf{addr}^{\mathsf{A}}_{\mathsf{sk}})$, respectively. Ignore if such entries do not exist.
- Retrieve $\mathsf{pk}_{\mathsf{acct}}$ from $\mathsf{addr}_{\mathsf{pk}}$ and $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$ from $\mathsf{addr}^{\mathsf{A}}_{\mathsf{pk}}$.
- Set $x_{\texttt{Issue}} \leftarrow (\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, leak $(\texttt{Prove}, \mathsf{sid}, x_{\texttt{Issue}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi_{\texttt{Issue}})$ from $\mathcal{A}$, update the $\mathcal{F}_{\mathsf{NIZK}}$ state by storing $(x_{\texttt{Issue}}, \pi_{\texttt{Issue}})$.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Append.req}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Issue}})$ to $\mathcal{A}$, where $\mathsf{tnx}_{\texttt{Issue}} = (x_{\texttt{Issue}}, \pi_{\texttt{Issue}})$.
- Upon receiving $(\texttt{Append.req.Ok}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Issue}})$ from $\mathcal{A}$, send $(\texttt{Issue.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{A})$ to $\mathcal{F}_{\mathsf{DART}}$.
- Update the $\mathcal{F}_{\mathsf{Ledger}}$ state by executing $\textsc{Update}_{\texttt{Issue}}$ following the real-world algorithms provided in Figure 3.

(ii) Malicious user P:

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, receive $(\texttt{Append}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Issue}})$ upon calling $\mathcal{F}_{\mathsf{Ledger}}$ by the adversary (malicious party P).
- Parse $\mathsf{tnx}_{\texttt{Issue}} = (x_{\texttt{Issue}}, \pi_{\texttt{Issue}})$. Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\texttt{Verify}, \mathsf{sid}, x_{\texttt{Issue}}, \pi_{\texttt{Issue}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Witness}, \mathsf{sid}, w_{\texttt{Issue}})$ from $\mathcal{A}$, check if the relation $\mathcal{R}_{\texttt{Issue}}(x_{\texttt{Issue}}, w_{\texttt{Issue}})$ holds. If so, store $(x_{\texttt{Issue}}, \pi_{\texttt{Issue}})$; otherwise, ignore.
- Parse $x_{\texttt{Issue}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}})$.
- Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, given $\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}$, retrieve A.
- Send $(\texttt{Issue}, \mathsf{sid}, \mathsf{a.t}, \mathsf{A})$ on behalf of corrupted party P to $\mathcal{F}_{\mathsf{DART}}$.
- Upon receiving $(\texttt{Issue}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{A})$ from $\mathcal{F}_{\mathsf{DART}}$, submit $(\texttt{Issue.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, \mathsf{A})$ back to $\mathcal{F}_{\mathsf{DART}}$ if execution of $\textsc{Validate}_{\texttt{Issue}}$, following the real-world algorithm provided in Figure 3, outputs 1.
- Execute $\textsc{Update}_{\texttt{Issue}}$ provided in Figure 3.

Account registration.

(i) Honest user P:

- Upon receiving $(\texttt{Register}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t})$ from $\mathcal{F}_{\mathsf{DART}}$, start honest user P emulation.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\texttt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.
- Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, retrieve $(\mathsf{sk}_{\mathsf{acct}}, \mathsf{pk}_{\mathsf{acct}})$ using P leaked by $\mathcal{F}_{\mathsf{DART}}$.
- Compute $\mathsf{acct}^{\mathsf{cm}}$ as in the algorithm described in Figure 4 using $\mathsf{a.t}$ leaked by $\mathcal{F}_{\mathsf{DART}}$.
- Set $x_{\texttt{Register}} \leftarrow (\mathsf{acct}^{\mathsf{cm}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t})$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, leak $(\texttt{Prove}, \mathsf{sid}, x_{\texttt{Register}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi_{\texttt{Register}})$ from $\mathcal{A}$, update the $\mathcal{F}_{\mathsf{NIZK}}$ state by storing $(x_{\texttt{Register}}, \pi_{\texttt{Register}})$.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Append.req}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Register}})$ to $\mathcal{A}$, where $\mathsf{tnx}_{\texttt{Register}} = (x_{\texttt{Register}}, \pi_{\texttt{Register}})$.

- Upon receiving $(\texttt{Append.req.Ok}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Register}})$ from $\mathcal{A}$, send $(\texttt{Register.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t})$ to $\mathcal{F}_{\mathsf{DART}}$.
- Update the state of $\mathcal{F}_{\mathsf{Ledger}}$ by executing $\textsc{Update}_{\texttt{Register}}$ following the real-world algorithm provided in Figure 4.

(ii) Malicious user P:

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, receive $(\texttt{Append}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Register}})$ upon calling $\mathcal{F}_{\mathsf{Ledger}}$ by the malicious user P.
- Parse $\mathsf{tnx}_{\texttt{Register}} = (x_{\texttt{Register}}, \pi_{\texttt{Register}})$. Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\texttt{Verify}, \mathsf{sid}, x_{\texttt{Register}}, \pi_{\texttt{Register}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Witness}, \mathsf{sid}, w_{\texttt{Register}})$ from $\mathcal{A}$, check if the relation $\mathcal{R}_{\texttt{Register}}(x_{\texttt{Register}}, w_{\texttt{Register}})$ holds. If so, store $(x_{\texttt{Register}}, \pi_{\texttt{Register}})$; otherwise, ignore.
- Parse $x_{\texttt{Register}} = (\mathsf{acct}^{\mathsf{cm}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t})$.
- Send $(\texttt{Register}, \mathsf{sid}, \mathsf{a.t})$ to $\mathcal{F}_{\mathsf{DART}}$ on behalf of the malicious user P.
- Upon receiving $(\texttt{Register}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t})$ from $\mathcal{F}_{\mathsf{DART}}$, send back $(\texttt{Register.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t})$ to $\mathcal{F}_{\mathsf{DART}}$ if the execution of $\textsc{Validate}_{\texttt{Register}}$, following the real-world algorithm provided in Figure 4, outputs 1.
- Execute $\textsc{Update}_{\texttt{Register}}$ provided in Figure 4.

Increase asset supply.

(i) Honest user P:

- Upon receiving $(\texttt{Increase}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, v)$ from $\mathcal{F}_{\mathsf{DART}}$, start honest user P emulation.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\texttt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.
- Compute a commitment $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}$ on dummy values.
- Sample a fresh random value from the PRF range as $\mathsf{nul}_{\mathsf{old}}$, subject to the condition described in proof of balance simulation (step *), using the user identifier P leaked via $\mathcal{F}_{\mathsf{DART}}$.
- Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, retrieve $\mathsf{pk}_{\mathsf{acct}}$ given P leaked via $\mathcal{F}_{\mathsf{DART}}$.
- Set $x_{\texttt{Increase}} \leftarrow (\mathsf{nul}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, v)$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, leak $(\texttt{Prove}, \mathsf{sid}, x_{\texttt{Increase}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi_{\texttt{Increase}})$ from $\mathcal{A}$, update the $\mathcal{F}_{\mathsf{NIZK}}$ state by storing $(x_{\texttt{Increase}}, \pi_{\texttt{Increase}})$.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Append.req}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Increase}})$ to $\mathcal{A}$, where $\mathsf{tnx}_{\texttt{Increase}} = (x_{\texttt{Increase}}, \pi_{\texttt{Increase}})$.
- Upon receiving $(\texttt{Append.req.Ok}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Increase}})$ from $\mathcal{A}$, send $(\texttt{Increase.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, v)$ to $\mathcal{F}_{\mathsf{DART}}$.
- Update the $\mathcal{F}_{\mathsf{Ledger}}$ state by executing $\textsc{Update}_{\texttt{Increase}}$ following the real-world algorithm provided in Figure 5.
- Record $(\cdot, \mathsf{P}, v, \mathsf{a.t}, \cdot, \texttt{increased})$.

(ii) Malicious user P:

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, receive $(\texttt{Append}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Increase}})$ upon calling $\mathcal{F}_{\mathsf{Ledger}}$ by the malicious user P.

- Parse $\mathsf{tnx}_{\mathtt{Increase}} = (x_{\mathtt{Increase}}, \pi_{\mathtt{Increase}})$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\mathtt{Verify}, \mathsf{sid}, x_{\mathtt{Increase}}, \pi_{\mathtt{Increase}})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Witness}, \mathsf{sid}, w_{\mathtt{Increase}})$ from $\mathcal{A}$, check if the relation $\mathcal{R}_{\mathtt{Increase}}(x_{\mathtt{Increase}}, w_{\mathtt{Increase}})$ holds. If so, store $(x_{\mathtt{Increase}}, \pi_{\mathtt{Increase}})$; otherwise, ignore.
- Parse $x_{\mathtt{Increase}} = (\mathsf{nul}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, v)$.
- Send $(\mathtt{Increase}, \mathsf{sid}, \mathsf{a.t}, v)$ on behalf of the corrupted user P to $\mathcal{F}_{\mathsf{DART}}$, where $(\mathsf{a.t}, v) \in x_{\mathtt{Increase}}$.
- Upon receiving $(\mathtt{Increase}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, v)$ from $\mathcal{F}_{\mathsf{DART}}$, send back $(\mathtt{Increase.Ok}, \mathsf{sid}, \mathsf{P}, \mathsf{a.t}, v)$ to $\mathcal{F}_{\mathsf{DART}}$ if the execution of $\textsc{Validate}_{\mathtt{Increase}}$, following the real-world algorithm provided in Figure 5, outputs 1.
- Execute $\textsc{Update}_{\mathtt{Increase}}$ provided in Figure 5.

Sender transaction.
(i) Honest sender P^{s}:

- Upon receiving $(\mathtt{Send}, \mathsf{sid}, \beta)$ from $\mathcal{F}_{\mathsf{DART}}$, start emulation of an honest sender.
- In case the auditor A or receiver P^{r} is malicious, parse $\beta = (\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$.[16]
- In case the auditor A or receiver P^{r} is malicious, emulating $\mathcal{F}_{\mathsf{KeyReg}}$, retrieve key pairs $\mathsf{addr}^{\mathsf{r}}_{\mathsf{pk}} = (\mathsf{pk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}})$ and $\mathsf{addr}^{\mathsf{s}}_{\mathsf{pk}} = (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{s}}_{\mathsf{en}})$ using the identifiers P^{s} and P^{r} leaked by $\mathcal{F}_{\mathsf{DART}}$; ignore if such an entry does not exist.
- In case the auditor A or receiver P^{r} is malicious, retrieve $(\cdot, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}, \mathsf{a.t})$ from $\mathbb{L}_{\mathsf{IAA}}$ using $\mathsf{a.t}$ leaked by $\mathcal{F}_{\mathsf{DART}}$; ignore if such an entry does not exist.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\mathtt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\mathtt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.
- Compute a commitment $\mathsf{acct}^{\mathsf{cm},\mathsf{s}}_{\mathsf{new}}$ on dummy values.
- Sample a fresh random value from the PRF range as $\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}$, subject to the condition described in proof of balance simulation (step *), using the sender identifier P^{s} leaked via $\mathcal{F}_{\mathsf{DART}}$ in case the auditor A is malicious.
- Compute $\mathsf{AUR}_{\mathsf{en}}$ and ψ:
  - In case both the auditor A and receiver P^{r} are honest, compute encryptions on dummy values as $\mathsf{AUR}_{\mathsf{en}}$ and ψ.
  - In the case of a malicious receiver P^{r}, compute $\mathsf{AUR}_{\mathsf{en}} \leftarrow \mathsf{Enc}_{\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t})$ (recall that in this case, the simulator receives $\beta = (\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$ from $\mathcal{F}_{\mathsf{DART}}$).
  - In the case of a malicious auditor A, compute $\psi \leftarrow \mathsf{Enc}_{\mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}}(\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t})$ (recall that in this case, the simulator receives $\beta = (\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$ from $\mathcal{F}_{\mathsf{DART}}$).
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, retrieve the most recent $\mathsf{v}_{\mathsf{ACCU}}$.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\mathtt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\mathtt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.
- Set $x_{\mathtt{Send}} \leftarrow (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm},\mathsf{s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi)$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, leak $(\mathtt{Prove}, \mathsf{sid}, x_{\mathtt{Send}})$ to $\mathcal{A}$.

[16] Otherwise, β is a random value.

- Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi_{\texttt{Send}})$ from $\mathcal{A}$, update the $\mathcal{F}_{\mathsf{NIZK}}$ state by storing $(x_{\texttt{Send}}, \pi_{\texttt{Send}})$.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Append.req}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Send}})$ to $\mathcal{A}$, where $\mathsf{tnx}_{\texttt{Send}} = (x_{\texttt{Send}}, \pi_{\texttt{Send}})$.
- Upon receiving $(\texttt{Append.req.Ok}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Send}})$ from $\mathcal{A}$, send $(\texttt{Send.Ok}, \mathsf{sid}, \beta, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$, where TNX is generated by executing $\textsc{Update}_{\texttt{Send}}$ as described in Figure 6.
- In case the auditor A is malicious, record $(\mathsf{P}^s, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{sent})$.

(ii) Malicious sender P^{s}:

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, receive $(\texttt{Append}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Send}})$ upon calling $\mathcal{F}_{\mathsf{Ledger}}$ by the malicious sender P^{s}.
- Parse $\mathsf{tnx}_{\texttt{Send}} = (x_{\texttt{Send}}, \pi_{\texttt{Send}})$. Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\texttt{Verify}, \mathsf{sid}, x_{\texttt{Send}}, \pi_{\texttt{Send}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Witness}, \mathsf{sid}, w_{\texttt{Send}})$ from $\mathcal{A}$, check if the relation $\mathcal{R}_{\texttt{Send}}(x_{\texttt{Send}}, w_{\texttt{Send}})$ holds. If so, store $(x_{\texttt{Send}}, \pi_{\texttt{Send}})$; otherwise, ignore.
- Parse

$$w_{\texttt{Send}} = (\mathsf{sk}^{\mathsf{s}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{p.b}^{\mathsf{s}}_{\mathsf{old}}, \rho^{\mathsf{s}}_{\mathsf{old}}, s^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{old}}, \rho^{\mathsf{s}}_{\mathsf{new}}, s^{\mathsf{s}}_{\mathsf{new}}, \mathsf{pk}^{\mathsf{A}}_{\mathsf{en}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t}, \pi_{\mathsf{ACCU}}, i).$$

- Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, retrieve P^{r} given $\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}$.
- Submit $(\texttt{Send}, \mathsf{sid}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$ on behalf of the malicious sender P^{s} to $\mathcal{F}_{\mathsf{DART}}$.
- Upon receiving $(\texttt{Send}, \mathsf{sid}, \beta)$ from $\mathcal{F}_{\mathsf{DART}}$, proceed as follows.
- Similar to the case above, if A or P^{r} is malicious, parse $\beta = (\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t})$ and ignore if there is an inconsistency with $w_{\texttt{Send}}$ extracted.
- Otherwise, ignore if executing the $\textsc{Validate}_{\texttt{Send}}$ algorithm provided in Figure 6 does not output 1.
- Otherwise, execute the $\textsc{Update}_{\texttt{Send}}$ algorithm provided in Figure 6 to generate $\mathsf{TNX} = (\mathsf{tnx}_{\texttt{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\texttt{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\texttt{Send}})$.
- Submit $(\texttt{Send.Ok}, \mathsf{sid}, \beta, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$.
- In case the auditor A is malicious, record $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{sent})$ where $(v, \mathsf{a.t}) \in w_{\texttt{Send}}$.

Receiver transaction.
(i) Honest receiver P^{r}:

- Start emulation of an honest receiver:
  - Upon receiving $(\texttt{Receive}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$ if P^{s} is honest.
  - Upon receiving $(\texttt{Receive}, \mathsf{sid}, \mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$ if P^{s} is malicious.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\texttt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.
- In case P^{s} is malicious, ignore if any of the following conditions hold, following the algorithm described in Figure 7:
  - $\mathsf{TNX} \notin \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$, where state is retrieved by emulating $\mathcal{F}_{\mathsf{Ledger}}$.

  - $o \neq (\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}, v, \mathsf{a.t})$, where $\mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}$ (in $\mathsf{Dec}_{\mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{AUR}_{\mathsf{en}})$) and $\mathsf{pk}^{\mathsf{s}}_{\mathsf{acct}}$ are retrieved given P^{r} and P^{s} leaked via $\mathcal{F}_{\mathsf{DART}}$.[17]
  - $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} \neq 0$.
- Otherwise, send $(\mathtt{Receive.Ok}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$.
- Compute a commitment $\mathsf{acct}^{\mathsf{cm},\mathsf{r}}_{\mathsf{new}}$ on dummy values.
- Sample a fresh random value from the PRF range as $\mathsf{nul}^{\mathsf{r}}_{\mathsf{old}}$, subject to the condition described in proof of balance simulation (step *), using the receiver identifier P^{r} retrieved from the entry recorded $(\cdot, \mathsf{P}^{\mathsf{r}}, \cdot, \cdot, \mathsf{TNX}, \mathtt{sent})$ using TNX leaked via $\mathcal{F}_{\mathsf{DART}}$ in case the auditor A is malicious (see the sender transaction emulation).
- Parse

$$\mathsf{TNX} = (\mathsf{tnx}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}}), \quad \mathsf{tnx}_{\mathtt{Send}} = (x_{\mathtt{Send}}, \pi_{\mathtt{Send}}),$$

$$x_{\mathtt{Send}} = (\mathsf{v}'_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm},\mathsf{s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi).$$

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, retrieve the most recent $\mathsf{v}_{\mathsf{ACCU}}$.
- Set $x_{\mathtt{Receive}} \leftarrow (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm},\mathsf{r}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}})$, where $\mathsf{AUR}_{\mathsf{en}} \in x_{\mathtt{Send}}$ and $\mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}} \in \mathsf{TNX}$.
- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, leak $(\mathtt{Prove}, \mathsf{sid}, x_{\mathtt{Receive}})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Proof}, \mathsf{sid}, \pi_{\mathtt{Receive}})$ from $\mathcal{A}$, update the $\mathcal{F}_{\mathsf{NIZK}}$ state by storing $(x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\mathtt{Append.req}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Receive}})$ to $\mathcal{A}$, where $\mathsf{tnx}_{\mathtt{Receive}} = (x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$.
- Upon receiving $(\mathtt{Append.req.Ok}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Receive}})$ from $\mathcal{A}$, proceed as follows.
- Upon receiving $(\mathtt{Affirmed}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$, execute $\textsc{Update}_{\mathtt{Receive}}$ as described in Figure 7.
- Submit $(\mathtt{Affirmed.Ok}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$.
- In case the auditor A is malicious, given TNX, retrieve the entry recorded $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \mathtt{sent})$, and record $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \mathtt{received})$.

(ii) Malicious receiver P^{r}:

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, receive $(\mathtt{Append}, \mathsf{sid}, \mathsf{tnx}_{\mathtt{Receive}})$ upon calling $\mathcal{F}_{\mathsf{Ledger}}$ by the malicious receiver P^{r}.
- Parse $\mathsf{tnx}_{\mathtt{Receive}} = (x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$. Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\mathtt{Verify}, \mathsf{sid}, x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Witness}, \mathsf{sid}, w_{\mathtt{Receive}})$ from $\mathcal{A}$, check if the relation $\mathcal{R}_{\mathtt{Receive}}(x_{\mathtt{Receive}}, w_{\mathtt{Receive}})$ holds. If so, store $(x_{\mathtt{Receive}}, \pi_{\mathtt{Receive}})$; otherwise, ignore.
- Parse

$$x_{\mathtt{Receive}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm},\mathsf{r}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}}).$$

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, given $\mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}} \in x_{\mathtt{Receive}}$, retrieve the associated TNX and submit $(\mathtt{Receive}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$ on behalf of the malicious receiver P^{r}. If no such TNX exists, ignore.

[17] The simulator could also check this via witness extraction in the malicious sender's proof using $\mathsf{tnx}_{\mathtt{Send}} = (x_{\mathtt{Send}}, \pi_{\mathtt{Send}})$.

- In case P^{s} is honest, receive $(\texttt{Receive}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$ and proceed to the next step. If P^{s} is malicious, upon receiving $(\texttt{Receive}, \mathsf{sid}, \mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$, check the consistency of leaked information with

$$w_{\texttt{Receive}} = (\mathsf{sk}^{\mathsf{r}}_{\mathsf{acct}}, \mathsf{f.b}^{\mathsf{r}}_{\mathsf{old}}, \rho^{\mathsf{r}}_{\mathsf{old}}, s^{\mathsf{r}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,r}}_{\mathsf{old}}, \pi_{\mathsf{ACCU}}, \mathsf{p.b}^{\mathsf{r}}_{\mathsf{old}}, \rho^{\mathsf{r}}_{\mathsf{new}}, s^{\mathsf{r}}_{\mathsf{new}}, v, \mathsf{a.t}, \mathsf{sk}^{\mathsf{r}}_{\mathsf{en}}).$$

  Given the keys, retrieve the associated identifiers by emulating $\mathcal{F}_{\mathsf{KeyReg}}$. Ignore if there is inconsistency.
- Submit $(\texttt{Receive.Ok}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$.
- Ignore if executing the $\textsc{Validate}_{\texttt{Receive}}$ algorithm provided in Figure 7 does not output 1.
- Otherwise, execute the $\textsc{Update}_{\texttt{Receive}}$ algorithm provided in Figure 7.
- Submit $(\texttt{Affirm}, \mathsf{sid}, \mathsf{TNX})$ on behalf of P^{r} to $\mathcal{F}_{\mathsf{DART}}$.
- Upon receiving $(\texttt{Affirmed}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$, submit back $(\texttt{Affirmed.Ok}, \mathsf{sid}, \mathsf{TNX})$.
- In case the auditor A is malicious, given TNX, retrieve the entry recorded $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{sent})$, and record $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{received})$.

Reversion.

(i) Honest sender P^{s}:

- Upon receiving $(\texttt{Reverse}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$, start emulation of an honest sender.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\texttt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.
- Compute a commitment $\mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}$ on dummy values.
- Sample a fresh random value from the PRF range as $\mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}$, subject to the condition described in proof of balance simulation (step *), using the sender identifier P^{s} retrieved by looking for an entry recorded of the form $(\mathsf{P}^{\mathsf{s}}, \cdot, \cdot, \cdot, \mathsf{TNX}, \texttt{sent})$ where TNX is leaked via $\mathcal{F}_{\mathsf{DART}}$ in case the auditor A is malicious.
- Parse

$$\mathsf{TNX} = (\mathsf{tnx}_{\texttt{Send}}, \mathsf{tnx}^{\mathsf{id}}_{\texttt{Send}}, \mathsf{tnx}^{\mathsf{st}}_{\texttt{Send}}), \quad \mathsf{tnx}_{\texttt{Send}} = (x_{\texttt{Send}}, \pi_{\texttt{Send}}),$$

$$x_{\texttt{Send}} = (\mathsf{v}'_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \psi).$$

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, retrieve the most recent $\mathsf{v}_{\mathsf{ACCU}}$.
- Set

$$x_{\texttt{Reverse}} \leftarrow (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\texttt{Send}}).$$

- Emulating $\mathcal{F}_{\mathsf{NIZK}}$, leak $(\texttt{Prove}, \mathsf{sid}, x_{\texttt{Reverse}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi_{\texttt{Reverse}})$ from $\mathcal{A}$, update the $\mathcal{F}_{\mathsf{NIZK}}$ state by storing $(x_{\texttt{Reverse}}, \pi_{\texttt{Reverse}})$.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Append.req}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Reverse}})$ to $\mathcal{A}$, where $\mathsf{tnx}_{\texttt{Reverse}} = (x_{\texttt{Reverse}}, \pi_{\texttt{Reverse}})$.
- Upon receiving $(\texttt{Append.req.Ok}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Reverse}})$ from $\mathcal{A}$, send $(\texttt{Reverse.Ok}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$.
- Execute $\textsc{Update}_{\texttt{Reverse}}$ as described in Figure 8.

- In case the auditor A is malicious, given TNX, retrieve the entry recorded $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{sent})$, and record $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{reversed})$.

(ii) Malicious sender P^{s}:

- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, receive $(\texttt{Append}, \mathsf{sid}, \mathsf{tnx}_{\texttt{Reverse}})$ upon calling $\mathcal{F}_{\mathsf{Ledger}}$ by the malicious sender P^{s}.
- Parse $\mathsf{tnx}_{\texttt{Reverse}} = (x_{\texttt{Reverse}}, \pi_{\texttt{Reverse}})$. Emulating $\mathcal{F}_{\mathsf{NIZK}}$, send $(\texttt{Verify}, \mathsf{sid}, x_{\texttt{Reverse}}, \pi_{\texttt{Reverse}})$ to $\mathcal{A}$.
- Upon receiving $(\texttt{Witness}, \mathsf{sid}, w_{\texttt{Reverse}})$ from $\mathcal{A}$, check if the relation $\mathcal{R}_{\texttt{Reverse}}(x_{\texttt{Reverse}}, w_{\texttt{Reverse}})$ holds. If so, store $(x_{\texttt{Reverse}}, \pi_{\texttt{Reverse}})$; otherwise, ignore.
- Parse
$$x_{\texttt{Reverse}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}^{\mathsf{s}}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm,s}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}}).$$
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, given $\mathsf{tnx}^{\mathsf{id}}_{\mathsf{Send}} \in x_{\texttt{Reverse}}$, retrieve the associated TNX and submit $(\texttt{Reverse}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$ on behalf of the malicious sender P^{s}. If no such TNX exists, ignore.
- Upon receiving $(\texttt{Reverse}, \mathsf{sid}, \mathsf{TNX})$ from $\mathcal{F}_{\mathsf{DART}}$, proceed as follows.
- Ignore if executing the $\textsc{Validate}_{\texttt{Reverse}}$ algorithm provided in Figure 8 does not output 1.
- Otherwise, execute the $\textsc{Update}_{\texttt{Reverse}}$ algorithm provided in Figure 8.
- Submit $(\texttt{Reverse.Ok}, \mathsf{sid}, \mathsf{TNX})$ to $\mathcal{F}_{\mathsf{DART}}$.
- In case the auditor A is malicious, given TNX, retrieve the entry recorded $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{sent})$, and record $(\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, v, \mathsf{a.t}, \mathsf{TNX}, \texttt{reversed})$.

Proof of balance.

(i) Honest user P:

- Upon receiving $(\texttt{Balance}, \mathsf{sid}, \gamma)$ from $\mathcal{F}_{\mathsf{DART}}$:
  - If A is malicious, parse $\gamma = (\mathsf{P}, \mathsf{a.t})$.
  - Otherwise, parse γ as a random value.
- Emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ. Proceed as follows if $(\texttt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.
- If A is malicious:
  - Emulating $\mathcal{F}_{\mathsf{KeyReg}}$, retrieve $\mathsf{pk}_{\mathsf{acct}}$ given P leaked by $\mathcal{F}_{\mathsf{DART}}$.
  - (*) Sample a fresh random value from the PRF range as nul and record $(\mathsf{P}, \mathsf{nul}, 0)$. Use the recorded entry $(\mathsf{P}, \mathsf{nul}, 0)$ whenever it is necessary to simulate a nullifier value for the honest user P in any protocol (with the malicious auditor A) that requires revealing a nullifier.[18] Retrieve nul from $(\mathsf{P}, \mathsf{nul}, 0)$, set the nullifier to be simulated as nul, and update the entry to $(\mathsf{P}, \mathsf{nul}, 1)$. The next time a nullifier needs to be simulated for honest P, if the entry for P is $(\mathsf{P}, \mathsf{nul}, 1)$, sample a fresh random value from the PRF range as the nullifier.
  - Retrieve the latest commitment (on dummy values) simulated for P as $\mathsf{acct}^{\mathsf{cm}}$.[19]

[18] Namely, increasing asset supply, sender transactions, receiver transactions, and reversion.

[19] Recall, the simulator assigns TNX for each transaction, and in case A is malicious, the simulator gets to know the sender P^{s} and receiver P^{r} of the transaction.

  - Retrieve all the entries recorded of the form:
    * $(\cdot, \mathsf{P}^*, \cdot, \cdot, \cdot, \texttt{increased})$
    * $(\mathsf{P}^*, \cdot, \cdot, \cdot, \cdot, \texttt{sent})$
    * $(\cdot, \mathsf{P}^*, \cdot, \cdot, \cdot, \texttt{sent})$
    * $(\mathsf{P}^*, \cdot, \cdot, \cdot, \cdot, \texttt{received})$
    * $(\cdot, \mathsf{P}^*, \cdot, \cdot, \cdot, \texttt{received})$
    * $(\mathsf{P}^*, \cdot, \cdot, \cdot, \cdot, \texttt{reversed})$
    * $(\cdot, \mathsf{P}^*, \cdot, \cdot, \cdot, \texttt{reversed})$

    where $\mathsf{P}^* = \mathsf{P}$.
  - Compute $\mathsf{f.b}$ and $\mathsf{p.b}$ of P based on the entries retrieved.
  - Given the retrieved TNX values from entries, retrieve $\overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}$ and $\overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)}$.
  - Compute $\pi^{\mathsf{A}}_{\mathrm{PoB}} \leftarrow \texttt{SimProve}(x^{\mathsf{A}}_{\mathrm{PoB}})$, and check that $\texttt{Verify}(x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}}) = 1$. If so, record the entry $(x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}})$.
  - Set
$$x^{\mathsf{A}}_{\mathrm{PoB}} \leftarrow (\mathsf{pk}_{\mathsf{acct}}, \mathsf{nul}, \mathsf{acct}^{\mathsf{cm}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}).$$
  - Set
$$\mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}} \leftarrow (x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)}).$$

- Emulating $\mathcal{F}_{\mathsf{ch}}$, leak $(\texttt{Send}, \mathsf{sid}, \mathsf{mid}, |\mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}}|)$ to the adversary $\mathcal{A}$ for a randomly chosen mid. Proceed as follows if $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid})$ is received from $\mathcal{A}$.
- Emulating $\mathcal{F}_{\mathsf{ch}}$, if A is malicious, submit $(\texttt{Received}, \mathsf{sid}, \mathsf{P}, \mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}})$ to the adversary.
- Send $(\texttt{Balance.Ok}, \mathsf{sid}, \gamma)$ to $\mathcal{F}_{\mathsf{DART}}$.

(ii) Malicious user P:

- Emulating $\mathcal{F}_{\mathsf{ch}}$, receive $(\texttt{Send}, \mathsf{sid}, \mathsf{A}, \mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}})$ upon calling $\mathcal{F}_{\mathsf{ch}}$ by the malicious user P.
- Emulating $\mathcal{F}^{\dagger}_{\mathsf{NIZK}}$, parse $\mathsf{tnx}^{\mathsf{A}}_{\mathrm{PoB}} = (x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(1)}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{(0)})$.
- Extract the witness $w^{\mathsf{A}}_{\mathrm{PoB}} = (\mathsf{sk}_{\mathsf{acct}}, \rho, s)$ by running $\texttt{Extract}(x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}})$. Check if the relation $\mathcal{R}_{\mathrm{PoB}}(x^{\mathsf{A}}_{\mathrm{PoB}}, w^{\mathsf{A}}_{\mathrm{PoB}})$ holds. If so, store $(x^{\mathsf{A}}_{\mathrm{PoB}}, \pi^{\mathsf{A}}_{\mathrm{PoB}})$; otherwise, ignore.
- Emulating an honest auditor A, execute the algorithm described in Figure 9 and abort if the algorithm aborts.
- Otherwise, parse
$$x^{\mathsf{A}}_{\mathrm{PoB}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{nul}, \mathsf{acct}^{\mathsf{cm}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}).$$
- Submit $(\texttt{Balance}, \mathsf{sid}, \mathsf{a.t})$ to $\mathcal{F}_{\mathsf{DART}}$ on behalf of the malicious user P.
- Upon receiving $(\texttt{Balance}, \mathsf{sid}, \gamma)$ from $\mathcal{F}_{\mathsf{DART}}$, send back $(\texttt{Balance.Ok}, \mathsf{sid}, \gamma)$ to $\mathcal{F}_{\mathsf{DART}}$ if emulating $\mathcal{F}_{\mathsf{ch}}$, $(\texttt{Ok}, \mathsf{sid}, \mathsf{mid})$ is received from the adversary $\mathcal{A}$,

Auditor operation.

- Upon receiving $(\texttt{Audit}, \mathsf{sid}, \eta)$ from $\mathcal{F}_{\mathsf{DART}}$, emulating $\mathcal{F}_{\mathsf{Ledger}}$, leak $(\texttt{Read}, \mathsf{sid}, \gamma)$ to the adversary $\mathcal{A}$ for a randomly chosen γ.
- Send $(\texttt{Audit.Ok}, \mathsf{sid}, \eta)$ to $\mathcal{F}_{\mathsf{DART}}$ if $(\texttt{Read.ok}, \mathsf{sid}, \gamma)$ is received from $\mathcal{A}$.

7.2 Security games

Indistinguishability games for Theorem 1 (sketch). Through a sequence of games, we show that the random variables $\mathsf{EXEC}_{\Pi_{\mathsf{DART}},\mathcal{A},\mathcal{Z}}$ and $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{DART}},\mathcal{S},\mathcal{Z}}$ are statistically close. We denote by $\Pr[\mathsf{Game}^{(i)}]$ the probability that the environment $\mathcal{Z}$ outputs 1 in $\mathsf{Game}^{(i)}$. Each game $\mathsf{Game}^{(i)}$ has its own $\mathcal{F}_{\mathsf{DART}}^{(i)}$ and $\mathcal{S}^{(i)}$. We start from the most leaky functionality $\mathcal{F}_{\mathsf{DART}}^{(0)}$ and the associated simulator $\mathcal{S}^{(0)}$ and gradually move toward the main functionality $\mathcal{F}_{\mathsf{DART}}$ and the simulator $\mathcal{S}$.

- $\underline{\mathsf{Game}^{(0)}}$: Initially, the most leaky functionality $\mathcal{F}_{\mathsf{DART}}^{(0)}$ forwards whatever it receives from the environment $\mathcal{Z}$ to the simulator $\mathcal{S}^{(0)}$. The simulator $\mathcal{S}^{(0)}$ corresponds to the real-world protocol execution $\mathsf{EXEC}_{\Pi_{\mathsf{DART}},\mathcal{A},\mathcal{Z}}$.
- $\underline{\mathsf{Game}^{(1)}}$: Same as $\mathsf{Game}^{(0)}$, except that $\mathcal{F}_{\mathsf{DART}}^{(1)}$ prevents the simulator $\mathcal{S}^{(1)}$ from submitting any message to $\mathcal{F}_{\mathsf{DART}}^{(1)}$ on behalf of the adversary $\mathcal{A}$ if $\mathcal{A}$ provides two different messages with the same corresponding commitment. This game is indistinguishable from $\mathsf{Game}^{(0)}$ due to the binding property of the underlying commitment scheme (Definition 9). Therefore,

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

- $\underline{\mathsf{Game}^{(2)}}$: Same as $\mathsf{Game}^{(1)}$, except that all ciphertexts generated by honest users, including $\mathsf{AUR}_{\mathsf{en}} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{en}}^{\mathsf{r}}}(\mathsf{pk}_{\mathsf{acct}}^{\mathsf{s}}, v, \mathsf{a.t})$ and $\psi = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{en}}^{\mathsf{A}}}(\mathsf{pk}_{\mathsf{acct}}^{\mathsf{s}}, \mathsf{pk}_{\mathsf{en}}^{\mathsf{r}}, v, \mathsf{a.t})$, are replaced with simulated ones. Particularly, the simulator $\mathcal{S}^{(2)}$ encrypts dummy values as plaintexts for $\mathsf{AUR}_{\mathsf{en}}$ and ψ. This game is indistinguishable from $\mathsf{Game}^{(1)}$ due to the IND-CPA security of the underlying encryption scheme (Definition 3). Therefore,

$$|\Pr[\mathsf{Game}^{(2)}] - \Pr[\mathsf{Game}^{(1)}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}.$$

- $\underline{\mathsf{Game}^{(3)}}$: Same as $\mathsf{Game}^{(2)}$, except that for all ciphertexts generated by the simulator, including $\mathsf{AUR}_{\mathsf{en}} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{en}}^{\mathsf{r}}}(\cdot,\cdot,\cdot)$ and $\psi = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{en}}^{\mathsf{A}}}(\cdot,\cdot,\cdot,\cdot)$, the receivers' and auditors' encryption public keys $\mathsf{pk}_{\mathsf{en}}^{\mathsf{r}}$ and $\mathsf{pk}_{\mathsf{en}}^{\mathsf{A}}$, respectively, are replaced with dummy public keys selected by $\mathcal{S}^{(3)}$. This game is indistinguishable from $\mathsf{Game}^{(2)}$ due to the key privacy of the underlying encryption scheme (Definition 4). Therefore,

$$|\Pr[\mathsf{Game}^{(3)}] - \Pr[\mathsf{Game}^{(2)}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{KeyPriv}}.$$

- $\underline{\mathsf{Game}^{(4)}}$: Same as $\mathsf{Game}^{(3)}$, except that all pseudorandom values generated by honest users, including nullifiers $\mathsf{nul} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho)$, are replaced by those simulated by $\mathcal{S}^{(4)}$ through sampling random values from the pseudorandom function (PRF) range. This game is indistinguishable from $\mathsf{Game}^{(3)}$ due to the pseudorandomness property of the underlying PRF scheme (Definition 12). Therefore,

$$|\Pr[\mathsf{Game}^{(4)}] - \Pr[\mathsf{Game}^{(3)}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF.Pseudo}}.$$

- $\underline{\mathsf{Game}^{(5)}}$: Same as $\mathsf{Game}^{(4)}$, except that $\mathcal{F}_{\mathsf{DART}}^{(5)} = \mathcal{F}_{\mathsf{DART}}$ prevents the simulator $\mathcal{S}^{(5)} = \mathcal{S}$ from submitting any message on behalf of the adversary $\mathcal{A}$ if $\mathcal{A}$ provides a valid membership proof π_{ACCU} for an element that is not part of the accumulator. This game is indistinguishable from $\mathsf{Game}^{(4)}$ due to the soundness property of the underlying accumulator scheme (Definition 16). Therefore,

$$|\Pr[\mathsf{Game}^{(5)}] - \Pr[\mathsf{Game}^{(4)}]| \leq \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Sound}}.$$

  This sequence of games starts from the real-world protocol execution $\mathsf{EXEC}_{\Pi_{\mathsf{DART}},\mathcal{A},\mathcal{Z}}$ and ends with the ideal-world execution $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{DART}},\mathcal{S},\mathcal{Z}}$. Given that each pair of consecutive games is computationally indistinguishable, the probability for any PPT environment $\mathcal{Z}$ to distinguish $\mathsf{EXEC}_{\Pi_{\mathsf{DART}},\mathcal{A},\mathcal{Z}}$ from $\mathsf{EXEC}_{\mathcal{F}_{\mathsf{DART}},\mathcal{S},\mathcal{Z}}$ is upper-bounded by

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind}} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{KeyPriv}} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF.Pseudo}} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Sound}}.$$

  This concludes the security proof.

□

Acknowledgements

Part of the work of the first author was supported by the Polymesh Association. The authors thank Adam Dossa and Robert Gabriel Jakabosky from the Polymesh Association for valuable discussions. This work was also supported by Input Output (iohk.io) through its funding of the University of Edinburgh Blockchain Technology Lab.

References

1. Almashaqbeh, G., Solomon, R.: Sok: Privacy-preserving computing in the blockchain era. In: 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P). pp. 124–139. IEEE (2022)
2. Baldimtsi, F., Chalkias, K.K., Madathil, V., Roy, A.: SoK: Privacy-preserving transactions in blockchains. Cryptology ePrint Archive, Paper 2024/1959 (2024), https://eprint.iacr.org/2024/1959
3. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 566–582. Springer (2001)
4. Benarroch, D., Gillespie, B., Lai, Y.T., Miller, A.: Sok: Programmable privacy in distributed systems. Cryptology ePrint Archive (2024)
5. Biçer, O., Tschudin, C.: Oblivious homomorphic encryption. Cryptology ePrint Archive (2023)
6. BNP Paribas: T+1 trade affirmation and settlement (2024), available at: Link
7. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Annual international cryptology conference. pp. 41–55. Springer (2004)

8. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
9. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA. pp. 315–334. IEEE Computer Society (2018). https://doi.org/10.1109/SP.2018.00020, https://doi.org/10.1109/SP.2018.00020
10. Buterin, V.: Ethereum white paper (2013), https://github.com/ethereum/wiki/wiki/White-Paper
11. Campanelli, M., Hall-Andersen, M.: Veksel: Simple, efficient, anonymous payments with large anonymity sets from well-studied assumptions. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. pp. 652–666 (2022)
12. Campanelli, M., Hall-Andersen, M., Kamp, S.H.: Curve trees: Practical and transparent zero-knowledge accumulators. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023. pp. 4391–4408. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/campanelli
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
14. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: Advances in Cryptology-CRYPTO 2006: 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006. Proceedings 26. pp. 78–96. Springer (2006)
15. Chatzigiannis, P., Chalkias, K.: Proof of assets in the diem blockchain. In: Applied Cryptography and Network Security Workshops: ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21–24, 2021, Proceedings. pp. 27–41. Springer (2021)
16. Chaum, D.: Blind signatures for untraceable payments. pp. 199–203 (1982)
17. Crites, E., Kiayias, A., Kohlweiss, M., Sarencheh, A.: SyRA: Sybil-resilient anonymous signatures with applications to decentralized identity. Cryptology ePrint Archive, Paper 2024/379 (2024), https://eprint.iacr.org/2024/379
18. Damgaard, I., Ganesh, C., Khoshakhlagh, H., Orlandi, C., Siniscalchi, L.: Balancing privacy and accountability in blockchain identity management. In: Paterson, K.G. (ed.) Topics in Cryptology - CT-RSA 2021 - Cryptographers' Track at the RSA Conference 2021, Virtual Event, May 17-20, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12704, pp. 552–576. Springer (2021). https://doi.org/10.1007/978-3-030-75539-3_23, https://doi.org/10.1007/978-3-030-75539-3_23
19. Diamond, B.E.: Many-out-of-many proofs and applications to anonymous zether. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1800–1817. IEEE (2021)
20. Dingledine, R., Mathewson, N., Syverson, P.F.: Tor: The second-generation onion router. pp. 303–320 (2004)
21. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) Public Key Cryptography - PKC 2005, 8th International

Workshop on Theory and Practice in Public Key Cryptography, Les Diablerets, Switzerland, January 23-26, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3386, pp. 416–431. Springer (2005). https://doi.org/10.1007/978-3-540-30580-4_28, https://doi.org/10.1007/978-3-540-30580-4_28
22. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: International Workshop on Public Key Cryptography. pp. 416–431. Springer (2005)
23. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE transactions on information theory **31**(4), 469–472 (1985)
24. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A new design for anonymous cryptocurrencies. In: Advances in Cryptology–ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part I 25. pp. 649–678. Springer (2019)
25. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986). https://doi.org/10.1007/3-540-47721-7_12, https://doi.org/10.1007/3-540-47721-7_12
26. Goldreich, O.: Towards a theory of software protection and simulation by oblivious rams. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing. pp. 182–194 (1987)
27. 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)
28. Guo, Y., Karthikeyan, H., Polychroniadou, A., Huussin, C.: Pride ct: Towards public consensus, private transactions, and forward secrecy in decentralized payments. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3904–3922. IEEE (2024)
29. Hansen, A.B., Nielsen, J.B., Simkin, M.: Ocash: Fully anonymous payments between blockchain light clients. Cryptology ePrint Archive (2024)
30. JAP: Jap anonymity & privacy (2009), http://anon.inf.tu-dresden.de/
31. Kerber, T., Kiayias, A., Kohlweiss, M., Zikas, V.: Ouroboros crypsinous: Privacy-preserving proof-of-stake. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 157–174. IEEE (2019)
32. Kosba, A.E., Zhao, Z., Miller, A., Qian, Y., Chan, T.H., Papamanthou, C., Pass, R., Shelat, A., Shi, E.: How to use snarks in universally composable protocols. IACR Cryptol. ePrint Arch. p. 1093 (2015), http://eprint.iacr.org/2015/1093
33. Kumar, A., Fischer, C., Tople, S., Saxena, P.: A traceability analysis of monero's blockchain. In: Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22. pp. 153–173. Springer (2017)
34. Madathil, V., Scafuro, A.: Prifhete: Achieving full-privacy in account-based cryptocurrencies is possible. Cryptology ePrint Archive (2023)
35. Meiklejohn, S., Orlandi, C.: Privacy-enhancing overlays in bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 127–141. Springer (2015)
36. Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. In: Proceedings of the 2013 conference on Internet measurement conference. pp. 127–140 (2013)

37. Metere, R., Dong, C.: Automated cryptographic analysis of the pedersen commitment scheme. In: Computer Network Security: 7th International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security, MMM-ACNS 2017, Warsaw, Poland, August 28-30, 2017, Proceedings 7. pp. 275–287. Springer (2017)
38. Micali, S., Rabin, M., Kilian, J.: Zero-knowledge sets. In: 44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings. pp. 80–91. IEEE (2003)
39. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: 2013 IEEE symposium on security and privacy. pp. 397–411. IEEE (2013)
40. Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., et al.: An empirical analysis of traceability in the monero blockchain. arXiv preprint arXiv:1704.04299 (2017)
41. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (October 31 2008), https://bitcoin.org/bitcoin.pdf
42. Narula, N., Vasquez, W., Virza, M.: zkledger: Privacy-preserving auditing for distributed ledgers. In: 15th USENIX symposium on networked systems design and implementation (NSDI 18). pp. 65–80 (2018)
43. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
44. Norton Rose Fulbright: SEC's Division of Trading and Markets Issues No-Action Letter. Published by Norton Rose Fulbright (October 2020), available at: Link
45. Payment Systems Regulator: Confirmation of payee - preventing app scams (2024), available at: Link
46. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
47. Sarencheh, A., Kiayias, A., Kohlweiss, M.: PEReDi: Privacy-enhanced, regulated and distributed central bank digital currencies. Cryptology ePrint Archive, Paper 2022/974 (2022), https://eprint.iacr.org/2022/974
48. Sarencheh, A., Kiayias, A., Kohlweiss, M.: PARScoin: A privacy-preserving, auditable, and regulation-friendly stablecoin. Cryptology ePrint Archive, Paper 2023/1908 (2023), https://eprint.iacr.org/2023/1908
49. Sasson, E.B., 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 (2014)
50. Setty, S.T.V.: Spartan: Efficient and general-purpose zksnarks without trusted setup. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III. Lecture Notes in Computer Science, vol. 12172, pp. 704–737. Springer (2020). https://doi.org/10.1007/978-3-030-56877-1_25, https://doi.org/10.1007/978-3-030-56877-1_25
51. Setty, S.T.V., Thaler, J., Wahby, R.S.: Unlocking the lookup singularity with lasso. In: Joye, M., Leander, G. (eds.) Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI. Lecture Notes in Computer Science, vol. 14656, pp. 180–209. Springer (2024). https://doi.org/10.1007/978-3-031-58751-1_7, https://doi.org/10.1007/978-3-031-58751-1_7

52. The National Law Review: Time to Do the Foxtrot: The Three-Step SEC Establishes for Improved Process in Settlement of Digital Asset Trades. Published in The National Law Review (September 2020), available at: Link
53. The Securities Transfer Association (STA): Guidelines (2023), available at: Link
54. UK Finance: Confirmation of payee guidance (2024), available at: Link
55. U.S. Securities and Exchange Commission: FINRA ATS Role in Settlement of Digital Asset Security Trades (September 2020), available at: Link
56. Wüst, K., Kostiainen, K., Delius, N., Capkun, S.: Platypus: A central bank digital currency with unlinkable transactions and privacy-preserving regulation. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2947–2960 (2022)

A Proof of balance: a generic solution

In Section 4.1.8 we proposed our first proof of balance (PoB) protocol that relies on asset specific auditor. In this section, we describe our second approach for PoB, which allows users to independently prove their finalized $\mathsf{f.b}$ and pending balances $\mathsf{p.b}$ directly to any verifier, without reliance on auditors.

Recall that in addition to $\mathsf{f.b}$, the user must also reveal their $\mathsf{p.b}$ to the PoB verifier. If $\mathsf{p.b} > 0$, this implies that there exists at least one transaction that is either in $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$ or $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$. In this case, the user needs to take further action by pointing to the transactions and justifying the value of $\mathsf{p.b}$. This is crucial from the balance verifying perspective because the transaction status determines whether the funds can still be reversed and, therefore, whether they could be potentially part of the user's balance or not. (i) if $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$, reversion is still possible, and (ii) if $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$, reversion is not possible. Therefore, in this case, the sender should point to transactions with $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$ and $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$, and prove that $\mathsf{p.b}$ equals the sum of the values of all transactions with $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$ plus the sum of the values of all transactions with $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$. This allows the PoB verifier to understand how much of $\mathsf{p.b}$ is irreversible and how much is reversible.

Therefore, in case $\mathsf{p.b} > 0$, the user needs to point to all finalized ($\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$) and pending transactions ($\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$). If the sender has many finalized transactions, pointing to all transactions becomes inefficient and also introduces information leakage. This is due to the ability of the PoB verifier to link every transaction in which the user has been the sender, which is undesirable as the PoB verifier needs to (only) verify the balance. To address this, the sender can scan the ledger for their transactions whose status has changed from the initial state of $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 0$ to $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$, and submit a specific transaction called pending-balance-update (pbu) transaction ($\mathsf{tnx}_{\mathsf{pbu}}$), to reduce $\mathsf{p.b}$, thereby moving the transaction status to $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 3$. Figure 11 illustrates the sender's transaction status $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} \in \{0, 1, 2, 3\}$ that is the extension of Figure 1. A transition occurs from $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$ to $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 3$ when the sender submits $\mathsf{tnx}_{\mathsf{pbu}}$.[20]

[20] Note that $\mathsf{tnx}_{\mathsf{pbu}}$ points to a transaction with $\mathsf{tnx}^{\mathsf{st}}_{\mathsf{Send}} = 1$, and any two transactions $\mathsf{tnx}_{\mathsf{pbu}}$ and $\mathsf{tnx}^{*}_{\mathsf{pbu}}$ are unlinkable, as they reference different transactions with

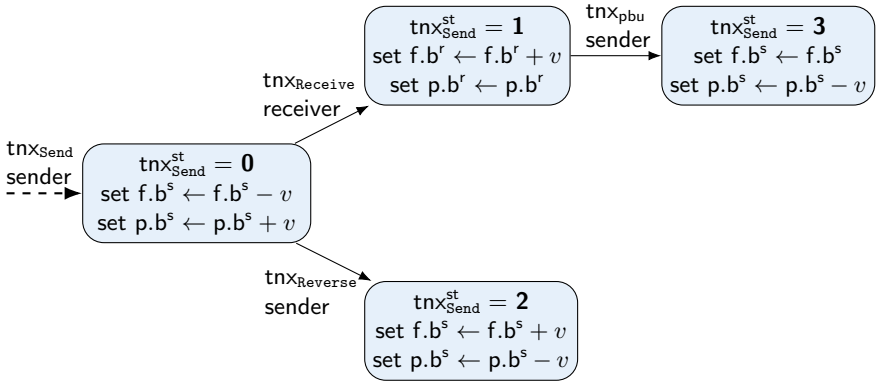

Fig. 11: Transaction status $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}}$ (extended).

The transition to $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 3$ avoids referencing finalized transactions (with $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$) for the PoB verifier and limits the pointer to only pending transactions (with $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 0$).
The pending balance $\mathsf{p.b}$ and $\mathsf{tnx}_{\mathsf{pbu}}$ transaction (which reduces $\mathsf{p.b}$) are only relevant for PoB. In other words, if a user knows that a specific asset type they hold will never be balance-verified, there is no need for the user to submit $\mathsf{tnx}_{\mathsf{pbu}}$ to reduce $\mathsf{p.b}$. The sole purpose of maintaining $\mathsf{p.b}$ in the user's account is to support PoB.
The user submits $\mathsf{tnx}_{\mathsf{pbu}}$ if applicable e.g., there exists a transaction with $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$. This can be done, in a single transaction, pointing to all finalized transactions and transitioning the account state by submitting a large $\mathsf{tnx}_{\mathsf{pbu}}$ (see aggregatable transactions in Section 5.1). This reduces their pending balance by the sum of the values of all finalized transactions in one step, offering better efficiency. However, this approach introduces linkability, as all finalized transactions are linked to each other through the single $\mathsf{tnx}_{\mathsf{pbu}}$. Alternatively, for better privacy, the user can submit separate $\mathsf{tnx}_{\mathsf{pbu}}$ transactions for each finalized transaction, reducing their pending balance sequentially. While this approach avoids linking finalized transactions, it comes at the cost of efficiency.
Note that pointing to transactions with $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 0$ still introduces privacy leakage, as the PoB verifier will link all these pending transactions. To mitigate this, the sender can reverse all pending transactions using the *Reversion* algorithm (Figure 8), reducing $\mathsf{p.b}$ and thus avoiding the need to link pending transactions. Reversing transactions, however, may be inefficient, as it would require reinitiating the transactions after completing the PoB protocol (assuming that the sender is still interested in completing these transactions).

$\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$ and user's account state transition occurs anonymously via the accumulator membership proof. In other words, the account state transition in $\mathsf{tnx}_{\mathsf{pbu}}$ provides unlinkability similar to all other transactions in DART.

As discussed earlier, the sender has two incentives for reversing the transaction, making it in the receiver's best interest to affirm the transaction and increase their account balance as soon as possible. Furthermore, the incorporation of an asset-specific auditor in DART allows auditors to construct a decrypted view of the ledger for all transactions involving a specific asset type. This includes identifying any pending AUR for a receiver who may either be potentially malicious or simply unaware of the incoming AUR due to not scanning the ledger prior to the PoB verification. Collaboration between the asset-specific auditor and the PoB verifier provides the necessary information to the verifier[21]. In the following, we present our second PoB approach, which operates independently of the auditor.

Upon receiving $(\texttt{Balance}, \mathsf{sid}, \mathsf{a.t})$ from the environment to prove the balance of the account associated with asset type $\mathsf{a.t}$, the user calls $\mathcal{F}_{\mathsf{Ledger}}$ to identify all transactions whose status has changed from $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 0$ to $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$. Then, the user submits $\mathsf{tnx}_{\mathsf{pbu}}$ to the ledger, where the user's account transitions to a new state, with the pending balance decreased by the value of the transaction whose status is $\mathsf{tnx}^{\mathsf{st}}_{\mathtt{Send}} = 1$. The user proves, in ZK, that they are indeed the creator/sender of the transaction. The submission of $\mathsf{tnx}_{\mathsf{pbu}}$ is done sequentially to avoid likability. The algorithm is provided in the Figure 12.

The NIZK witness, statement, and relation are defined as follows.

- $w_{\mathsf{pbu}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}_{\mathsf{old}}, \mathsf{p.b}_{\mathsf{old}}, \rho_{\mathsf{old}}, s_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}, \rho_{\mathsf{new}}, s_{\mathsf{new}}, \mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}, v, \mathsf{a.t}, \pi_{\mathsf{ACCU}})$
- $x_{\mathsf{pbu}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{AUR}_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}}_{\mathtt{Send}})$
- The relation $\mathcal{R}_{\mathsf{pbu}}(x_{\mathsf{pbu}}, w_{\mathsf{pbu}})$ is defined as:

$$
\begin{aligned}
\mathcal{R}_{\mathsf{pbu}} = \Big\{ & \mathsf{ACCU.VfyMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}, \pi_{\mathsf{ACCU}}) = 1 \\
& \wedge \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}} = \mathsf{Com}(\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}_{\mathsf{old}}, \mathsf{p.b}_{\mathsf{old}} - v, \mathsf{a.t}, \rho_{\mathsf{new}}; s_{\mathsf{new}}) \\
& \wedge \mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}} = \mathsf{Com}(\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}_{\mathsf{old}}, \mathsf{p.b}_{\mathsf{old}}, \mathsf{a.t}, \rho_{\mathsf{old}}; s_{\mathsf{old}}) \\
& \wedge \mathsf{nul}_{\mathsf{old}} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho_{\mathsf{old}}) \\
& \wedge (\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}}) \in \mathsf{Acc.KeyGen}(1^{\lambda}) \\
& \wedge \mathsf{AUR}_{\mathsf{en}} = \mathsf{Enc}_{\mathsf{pk}^{\mathsf{r}}_{\mathsf{en}}}(\mathsf{pk}_{\mathsf{acct}}, v, \mathsf{a.t}) \Big\}.
\end{aligned}
$$

$\mathsf{tnx}_{\mathsf{pbu}}$ Generation

▷ Upon receiving $(\texttt{Balance}, \mathsf{sid}, \mathsf{a.t})$ from $\mathcal{Z}$, parse $\mathsf{addr}_{\mathsf{pk}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{pk}_{\mathsf{en}})$, and $\mathsf{addr}_{\mathsf{sk}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{en}})$.

▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$. Upon receiving $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.

▷ Retrieve the recorded entry $(\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}, \mathsf{acct}_{\mathsf{old}})$ where $\mathsf{acct}_{\mathsf{old}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}_{\mathsf{old}}, \mathsf{p.b}_{\mathsf{old}}, \mathsf{a.t}, \rho_{\mathsf{old}}, s_{\mathsf{old}})$.

[21] Note that this differs from our first PoB approach (Section 4.1.8), in which the auditor selectively decrypts transactions explicitly pointed to by the user.

▷ Retrieve all entries $(\mathsf{AUR}^i, \mathsf{AUR}^i_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}})$ in $\mathsf{L}_{\mathsf{AUR}}$ where, upon parsing $\mathsf{AUR}^i = (\mathsf{pk}^{\mathsf{s}_i}_{\mathsf{acct}}, v^{(i)}, \mathsf{a.t}^i; \mathsf{pk}^{\mathsf{r}_i}_{\mathsf{en}})$, the conditions $\mathsf{a.t}^i = \mathsf{a.t}$ and $\mathsf{pk}^{\mathsf{s}_i}_{\mathsf{acct}} = \mathsf{pk}_{\mathsf{acct}}$ hold.

▷ For each such entry $(\mathsf{AUR}^i, \mathsf{AUR}^i_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}})$, if there is any transaction $\mathsf{TNX}^j \in \mathsf{state}'$ where $\mathsf{TNX}^j = (\mathsf{tnx}^j_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}})$, $\mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}} = 1$, and $\mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}} = \mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}}$, proceed as follows:

- parse $\mathsf{AUR}^j = (\mathsf{pk}_{\mathsf{acct}}, v^{(j)}, \mathsf{a.t}; \cdot)$.
- compute $\mathsf{nul}_{\mathsf{old}} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho_{\mathsf{old}})$.
- sample $\rho_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$, and $s_{\mathsf{new}} \xleftarrow{\$} \mathbb{Z}_q$.
- set $\mathsf{acct}_{\mathsf{new}} \leftarrow (\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}_{\mathsf{old}}, \mathsf{p.b}_{\mathsf{old}} - v^{(j)}, \mathsf{a.t}, \rho_{\mathsf{new}}, s_{\mathsf{new}})$.
- compute $\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}} \leftarrow \mathsf{Com}(\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}_{\mathsf{old}}, \mathsf{p.b}_{\mathsf{old}} - v^{(j)}, \mathsf{a.t}, \rho_{\mathsf{new}}; s_{\mathsf{new}})$.
- compute $\pi_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.PrvMem}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{state}', \mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}})$.
- call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, x_{\mathsf{pbu}}, w_{\mathsf{pbu}})$ and receive $(\mathtt{Proof}, \mathsf{sid}, \pi_{\mathsf{pbu}})$.
- set $\mathsf{tnx}_{\mathsf{pbu}} \leftarrow (x_{\mathsf{pbu}}, \pi_{\mathsf{pbu}})$, and call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Append}, \mathsf{sid}, \mathsf{tnx}_{\mathsf{pbu}})$.
- call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$; upon receiving back $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$, if $\mathsf{tnx}_{\mathsf{pbu}} \in \mathsf{state}'$ for $\mathsf{state}' \in \mathsf{state}$, set $(\mathsf{acct}^{\mathsf{cm}}_{\mathsf{old}}, \mathsf{acct}_{\mathsf{old}}) \leftarrow (\mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{acct}_{\mathsf{new}})$, and set: $\mathsf{L}_{\mathsf{AUR}} \leftarrow \mathsf{L}_{\mathsf{AUR}} \setminus \{(\mathsf{AUR}^i, \mathsf{AUR}^i_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}_i}_{\mathtt{Send}})\}$.
- else, ignore.

Proceed by executing the $\mathsf{tnx}_{\mathrm{PoB}}$ generation algorithm, Figure 13 (*note: there is no output to $\mathcal{Z}$ yet*).

$\mathsf{tnx}_{\mathsf{pbu}}$ Verification

▷ Execute $\textsc{Validate}_{\mathsf{pbu}}(\mathsf{state}, \mathsf{tnx}_{\mathsf{pbu}})$:

- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathsf{pbu}} = (x_{\mathsf{pbu}}, \pi_{\mathsf{pbu}})$, and $x_{\mathsf{pbu}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{AUR}^j_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}})$.
- given $\mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}} \in x_{\mathsf{pbu}}$, retrieve associated $\mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}} \in \mathsf{state}'$ and $\mathsf{AUR}^*_{\mathsf{en}} \in \mathsf{state}'$.
- ignore if any of the following conditions hold:
  - $\mathsf{AUR}^*_{\mathsf{en}} \neq \mathsf{AUR}^j_{\mathsf{en}}$
  - $\mathsf{nul}_{\mathsf{old}} \in \mathbb{L}_{\mathsf{NUL}}$
  - $\mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}} \neq 1$
  - upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x_{\mathsf{pbu}}, \pi_{\mathsf{pbu}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.
- else, output 1.

▷ Execute $\textsc{Update}_{\mathsf{pbu}}(\mathsf{state}, \mathsf{tnx}_{\mathsf{pbu}})$:

- parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$, $\mathsf{tnx}_{\mathsf{pbu}} = (x_{\mathsf{pbu}}, \pi_{\mathsf{pbu}})$, and $x_{\mathsf{pbu}} = (\mathsf{v}_{\mathsf{ACCU}}, \mathsf{nul}_{\mathsf{old}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}}, \mathsf{AUR}^j_{\mathsf{en}}, \mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}})$.
- given $\mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}} \in x_{\mathsf{pbu}}$, retrieve associated $\mathsf{TNX}^j = (\mathsf{tnx}^j_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{id}_j}_{\mathtt{Send}}, \mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}})$.
- set $\mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}} \leftarrow 3$, $\mathsf{TNX}^j \leftarrow (\cdot, \cdot, \mathsf{tnx}^{\mathsf{st}_j}_{\mathtt{Send}})$ and $\mathsf{state}' \leftarrow \mathsf{state}' \cup \{\mathsf{TNX}^j\}$.
- call $\mathsf{v}_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp}_{\mathsf{ACCU}}, \mathsf{v}_{\mathsf{ACCU}}, \mathsf{acct}^{\mathsf{cm}}_{\mathsf{new}})$.
- set $\mathbb{L}_{\mathsf{NUL}} \leftarrow \mathbb{L}_{\mathsf{NUL}} \cup \{\mathsf{nul}_{\mathsf{old}}\}$.
- set $\mathsf{state} \leftarrow \mathsf{state} \cup \{(\mathsf{state}', \cdot, \cdot, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})\}$ and output state.

Fig. 12: Pending balance update (pbu) transaction $\mathsf{tnx}_{\mathsf{pbu}}$

Finally, the user executes PoB generation algorithm described in Figure 13. The NIZK witness, statement, and relation are defined as follows.

- $w_{\mathrm{PoB}} = (\mathsf{sk}_{\mathsf{acct}}, \rho, s, \overrightarrow{\mathsf{AUR}}, \{\mathsf{pk}_{\mathsf{en}}^{\mathsf{r}_k}\}_k)$
- $x_{\mathrm{PoB}} = (\mathsf{nul}, \mathsf{acct}^{\mathsf{cm}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}, \overrightarrow{\mathsf{AUR}}_{\mathsf{en}}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{\mathsf{Send}})$
- The relation $\mathcal{R}_{\mathrm{PoB}}(x_{\mathrm{PoB}}, w_{\mathrm{PoB}})$ is defined as:

$$\begin{aligned}\mathcal{R}_{\mathrm{PoB}} = \Big\{ & \mathsf{acct}^{\mathsf{cm}} = \mathsf{Com}(\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}, \rho; s) \\ & \wedge \mathsf{nul} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho) \\ & \wedge \mathsf{AUR}_{\mathsf{en}}^{k} = \mathsf{Enc}_{\mathsf{pk}_{\mathsf{en}}^{r_k}}(\mathsf{AUR}^{k}) \quad \forall \mathsf{AUR}^{k} \in \overrightarrow{\mathsf{AUR}} \\ & \wedge \sum_{k} \mathsf{AUR}^{k}.v = \mathsf{p.b} \\ & \wedge (\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}}) \in \mathsf{Acc.KeyGen}(1^{\lambda}) \Big\}.\end{aligned}$$

$\mathsf{tnx}_{\mathbf{PoB}}$ Generation

▷ Execute steps 1 to 4 of the $\mathsf{tnx}_{\mathsf{pbu}}$ generation algorithm, Figure 12.
▷ For each $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_i}$: retrieve all $\mathsf{TNX}^j \in \mathsf{state}'$ values where $\mathsf{TNX}^j = (\mathsf{tnx}_{\mathsf{Send}}^{j}, \mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_j}, \mathsf{tnx}_{\mathsf{Send}}^{\mathsf{st}_j})$, $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{st}_j} = 0$, and $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_i} = \mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_j}$ hold.
▷ Let the aggregation of AUR^j, $\mathsf{AUR}_{\mathsf{en}}^{j}$, and $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_j}$ for all j be denoted as $\overrightarrow{\mathsf{AUR}}$, $\overrightarrow{\mathsf{AUR}}_{\mathsf{en}}$, and $\overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{\mathsf{Send}}$ respectively.
▷ Retrieve the recorded entry $(\mathsf{acct}^{\mathsf{cm}}, \mathsf{acct})$ where $\mathsf{acct} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}, \rho, s)$.
▷ Compute nullifier $\mathsf{nul} \leftarrow \mathsf{PRF}_{\mathsf{sk}_{\mathsf{acct}}}(\rho)$.
▷ Call $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Prove}, \mathsf{sid}, x_{\mathrm{PoB}}, w_{\mathrm{PoB}})$ and receive $(\mathtt{Proof}, \mathsf{sid}, \pi_{\mathrm{PoB}})$.
▷ Set PoB transaction $\mathsf{tnx}_{\mathrm{PoB}} \leftarrow (x_{\mathrm{PoB}}, \pi_{\mathrm{PoB}})$.
▷ Output $(\mathtt{Balance}, \mathsf{sid}, \mathsf{tnx}_{\mathrm{PoB}})$ to $\mathcal{Z}$.

$\mathsf{tnx}_{\mathbf{PoB}}$ Verification (off-chain executed by PoB verifier)

▷ Upon receiving $(\mathtt{VerifyPoB}, \mathsf{sid}, \mathsf{tnx}_{\mathrm{PoB}})$ from $\mathcal{Z}$ proceed as follows.
▷ Parse $\mathsf{tnx}_{\mathrm{PoB}} = (x_{\mathrm{PoB}}, \pi_{\mathrm{PoB}})$, and $x_{\mathrm{PoB}} = (\mathsf{nul}, \mathsf{acct}^{\mathsf{cm}}, \mathsf{pk}_{\mathsf{acct}}, \mathsf{f.b}, \mathsf{p.b}, \mathsf{a.t}, \overrightarrow{\mathsf{AUR}}_{\mathsf{en}}, \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{\mathsf{Send}})$.
▷ Call $\mathcal{F}_{\mathsf{Ledger}}$ with $(\mathtt{Read}, \mathsf{sid})$. Upon receiving $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ from $\mathcal{F}_{\mathsf{Ledger}}$, parse $\mathsf{state} = (\mathsf{state}', \mathbb{L}_{\mathsf{IAA}}, \mathbb{L}_{\mathsf{KA}}, \mathbb{L}_{\mathsf{NUL}}, \mathsf{v}_{\mathsf{ACCU}})$.
▷ For each j, $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_j} \in \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{\mathsf{Send}}$: ignore if any of the following conditions hold:
  - $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_j} \notin \mathsf{state}'$
  - $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_j} \neq 0$
  - given $\mathsf{tnx}_{\mathsf{Send}}^{\mathsf{id}_j} \in \overrightarrow{\mathsf{tnx}}^{\mathsf{id}}_{\mathsf{Send}}$, $\mathsf{AUR}_{\mathsf{en}}^{*j} \neq \mathsf{AUR}_{\mathsf{en}}^{j}$ where $\mathsf{AUR}_{\mathsf{en}}^{*j} \in \mathsf{state}'$ and $\mathsf{AUR}_{\mathsf{en}}^{j} \in \overrightarrow{\mathsf{AUR}}_{\mathsf{en}}$.
▷ Else, ignore if any of the following conditions hold:

- $\mathsf{nul} \in \mathbb{L}_{\mathsf{NUL}}$
- upon calling $\mathcal{F}_{\mathsf{NIZK}}$ with $(\mathtt{Verify}, \mathsf{sid}, x_{\mathrm{PoB}}, \pi_{\mathrm{PoB}})$, $(\mathtt{Vrfed}, \mathsf{sid}, 0)$ is returned.

▷ Else, output $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{a.t}, \mathsf{f.b}, \mathsf{p.b})$.

Fig. 13: Proof of balance transaction $\mathsf{tnx}_{\mathrm{PoB}}$ – 2nd approach

B Ideal functionalities

B.1 Key generation functionality

The ideal functionality $\mathcal{F}_{\mathsf{KeyReg}}$ specifies a public key encryption (PKE) and account key generations, parameterized by a PKE key generation algorithm $(\mathsf{pk}_{\mathsf{en}}, \mathsf{sk}_{\mathsf{en}}) \xleftarrow{\$} \mathsf{PKE.KeyGen}(1^\lambda)$ (Definition 1) and an account key pair $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}})$ derived from $\mathsf{Acc.KeyGen}(1^\lambda)$, where $\mathsf{sk}_{\mathsf{acct}} \xleftarrow{\$} \mathbb{Z}_q^*$ and $\mathsf{pk}_{\mathsf{acct}} \leftarrow g^{\mathsf{sk}_{\mathsf{acct}}}$ for a generator g of a cyclic group $\mathbb{G}$ of prime order q. The functionality supports three operations: (i) *key generation:* upon receiving a key generation request from a party P, it generates new PKE and account key pairs if none already exist for P, otherwise output already exist ones. Upon generation $\mathcal{F}_{\mathsf{KeyReg}}$ stores them as $\mathsf{addr}_{\mathsf{pk}} = (\mathsf{pk}_{\mathsf{acct}}, \mathsf{pk}_{\mathsf{en}})$ and $\mathsf{addr}_{\mathsf{sk}} = (\mathsf{sk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{en}})$ under P, and outputs them to P. (ii) *key retrieval:* upon receiving P^* from any network entities, $\mathcal{F}_{\mathsf{KeyReg}}$ returns the public key pair $\mathsf{addr}_{\mathsf{pk}}$ corresponding to P^* if it exists. (iii) *identifier retrieval:* upon receiving pk from any network entity P, $\mathcal{F}_{\mathsf{KeyReg}}$ identifies and returns the associated party P^* if pk matches either the PKE or account public key recorded for P^*.

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

- **Key generation.** Upon input $(\mathtt{Gen.Key}, \mathsf{sid})$ from some party P, if there already exists $(\mathsf{P}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$, output $(\mathtt{Gen.Key}, \mathsf{sid}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ to P and abort. Else, call $\mathsf{PKE.KeyGen}$, and $\mathsf{Acc.KeyGen}$ to generate $(\mathsf{pk}_{\mathsf{en}}, \mathsf{sk}_{\mathsf{en}})$ and $(\mathsf{pk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{acct}})$ respectively (re-call if they are not fresh). Set $\mathsf{addr}_{\mathsf{pk}} \leftarrow (\mathsf{pk}_{\mathsf{acct}}, \mathsf{pk}_{\mathsf{en}})$, and $\mathsf{addr}_{\mathsf{sk}} \leftarrow (\mathsf{sk}_{\mathsf{acct}}, \mathsf{sk}_{\mathsf{en}})$. Record $(\mathsf{P}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$. Output $(\mathtt{Gen.Key}, \mathsf{sid}, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ to P via private-delayed output.
- **Key retrieval.** Upon input $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{P}^*)$ from some party P, ignore if $(\mathsf{P}^*, \mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ is not recorded. Else, output $(\mathtt{Rtrv.Key}, \mathsf{sid}, \mathsf{P}^*, \mathsf{addr}_{\mathsf{pk}})$ to P.
- **Identifier retrieval.** Upon input $(\mathtt{Rtrv.id}, \mathsf{sid}, \mathsf{pk})$ from some party P, ignore if $(\mathsf{P}^*, (\mathsf{pk}, \cdot), \cdot)$ or $(\mathsf{P}^*, (\cdot, \mathsf{pk}), \cdot)$ is not recorded for some P^*. Else, output $(\mathtt{Rtrved.id}, \mathsf{sid}, \mathsf{P}^*)$ to P.

B.2 Non-interactive zero-knowledge functionality

The functionality $\mathcal{F}_{\mathsf{NIZK}}$ defines a *non-interactive zero-knowledge* (NIZK) proof system parameterized by a relation $\mathcal{R}(x, w)$, where x is a statement and w is a corresponding witness introduced by Groth et al. [27]. $\mathcal{F}_{\mathsf{NIZK}}$ consists of two main operations: *proof generation* and *proof verification*. In contrast to interactive zero-knowledge proofs, NIZK does not require the prior specification of the verifier. Consequently, the resulting proof can undergo verification by any party. In the *proof generation* phase, when a party U requests to generate a proof by sending (x, w), $\mathcal{F}_{\mathsf{NIZK}}$ first checks if w satisfies the relation $\mathcal{R}(x, w) = 1$. If the relation holds, it notifies the adversary $\mathcal{A}$ with x but not w. The adversary then sends back a proof π, which $\mathcal{F}_{\mathsf{NIZK}}$ stores together with x and returns π to U. In the *proof verification* phase, when a party U sends a request (x, π) to verify a proof, $\mathcal{F}_{\mathsf{NIZK}}$ checks if the pair (x, π) has already been stored. If so, it outputs a verification success. If not, it forwards the request to the adversary $\mathcal{A}$ and waits for a response in the form of a witness w. If $\mathcal{R}(x, w) = 1$, the proof is considered valid, and $\mathcal{F}_{\mathsf{NIZK}}$ stores the pair (x, π) and outputs verification success. This ensures that proofs can only be verified if valid witnesses exist, while maintaining zero-knowledge by hiding the witness during proof generation.

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

$\mathcal{F}_{\mathsf{NIZK}}$ is parameterized by a relation $\mathcal{R}$.

- **Proof generation:** On receiving $(\texttt{Prove}, \mathsf{sid}, x, w)$ from some party P, ignore if $\mathcal{R}(x, w) = 0$. Else, send $(\texttt{Prove}, \mathsf{sid}, x)$ to $\mathcal{A}$. Upon receiving $(\texttt{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{A}$, store (x, π) and send $(\texttt{Proof}, \mathsf{sid}, \pi)$ to P.
- **Proof verification:** Upon receiving $(\texttt{Verify}, \mathsf{sid}, x, \pi)$ from some party P, check whether (x, π) is stored. If not send $(\texttt{Verify}, \mathsf{sid}, x, \pi)$ to $\mathcal{A}$. Upon receiving the answer $(\texttt{Witness}, \mathsf{sid}, w)$ from $\mathcal{A}$, check $\mathcal{R}(x, w) = 1$ and if so, store (x, π). If (x, π) has been stored, output $(\texttt{Vrfed}, \mathsf{sid}, 1)$ to P, else output $(\texttt{Vrfed}, \mathsf{sid}, 0)$.

A concept closely related to NIZK is the notion of *signatures of knowledge* (SoK). SoK combines digital signatures with zero-knowledge proofs and allows a signer to authenticate a message not only by proving possession of a secret key but also by demonstrating knowledge of a witness w for an NP statement $x \in L$. Such a signature ensures that, upon verification, it is confirmed that the signer possesses a valid witness for the truth of the statement. The concept of SoK was formally defined by Chase et al. [14] via an ideal functionality $\mathcal{F}_{\mathsf{SoK}}$. Similar to $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{SoK}}$ is for $(x, w) \in \mathcal{R}$. It consists of three primary phases: *setup*, *signature generation*, and *signature verification*. Unlike $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{SoK}}$ receives $(\texttt{Verify}, \texttt{Sign}, \mathsf{SimSign}, \mathsf{Extract})$ from the adversary $\mathcal{A}$. Upon receiving a request $(\texttt{Sign}, \mathsf{sid}, m, x, w)$ from a party P, $\mathcal{F}_{\mathsf{SoK}}$ checks whether $(x, w) \in \mathcal{R}$. If the check passes, $\mathcal{F}_{\mathsf{SoK}}$ computes $\sigma \leftarrow \mathsf{SimSign}(m, x)$. Unlike $\mathcal{F}_{\mathsf{NIZK}}$, $\mathcal{F}_{\mathsf{SoK}}$ does

not leak the statement x to the adversary. To avoid leaking the statement to the adversary in the proof generation of proof of balance, we utilize $\mathcal{F}_{\mathsf{SoK}}$, which inherently prevents such leakage. Although $\mathcal{F}_{\mathsf{SoK}}$ introduces a message m as part of the signature, this component is unnecessary in our setting. Therefore, we simplify $\mathcal{F}_{\mathsf{SoK}}$ by fixing m to a predefined dummy message for all sessions. This transforms $\mathcal{F}_{\mathsf{SoK}}$ into a form that functions as $\mathcal{F}_{\mathsf{NIZK}}$ without the undesired statement leakage. Therefore, in the following we present the formal definition of $\mathcal{F}_{\mathsf{SoK}}$ from [14] under a simplifying assumption: the message m in all interfaces and algorithms is a predefined (dummy) message, hence, we avoid addressing it. Without loss of generality, we denote the following functionality as $\mathcal{F}^{\dagger}_{\mathsf{NIZK}}$.

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

The functionality is parameterized by a relation $\mathcal{R}$. `Prove`, `SimProve`, and `Extract` are descriptions of PPT TMs, and `Verify` is a description of a deterministic polytime TM.

- **Setup:** Upon receiving $(\texttt{Setup}, \mathsf{sid})$ from some party P, if this is the first time that $(\texttt{Setup}, \mathsf{sid})$ is received, send $(\texttt{Setup}, \mathsf{sid})$ to $\mathcal{A}$. Upon receiving $(\texttt{Algorithms}, \mathsf{sid}, \texttt{Prove}, \texttt{Verify}, \texttt{SimProve}, \texttt{Extract})$ from $\mathcal{A}$, store these algorithms. Output $(\texttt{Algorithms}, \mathsf{sid}, \texttt{Prove}, \texttt{Verify})$ to P.
- **Proof Generation:** Upon receiving $(\texttt{Prove}, \mathsf{sid}, x, w)$ from P, if $(x, w) \notin \mathcal{R}$, ignore. Else, compute $\pi \leftarrow \texttt{SimProve}(x)$, and check that $\texttt{Verify}(x, \pi) = 1$. If so, output $(\texttt{Proof}, \mathsf{sid}, \pi)$ to P and record the entry (x, π). Else, output an error message (*Completeness error*) to P and halt.
- **Proof Verification:** Upon receiving $(\texttt{Verify}, \mathsf{sid}, x, \pi)$ from P, check whether (x, π) is stored. If stored, output $(\texttt{Vrfed}, \mathsf{sid}, 1)$ to P. Else, let $w \leftarrow \texttt{Extract}(x, \pi)$. If $(x, w) \in \mathcal{R}$, output $(\texttt{Vrfed}, \mathsf{sid}, 1)$ to P. Else, if $\texttt{Verify}(x, \pi) = 0$, output $(\texttt{Vrfed}, \mathsf{sid}, 0)$ to P. Otherwise, output an error message (*Verification error*) to P and halt.

B.3 Ledger functionality

The ledger functionality $\mathcal{F}_{\mathsf{Ledger}}$ abstracts a public ledger and provides a global access to the ledger to all parties, parameterized by a VALIDATE predicate (to check transaction validity), an UPDATE function (to update the ledger state), and an initial state state (which is empty at the start). The functionality supports two main operations: *read* and *append*. In the *read* operation, a party U requests the current state of the ledger. Upon receiving this request, $\mathcal{F}_{\mathsf{Ledger}}$ informs the adversary $\mathcal{A}$ that a request for read is submitted. If the adversary permits the read, the current ledger state is returned to the requester. Otherwise, the request is ignored, ensuring that reads occur only when permitted by $\mathcal{A}$.

In the *append* operation, a party submits a transaction tnx to be added to the ledger. The functionality forwards this tnx to the adversary. If the adversary approves, the VALIDATE function checks whether the transaction tnx is valid according to the current state state. If valid, the UPDATE function updates the state by appending tnx to the ledger. Otherwise, the transaction is discarded. This model captures both transparency and adversarial control typical in blockchain systems, where reads and appends are subject to validation rules and external influence while ensuring consistent state updates. Depending on the type of transaction tnx, the functionality $\mathcal{F}_{\mathsf{Ledger}}$ invokes a sub-VALIDATE and sub-UPDATE algorithms tailored to different transaction types described in Figures 3 to 8 and 12.

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

The functionality is globally available to all parties and is parameterized by a predicate VALIDATE, and UPDATE function and a variable state where initially $\mathsf{state} \leftarrow \emptyset$.

- **Read:** Upon receiving $(\mathtt{Read}, \mathsf{sid})$ from a party P, generate a fresh γ and set $L(\gamma) \leftarrow \mathsf{P}$. Send $(\mathtt{Read}, \mathsf{sid}, \gamma)$ to $\mathcal{A}$. Upon receiving $(\mathtt{Read.ok}, \mathsf{sid}, \gamma)$ from $\mathcal{A}$, ignore if $L(\gamma) = \perp$. Else, return $(\mathtt{Read}, \mathsf{sid}, \mathsf{state})$ to P where $L(\gamma) = \mathsf{P}$. Set $L(\gamma) \leftarrow \perp$.
- **Append:** Upon receiving $(\mathtt{Append}, \mathsf{sid}, \mathsf{tnx})$ from a party P, send $(\mathtt{Append.req}, \mathsf{sid}, \mathsf{tnx})$ to $\mathcal{A}$. Upon receiving $(\mathtt{Append.req.Ok}, \mathsf{sid}, \mathsf{tnx})$ from $\mathcal{A}$, invoke VALIDATE$(\mathsf{state}, \mathsf{tnx})$, and if it outputs 1, update state via calling $\mathsf{state} \leftarrow$ UPDATE$(\mathsf{state}, \mathsf{tnx})$. Otherwise, ignore.

B.4 Communication channel functionality

As described in [17] despite robust cryptographic protections at the protocol level, "network leakage" can still reveal information about the communicating parties. Therefore, in our UC framework, maintaining a degree of sender anonymity is essential to preserve (UC) anonymity. This issue is not unique to Π_{DART} but arises in any UC-based formalization of privacy-preserving mechanisms. For example, Zerocash [49] would provide no anonymity without a sender-anonymous channel. Practical deployments of Π_{DART} could use a VPN, the Tor network [20], temporary network identities (e.g., public terminals), or JAP [30]. The functionality $\mathcal{F}_{\mathsf{ch}}$ defines a secure and anonymous communication channel between a sender P^{s} and a receiver P^{r}. When the sender P^{s} inputs a message m along with the receiver's identifier P^{r}, $\mathcal{F}_{\mathsf{ch}}$ records the sender and receiver identifiers, and message in an internal mapping associated with a randomly generated message ID, mid. It notifies the adversary $\mathcal{A}$ of mid and the message length. When the adversary approves, $\mathcal{F}_{\mathsf{ch}}$ retrieves the corresponding message and de-

livers it to the receiver, indicating the sender's identifier and the full message contents.

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

Let P^{s} be a sender and P^{r} be a receiver. The message is denoted by m.

- Upon input $(\mathtt{Send}, \mathsf{sid}, \mathsf{P}^{\mathsf{r}}, m)$ from P^{s}, record a mapping $\mathsf{P}(\mathsf{mid}) \leftarrow (\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, m)$ where mid is chosen at random. Output $(\mathtt{Send}, \mathsf{sid}, \mathsf{mid}, |m|)$ to $\mathcal{A}$.
- Upon receiving $(\mathtt{Ok}, \mathsf{sid}, \mathsf{mid})$ from $\mathcal{A}$, retrieve $\mathsf{P}(\mathsf{mid}) = (\mathsf{P}^{\mathsf{s}}, \mathsf{P}^{\mathsf{r}}, m)$ and send $(\mathtt{Received}, \mathsf{sid}, \mathsf{P}^{\mathsf{s}}, m)$ to P^{r} via private-delayed output.

C Cryptographic schemes

C.1 Public key encryption (PKE) schemes

Definition 1 (Public key encryption (PKE) scheme). *Let* $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *denote a public key encryption (PKE) scheme. A PKE scheme is defined by the following components:*

- $\mathsf{PKE.KeyGen}$ ***(Key generation):*** *This algorithm takes a security parameter* λ *as input and outputs a pair* $(\mathsf{pk}, \mathsf{sk})$*, where* pk *is the public key and* sk *is the secret (private) key. The public key* pk *also (implicitly) defines a corresponding message space* $\mathsf{MessageSpace}(\mathsf{pk})$ *which specifies the set of valid plaintexts for encryption.*
- $\mathsf{PKE.Enc}$ ***(Encryption):*** *This algorithm takes the public key* pk *and a plaintext message* m *(where* $m \in \mathsf{MessageSpace}(\mathsf{pk})$*) as input and produces a ciphertext* C*.*
- $\mathsf{PKE.Dec}$ ***(Decryption):*** *The decryption algorithm takes the private key* sk *and a ciphertext* C *as input and outputs either the original plaintext message* m *or a special failure symbol* $\perp$ *if decryption fails.*

Definition 2 (PKE correctness). *A public key encryption (PKE) scheme* $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is said to be correct if, for all* $\lambda \in \mathbb{N}$*, for all key pairs* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$*, and for all messages* $m \in \mathsf{MessageSpace}(\mathsf{pk})$*,* $\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pk}, m)) = m$ *holds with overwhelming probability.*

Definition 3 (PKE CPA security). *A public key encryption (PKE) scheme* $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is* $\mathsf{IND\text{-}CPA}$*-secure if, for all PPT adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda)$ *in the following experiment is negligible in* λ*:* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$*.*
The experiment, denoted $\mathsf{IND\text{-}CPA}^{b}_{\mathsf{PKE}}(\mathcal{A}, \lambda)$*, is defined between a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *and a challenger, with a security parameter* λ *and a bit* $b \in \{0, 1\}$*:*

1. *The challenger generates a key pair* $(\mathsf{pk}, \mathsf{sk})$ *by running* $\mathsf{PKE.KeyGen}(1^\lambda)$ *and sends the public key* pk *to the adversary* $\mathcal{A}$.
2. *The adversary* $\mathcal{A}$ *submits two plaintext messages* (m_0, m_1)*, ensuring that* $|m_0| = |m_1|$ *and* $m_0, m_1 \in \mathsf{MessageSpace}(\mathsf{pk})$.
3. *The challenger selects a random bit* $b \xleftarrow{\$} \{0,1\}$*, chooses one of the two plaintexts based on* b*, and computes the ciphertext* $C_b = \mathsf{PKE.Enc}(\mathsf{pk}, m_b)$.
4. *The challenger sends the ciphertext* C_b *to the adversary* $\mathcal{A}$.
5. *The adversary* $\mathcal{A}$ *outputs a bit* b' *as its guess for* b*. If* $\mathcal{A}$ *does not output any value, the guess* b' *is set to 0 by default.*

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

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

Definition 4 (PKE key privacy). *Key privacy ensures that the public key used to generate a ciphertext remains hidden [3]. The* KeyPriv *security experiment is conducted between a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *and a challenger. A public key encryption scheme* PKE *is* KeyPriv*-secure if, for all PPT adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}^{\mathsf{KeyPriv}}_{\mathcal{A}}(\lambda)$ *is negligible in the security parameter* λ*:* $\mathsf{Adv}^{\mathsf{KeyPriv}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$.
This experiment, denoted $\mathsf{KeyPriv}^b_{\mathsf{PKE}}(\mathcal{A}, \lambda)$*, is defined with a bit* $b \in \{0,1\}$ *and a security parameter* λ*:*

1. *The adversary is given access to two encryption-decryption oracles associated with different public keys.*
2. *For* $k \in \{0,1\}$*:* $(\mathsf{pk}_k, \mathsf{sk}_k) \xleftarrow{\$} \mathsf{PKE.KeyGen}(1^\lambda)$*, where* pk_k *and* sk_k *are the public and secret keys for the* k*-th oracle, respectively.*
3. *The adversary submits a message* m*, where:* $m \leftarrow \mathcal{A}^{O_{\mathsf{sk}_0,\mathsf{pk}_0}, O_{\mathsf{sk}_1,\mathsf{pk}_1}}(\mathsf{pk}_0, \mathsf{pk}_1)$.
4. *The challenger selects a random bit* $b \xleftarrow{\$} \{0,1\}$.
5. *A ciphertext* C *is generated by encrypting* m *under the public key* pk_b*:* $C \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_b, m)$.
6. *The adversary receives* C *and outputs a bit* b' *as its guess for* b*:* $b' \leftarrow \mathcal{A}^{O_{\mathsf{sk}_0,\mathsf{pk}_0}, O_{\mathsf{sk}_1,\mathsf{pk}_1}}(\mathsf{pk}_0, \mathsf{pk}_1, C)$.
7. *The adversary wins if* $b = b'$*. The oracles reject decryption queries for the challenge ciphertext* C*.*

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

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

Definition 5 (ElGamal encryption scheme). *The ElGamal encryption scheme [23], denoted* EG*, is a public-key encryption (PKE) scheme that operates over a prime-order cyclic group* $\mathbb{G}_q$ *with generator* g*. The scheme consists of three main algorithms* $\mathsf{EG} = (\mathsf{EG.KeyGen}, \mathsf{EG.Enc}, \mathsf{EG.Dec})$ *defined as follows:*

- $\mathsf{EG.KeyGen}$ ***(Key generation):*** *Let* $\mathcal{G}$ *be a prime-order group generator that outputs* (q, g)*, where* q *is a prime and* g *is a generator of the cyclic group* $\mathbb{G}_q$*. The key generation algorithm proceeds as follows:*

  - *Take the security parameter* 1^λ *as input.*
  - *Sample the secret key* $\mathsf{sk} = x \xleftarrow{\$} \mathbb{Z}_q$ *uniformly at random.*
  - *Compute the public key component* $\mathsf{P} = g^x$.
  - *The public key is* $\mathsf{pk} = (q, g, \mathsf{P})$, *and the secret key is* $\mathsf{sk} = x$.
- EG.Enc ***(Encryption):*** *To encrypt a message* $M \in \mathbb{G}_q$, *the encryption process is as follows:*
  - *Sample a random value* $r \xleftarrow{\$} \mathbb{Z}_q$.
  - *Compute:* $\mathsf{C}_1 = g^r, \quad \mathsf{B} = \mathsf{P}^r$.
  - *Compute the ciphertext components:* $\psi = (\mathsf{C}_1, \mathsf{C}_2), \quad$ *where* $\mathsf{C}_2 = \mathsf{B} \cdot M$.
- EG.Dec ***(Decryption):*** *Given a ciphertext* $\psi = (\mathsf{C}_1, \mathsf{C}_2)$, *the decryption process is as follows:*
  - *Compute:* $\mathsf{B} = \mathsf{C}_1^x$
  - *Recover the original message* M *by computing:* $M = \mathsf{C}_2 \cdot \mathsf{B}^{-1}$.

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

C.2 Commitment schemes

Definition 6 (Commitment scheme). *Let* $\mathsf{COM} = (\mathsf{COM.KeyGen}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ *denote a commitment scheme. A commitment scheme is defined by the following components:*

- COM.KeyGen ***(Key generation):*** *This probabilistic algorithm takes a security parameter* λ *as input and outputs a pair of keys* $(\mathsf{ck}, \mathsf{vk})$, *where* ck *is the commitment key used by the committer and* vk *is the verification key used by the verifier. In the case of a publicly verifiable scheme,* $\mathsf{ck} = \mathsf{vk}$.
- COM.Commit ***(Commitment phase):*** *This probabilistic algorithm takes the commitment key* ck *and a message* m *(where* $m \in \mathsf{MessageSpace}(\mathsf{ck})$*) as input. It outputs a commitment value* c *and an opening value* d*:* $(c, d) \leftarrow \mathsf{COM.Commit}(\mathsf{ck}, m)$. *The pair* (c, d) *allows the committer to convince the verifier that the committed message is* m *during the verification phase.*
- COM.Verify ***(Verification phase):*** *This deterministic algorithm takes the verification key* vk, *a message* m, *and an opening value* d *as input. It outputs* accept *if* (c, d) *correctly reconstructs the message* m *and satisfies the commitment's consistency conditions; otherwise, it outputs* reject. *More formally:*

$$\mathsf{COM.Verify}(\mathsf{vk}, c, m, d) = \begin{cases} \mathsf{accept} & \text{if } (c, d) \text{ is a valid opening for } m, \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

Definition 7 (Commitment correctness). *The correctness property ensures that, assuming honest behavior by both the committer and the verifier, a valid commitment for any (valid) message* m *is always accepted by the verifier with the probability 1.*

Definition 8 (Commitment hiding). *Let* $\mathsf{COM} = (\mathsf{COM.KeyGen}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ *denote a (publicly verifiable) commitment scheme.* ***Hiding*** *security, for all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*, is defined as follows. The hiding experiment* $\mathsf{Hide}^{b}_{\mathsf{COM}}(\mathcal{A}, \lambda)$ *between a PPT adversary* $\mathcal{A}$ *and a challenger proceeds as follows:*

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

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

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

A commitment scheme satisfies the ***hiding security*** *property if:*

- ***Perfect hiding:*** *For all adversaries* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) = 0$
- ***Computational hiding:*** *For all PPT adversaries* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{Hide}}_{\mathcal{A}}(\lambda) \leq \mathsf{negl}(\lambda)$

Intuitively, this guarantees that no adversary can distinguish the commitment of m_0 *from* m_1 *with probability significantly better than a random guess. Hence, the commitment hides the underlying message.*

Definition 9 (Commitment binding). *Let* $\mathsf{COM} = (\mathsf{COM.KeyGen}, \mathsf{COM.Commit}, \mathsf{COM.Verify})$ *denote a (publicly verifiable) commitment scheme. The* ***binding*** *security for all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$ *is defined as follows. The binding experiment* $\mathsf{Bind}_{\mathsf{COM}}(\mathcal{A}, \lambda)$ *between a PPT adversary* $\mathcal{A}$ *and a challenger proceeds as follows:*

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

$$\mathsf{COM.Verify}(\mathsf{vk}, m, c, d) = 1 \quad \textit{and} \quad \mathsf{COM.Verify}(\mathsf{vk}, m', c, d') = 1.$$

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

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

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

A commitment scheme satisfies the ***binding security*** *property if:*

- ***Perfect binding:*** *For all adversaries* $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind}}(\lambda) = 0$.
- ***Computational binding:*** *For all PPT adversaries* $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Bind}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Intuitively, this guarantees that no adversary can produce a valid commitment that can be opened to two different messages with probability significantly better than negligible. Hence, the commitment binds the committer to a single value.

Definition 10 (Pedersen commitment scheme). *Let* $\mathsf{PCOM} = (\mathsf{PCOM.KeyGen}, \mathsf{PCOM.Commit}, \mathsf{PCOM.Verify})$ *denote the Pedersen commitment scheme [46]. The Pedersen commitment scheme is defined by the following components:*

- $\mathsf{PCOM.KeyGen}$ ***(Key generation):*** *This probabilistic algorithm takes a security parameter* λ *as input and outputs the cyclic group parameters* (G, q, g)*, where* G *is a cyclic group of prime order* q*, and* g *is a generator of* G*. Additionally, it selects a random group element* $h \xleftarrow{\$} G$ *and outputs* $\mathsf{ck} = \mathsf{vk} = (G, q, g, h)$.
- $\mathsf{PCOM.Commit}$ ***(Commitment):*** *This probabilistic algorithm takes the commitment key* $\mathsf{ck} = (G, q, g, h)$ *and a message* $m \in \mathbb{Z}_q$ *as input. It generates a random opening value* $d \xleftarrow{\$} \mathbb{Z}_q$ *and computes the commitment value* $c = g^d \cdot h^m$*. The output is the pair* (c, d)*:* $(c, d) \leftarrow \mathsf{PCOM.Commit}(\mathsf{ck}, m)$.
- $\mathsf{PCOM.Verify}$ ***(Verification):*** *This deterministic algorithm takes the verification key* $\mathsf{vk} = (G, q, g, h)$*, a message* $m \in \mathbb{Z}_q$*, an opening value* $d \in \mathbb{Z}_q$*, and a commitment value* $c \in G$ *as input. The algorithm outputs* accept *if the equality* $g^d \cdot h^m = c$ *holds, indicating a valid commitment to* m*, and* reject *otherwise:* $\mathsf{PCOM.Verify}(\mathsf{vk}, c, m, d) = \begin{cases} \mathsf{accept} & \text{if } g^d h^m = c, \\ \mathsf{reject} & \text{otherwise.} \end{cases}$

The Pedersen commitment scheme PCOM *is perfectly hiding and computationally binding [37, 38].*

C.3 Pseudorandom function (PRF) schemes

Definition 11 (Pseudorandom function (PRF) scheme). *Let* $\mathsf{PRF} = (\mathsf{F.KeyGen}, \mathsf{F.Eval})$ *denote a pseudorandom function (PRF) scheme. A PRF scheme is defined by the following components:*

- $\mathsf{F.KeyGen}$ ***(Key generation):*** *This algorithm takes a security parameter* λ *as input and outputs a secret key* $k \in K$*, where* K *denotes the key space.*
- $\mathsf{F.Eval}$ ***(Function evaluation):*** *This algorithm takes the secret key* k *and an input* $x \in X$*, where* X *is the domain, and outputs a value* $y \in Y$*, where* Y *is the range. Formally, the function is defined as* $F_k : X \to Y$*, where* $F_k(x) = y$.

Definition 12 (PRF pseudorandomness). *A pseudorandom function (PRF) scheme* $\mathsf{PRF} = (\mathsf{F.KeyGen}, \mathsf{F.Eval})$ *is pseudorandom if, for all probabilistic polynomial-time (PPT) adversaries* $\mathcal{A}$*, the advantage* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF.Pseudo}}(\lambda)$ *in the following experiment is negligible in* λ*:* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF.Pseudo}}(\lambda) \leq \mathsf{negl}(\lambda)$.

The security experiment, denoted $\mathsf{Pseudo}^{b}_{\mathsf{PRF}}(\mathcal{A}, \lambda)$ *for* $b \in \{0,1\}$*, is defined between a probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$ *and a challenger as follows:*

1. *The challenger takes the security parameter* λ *and generates a secret key* $k \in K$ *by running* $\mathsf{F.KeyGen}(1^{\lambda})$.
2. *For* $b = 0$*: The challenger defines the function* $F_k : X \to Y$ *as* $F_k(x) = \mathsf{F.Eval}(k, x)$*, where* X *is the domain and* Y *is the range.*
3. *For* $b = 1$*: The challenger chooses a truly random function* $f : X \to Y$ *uniformly at random.*
4. *The challenger selects a random bit* $b \xleftarrow{\$} \{0,1\}$.
5. *The adversary* $\mathcal{A}$ *submits* q *queries* $x_1, \ldots, x_q \in X$*. For each query* x_i*, the challenger responds with* $f(x_i)$ *(if* $b = 1$*) or* $F_k(x_i)$ *(if* $b = 0$*).*
6. *The adversary* $\mathcal{A}$ *outputs a bit* $b' \in \{0,1\}$*, indicating its guess of whether the function is pseudorandom (*$b = 0$*) or truly random (*$b = 1$*).*

The adversary's advantage $\mathsf{Adv}^{\mathsf{PRF.Pseudo}}_{\mathcal{A}}(\lambda)$ *is defined as:*

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

Definition 13 (Dodis-Yampolskiy PRF). *Let* $\mathsf{DY} = (\mathsf{DY.KeyGen}, \mathsf{DY.Eval})$ *denote the Dodis-Yampolskiy pseudorandom function (DY PRF) scheme [22], which is based on the Boneh-Boyen unpredictable function [7]. The DY PRF is defined as follows:*

- $\mathsf{DY.KeyGen}$ ***(Key generation):*** *This algorithm takes a security parameter* λ *as input and outputs a secret key* $k \in \mathbb{Z}_q$*, where* q *is the order of a cyclic group* $\mathbb{G} = \langle g \rangle$.
- $\mathsf{DY.Eval}$ ***(Function evaluation):*** *This algorithm takes the secret key* $k \in \mathbb{Z}_q$ *and an input* x *and outputs a value* $y \in \mathbb{G}$*. The evaluation function is defined as:*

$$F_k(x) = g^{1/(k+x)},$$

where g *is a generator of the cyclic group* $\mathbb{G}$.

The security of the Dodis-Yampolskiy PRF relies on the hardness of the q-Decisional Diffie-Hellman Inversion (q-DDHI) problem [22].

C.4 Accumulator schemes

An accumulator scheme enables the compact representation of an arbitrarily large set S of elements. Given a membership witness for an accumulated element x and the current accumulator value $\mathsf{v}_{\mathsf{ACCU}}$, it is possible to efficiently verify whether x is a valid member of the accumulated set. Importantly, the accumulated set can be dynamically updated as elements are added.

Definition 14 (Accumulator scheme). *Let* $\mathsf{ACCU} = (\mathsf{ACCU.Setup}, \mathsf{Accu}, \mathsf{ACCU.Add}, \mathsf{ACCU.PrvMem}, \mathsf{ACCU.VfyMem})$ *denote an accumulator scheme over the universe* $\mathcal{U}$*. An accumulator scheme is defined by the following components:*

- $\mathsf{ACCU.Setup}$ ***(Setup):*** *This algorithm takes the security parameter λ as input and outputs the public parameters $\mathsf{pp_{ACCU}}$:* $\mathsf{pp_{ACCU}} \leftarrow \mathsf{ACCU.Setup}(1^\lambda)$.
- Accu ***(Accumulate):*** *This deterministic algorithm takes the public parameters $\mathsf{pp_{ACCU}}$ and a set $S \subseteq \mathcal{U}$ as input and computes an accumulator value $\mathsf{v_{ACCU}}$ for the set S:* $\mathsf{v_{ACCU}} \leftarrow \mathsf{Accu}(\mathsf{pp_{ACCU}}, S)$.
- $\mathsf{ACCU.Add}$ ***(Add element):*** *This algorithm takes the public parameters $\mathsf{pp_{ACCU}}$, an accumulator value $\mathsf{v_{ACCU}}$, and an element $x \in \mathcal{U}$ as input and outputs an updated accumulator value $\mathsf{v'_{ACCU}}$:* $\mathsf{v'_{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp_{ACCU}}, \mathsf{v_{ACCU}}, x)$.
- $\mathsf{ACCU.PrvMem}$ ***(Prove membership):*** *This algorithm takes the public parameters $\mathsf{pp_{ACCU}}$, a set S, and an element x as input and outputs a membership proof π_{ACCU} showing that x is in the accumulated set S:* $\pi_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.PrvMem}(\mathsf{pp_{ACCU}}, S, x)$.
- $\mathsf{ACCU.VfyMem}$ ***(Verify membership):*** *This algorithm takes the public parameters $\mathsf{pp_{ACCU}}$, an accumulator value $\mathsf{v_{ACCU}}$, an element x, and a membership proof π_{ACCU} as input and outputs a bit $\{0,1\}$ indicating whether x belongs to the set accumulated in $\mathsf{v_{ACCU}}$:* $\{0,1\} \leftarrow \mathsf{ACCU.VfyMem}(\mathsf{pp_{ACCU}}, \mathsf{v_{ACCU}}, x, \pi_{\mathsf{ACCU}})$.

Definition 15 (Accumulator correctness). *Let* $\mathsf{ACCU} = (\mathsf{ACCU.Setup}, \mathsf{Accu}, \mathsf{ACCU.Add}, \mathsf{ACCU.PrvMem}, \mathsf{ACCU.VfyMem})$ *denote an accumulator scheme over the universe $\mathcal{U}$. An accumulator scheme is said to be correct if the following properties hold, for all security parameters $\lambda \in \mathbb{N}$, all elements $x \in \mathcal{U}$, and all sets $S \subseteq \mathcal{U}$:*

- *let* $\mathsf{pp_{ACCU}} \leftarrow \mathsf{ACCU.Setup}(1^\lambda)$ *and* $\mathsf{v_{ACCU}} \leftarrow \mathsf{Accu}(\mathsf{pp_{ACCU}}, S)$. *Then, for the updated accumulator* $\mathsf{v'_{ACCU}} \leftarrow \mathsf{ACCU.Add}(\mathsf{pp_{ACCU}}, \mathsf{v_{ACCU}}, x)$, *we have:* $\mathsf{v'_{ACCU}} = \mathsf{Accu}(\mathsf{pp_{ACCU}}, S \cup \{x\})$.
- *for $x \in S$, let* $\mathsf{pp_{ACCU}} \leftarrow \mathsf{ACCU.Setup}(1^\lambda)$, $\mathsf{v_{ACCU}} \leftarrow \mathsf{Accu}(\mathsf{pp_{ACCU}}, S)$, *and* $\pi_{\mathsf{ACCU}} \leftarrow \mathsf{ACCU.PrvMem}(\mathsf{pp_{ACCU}}, S, x)$. *Then the membership proof π_{ACCU} is valid, i.e.,* $\mathsf{ACCU.VfyMem}(\mathsf{pp_{ACCU}}, \mathsf{v_{ACCU}}, x, \pi_{\mathsf{ACCU}}) = 1$.

Definition 16 (Accumulator soundness). *Soundness guarantees that no efficient adversary can choose a set S and generate a proof that verifies against* $\mathsf{Accu}(\mathsf{pp_{ACCU}}, S)$ *for an element $x \notin S$. More formally, an accumulator* ACCU *is said to be sound if the following property holds for all security parameters $\lambda \in \mathbb{N}$, and PPT adversaries $\mathcal{A}$:*

$$\Pr\begin{bmatrix}\mathsf{pp_{ACCU}} \leftarrow \mathsf{ACCU.Setup}(1^\lambda) \\ (S, x, \pi_{\mathsf{ACCU}}) \leftarrow \mathcal{A}(\mathsf{pp_{ACCU}})\end{bmatrix} : \begin{array}{l} x \notin S \ \wedge \ \mathsf{v_{ACCU}} \leftarrow \mathsf{Accu}(\mathsf{pp_{ACCU}}, S) \ \wedge \\ \mathsf{ACCU.VfyMem}(\mathsf{pp_{ACCU}}, \mathsf{v_{ACCU}}, x, \pi_{\mathsf{ACCU}}) = 1 \end{array}\Bigg] \leq \mathsf{negl}(\lambda)$$