AQQUA: Augmenting Quisquis with Auditability

George Papadoulis[1], Danai Balla[12], Panagiotis Grontas[1], and Aris Pagourtzis[12]

[1] National Technical University of Athens
[2] Archimedes/Athena RC
geopapadoulis@gmail.com, balla.danai@gmail.com, pgrontas@corelab.ntua.gr, pagour@cs.ntua.gr

Abstract. We propose AQQUA: a digital payment system that combines auditability and privacy. AQQUA extends Quisquis by adding two authorities; one for registration and one for auditing. These authorities do not intervene in the everyday transaction processing; as a consequence, the decentralized nature of the cryptocurrency is not disturbed. Our construction is account-based. An account consists of an updatable public key which functions as a cryptographically unlinkable pseudonym, and of commitments to the balance, the total amount of coins spent, and the total amount of coins received. In order to participate in the system a user creates an initial account with the registration authority. To protect their privacy, whenever the user wants to transact they create unlinkable new accounts by updating their public key and the total number of accounts they own (maintained in committed form). The audit authority may request an audit at will. The user must prove in zero-knowledge that all their accounts are compliant to specific policies. We formally define a security model capturing the properties that a private and auditable digital payment system should possess and we analyze the security of AQQUA under this model.

Keywords: digital payment systems, cryptocurrencies, privacy, auditability, updatable public keys

1 Introduction

Privacy vs Auditability: Addressing this dilemma becomes crucial, as blockchain technology advances and decentralized digital payment systems (DPS) evolve and gain in popularity. This prominence of DPS brings about integration with the heavily regulated traditional financial systems. A major question that must be answered is to what extent this can be achieved without sacrificing privacy and decentralization.

All flavors of DPS have some built-in support for privacy and regulation, even if it is rudimentary. Starting with Bitcoin [16], all DPS share the common feature of depending on a globally distributed, append-only, public ledger to document monetary transactions in a transparent, verifiable and immutable manner. The underlying consensus mechanism used to settle exchange history and introduce new transactions, along with the security properties of the cryptographic primitives employed, makes sure that these systems adhere to some (simple) rules. Further auditing can be achieved by merely inspecting this ledger, as everything is in the clear. To protect their privacy, users rely on the use of renewable pseudonyms to obscure their identities (but not the amounts exchanged). It has been shown, though, that by combining publicly available data from the blockchain in a smart way [15], anyone could link the pseudoidentities of the users and even uncover their real-world identities.

To overcome this problem, privacy-enhanced cryptocurrencies (e.g. Zerocash [2], Monero [18], Quisquis [10]) arose. These systems hide transaction identities and amounts exchanged, thus providing privacy in a provable cryptographic manner. At the same time, however, they allow malicious users to conduct illegal activities (e.g. money laundering, unauthorized money transition, tax evasion). This misuse of privacy has led to the need for a compromise, i.e. the creation of protocols that combine user privacy and auditability. Such auditable privacy solutions [11, 7, 13, 14, 17] aim to guarantee that both the system and its participants comply with financial regulations and laws, preventing them from engaging in illicit activities without being accountable to the authorities. Financial regulations that are usually supported in such schemes are KYC (Know-Your-Customer), Anti Money Laundering (AML), as well as restrictions to the number or the value of transactions a single user can make, or the total value that can be exchanged in a single transaction.

Our proposal. We propose AQQUA: a system to equip DPS with auditability, without changing its decentralized, permissionless, and trustless nature. AQQUA extends the QuisQuis [10] DPS with mechanisms to allow the auditing of users, while preserving privacy and confidentiality of transactions.

AQQUA introduces two new entities: A Registration Authority (RA), whose purpose is to enroll users into the system, and an Audit Authority (AA), whose purpose is to perform audits to users. In order to transact in AQQUA, users must first register to the RA and provide their real-world credentials, thus fulfilling KYC. They then acquire a cryptographic pseudonym, a unique initial public key, which can be used to create new accounts within AQQUA.

Contrary to other private and auditable DPS that either restrict the number of accounts a user can own within the system or trade anonymity for auditability, AQQUA enables users to own as many accounts as they wish within the system, while preserving anonymity and confidentiality of transactions and allowing auditing of users. This is achieved by splitting the state into two sets: The UTXOSet, which is similar to the state of QuisQuis and contains user accounts, and a UserSet that maintains a mapping between registered public keys and commitments to the number of accounts they own. The addition of a new public key to the UserSet can happen only after the approval of the RA, however the RA cannot censor or identify user transactions after their enrollment.

After the enrollment of a user in the system and in order to ensure anonymous participation, anyone can create new accounts for them that are provably unlinkable to their registered public key. This is achieved by utilizing *updatable public keys*, introduced in [10], which allow the creation of new, provably indistinguishable and independent public keys, from an initial key pair, without changing the underlying secret counterpart. While each user can create accounts on their own, the commitment to the number of accounts correspoding to their initial public key in the UserSet must be updated.

AQQUA accounts consist of commitments (for confidentiality) for the balance, the total amount of coins spent and the total amount of coins received in the corresponding updatable public key. In AQQUA, transactions can be thought of as 'wealth redistribution' between inputs and outputs, an idea originating from Quisquis [10]. Input accounts include the senders, the recipients as well as an anonymity set. Output accounts are new, updated but unlikable accounts for the senders, recipients, and decoys. To counter theft prevention, the sender proves in zero-knowledge that they have correctly updated the accounts and have not taken coins away from anyone except themselves.

Finally, the audit is executed by the AA asynchronously on the initial public key of each user. During auditing, each user should prove in zero-knowledge that for a specified period of time all of the their accounts are compliant to the system's policies, using data that are only stored on-chain. Penalties for non-compliance can then be enforced to misbehaved users.

2 Related Work

The proposed solutions in the literature that aim to combine privacy and transparency, can be examined from various angles depending on their approach to compliance.

A first criterion to sort the various approaches is the extent of control that the regulating authority is allowed to exercise. It must be noted, that in regulated cryptocurrencies the existence of some point of concentration cannot be entirely avoided, since there must be some mapping between cryptocurrency accounts and real-world identities. Some solutions [13, 20] try to limit their power by distributing its functionalities to different parties and using secure multi-party computation techniques to apply the regulation policies.

Regulation can be embedded into the consensus layer, allowing only compliant transactions to become part of the blockchain or they can be external by auditing public transactions at specific intervals [5]. The former approach is called *accountability*, while the latter *auditability* [5], and can also include some countermeasures for the offending parties. Accountability is better suited to permissioned blockchains, since the additional checks implied by regulation will place more burden on standalone miners in permissionless systems thus lowering their throughput even more. Other schemes such as [20] have the unique approach of allowing users a privacy budget to spend in order to satisfy KYC policies.

Some regulation is combined with points of concentration on the decentralized blockchains such as privacy mixers [3], since they know the actual senders and receivers of transactions. For privacy-preserving mixers, recent compliant solutions aim to allow transactions only from approved senders or disallow transactions from blacklisted participants [9]. Exchanges are also natural candidates for regulation, since they maintain the mapping between real-world identities and account identifiers and as a result they can report to the authorities aggregated statistics on the behavior of their users [14].

Another aspect on which the various approaches differ is on whether they prioritize privacy over regulation or vice versa. In the former case regulation is usually added as an addon to an already privacy preserving cryptocurrency. For instance, one of the first such works [11] adds policies for auditability to Zerocash [2]-like systems, by extending their coin format to accommodate a counter that is used when aggregating transactions for auditing. These counters are bound to real-world identities and are incremented for each transaction, and enable the system to enforce policies, such as spending limits, taxes and even allow the tracing of users.

PGC [7] provides a generalized design and an implementation of a scheme that combines privacy with auditability, leaning towards the latter. Their proposed schemes support a rich set of regulation policies to limit money laundering and enable taxation.

As far as usage is concerned, some proposals are aimed towards blockchains, or (better in this case) distributed ledgers whose users are large organizations (e.g. banks, or exchanges) [8, 17] that usually aggregate assets from many participants, while others pose no such restrictions and are aimed to regular blockchain users.

AQQUA combines the anonymity of Quisquis [10] with the policy expressiveness and regulation of [7]. While there are two centralized entities, the RA and the AA, their intervention is minimal as they cannot censor transactions. AQQUA does not offer traceability, but suffers from anonymity loss only between the system snapshots when auditing takes place. From the next transaction after auditing and onwards anonymity is restored. AQQUA can be used by simple users and large organizations alike. Thus we can claim that AQQUA is one of the most privacy - preserving proposals in the literature.

3 Preliminaries

3.1 Notation

We denote by λ the security parameter. We denote by $\mathcal{M}$ the message space and by $\mathcal{R}$ the randomness space of our cryptographic schemes. $\mathcal{V} = \{0, \dots, V\}$ is the set that defines the range of valid currency values, where V is an upper bound on the maximum possible number of coins in the system ($|\mathcal{V}| \lll |\mathcal{M}|$). When an element x is sampled uniformly at random from a set $\mathcal{X}$, we write $x \leftarrow\$ \mathcal{X}$. Given a tuple $t = (a, b)$ we refer to its parts using the dot notation, i.e. $t.a$ or $t.b$. We denote (a^x, b^x) as $t^x = (a, b)^x$.

3.2 Updatable Public Keys

We utilize the Updatable Public Key (UPK) primitive from [10] to implement accounts. The concept of an UPK scheme is that public keys can be updated while remaining indistinguishable from freshly generated keys. A UPK scheme is a tuple of algorithms $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Update}, \mathsf{VerifyKP}, \mathsf{VerifyUpdate})$.

- Setup generates the public parameters, which are implicitly given as input to all other algorithms, i.e. $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$. For instance, pp could be a prime-order group $(\mathbb{G}, g, p)$.
- KGen generates a keypair $(\mathsf{pk}, \mathsf{sk})$. Concretely, it is implemented as: Sample $r, \mathsf{sk} \leftarrow\$ \mathbb{F}_p$, calculate $\mathsf{pk} = (g^r, g^{r\dot{}\mathsf{sk}})$ and output $(\mathsf{sk}, \mathsf{pk})$.
- Update takes as input a set of public keys $\{\mathsf{pk}\}_{i=1}^n$ and a secret key and generates a new set $\{\mathsf{pk}'\}_{i=1}^n$ where $\mathsf{pk}'_i = \mathsf{pk}_i^r = (g_i^r, g_i^{r\cdot\mathsf{sk}})$ for all i.
- $\mathsf{VerifyKP}$ takes as input a keypair $(\mathsf{sk}, \mathsf{pk})$ and checks if it is valid, i.e. if pk corresponds to sk. It is constructed by parsing $\mathsf{pk} = (g', h')$ and outputting the result of the check $(g')^{\mathsf{sk}} \overset{?}{=} h'$.
- $\mathsf{VerifyUpdate}$ takes as input a pair of public keys and some randomness $(\mathsf{pk}', \mathsf{pk}, r)$ and checks if pk' is a valid update of pk using r. This is done by checking if $\mathsf{Update}(\mathsf{pk}; r) \overset{?}{=} \mathsf{pk}'$.

An UPK scheme must satisfy the following properties, formally defined in [10]:

- **Correctness**: All honestly generated keys verify correctly, the update process can be verified and the updated keys also verify successfully.
- **Indistinguishability**, meaning that an adversary cannot distinguish between a freshly generated public key and an updated version of a public key it already knows.
- **Unforgeability**, meaning that for every honestly generated keypair an adversary cannot learn the secret key of an updated public key without knowing the secret key of the original public key.

If the DDH assumption holds in $(\mathbb{G}, g, p)$ then this construction satisfies correctness, indistinguishability and unforgeability [10].

3.3 Commitments

We use a commitment scheme Commit relative to a public key pk that, given a message $m \in \mathcal{M}$ and randomness $r \in \mathcal{R}$, computes $\boxed{m} \leftarrow \mathsf{Commit}(\mathsf{pk}, m; r)$. Our commitments must satisfy the following properties:

- **Computational hiding**: An adversary has negligible advantage in distinguishing between $\mathsf{Commit}(\mathsf{pk}, m_0; r_0)$ and $\mathsf{Commit}(\mathsf{pk}, m_1; r_1)$, where $r_0, r_1 \leftarrow\$ \mathcal{R}$.
- **Unconditional binding**: A commitment cannot be opened to two different messages, even with the knowledge of the secret key sk.
- **Additively homomorphic**: For given operation $\odot$ it holds that $\mathsf{Commit}(\mathsf{pk}, m; r) \odot \mathsf{Commit}(\mathsf{pk}, m'; r') = \mathsf{Commit}(\mathsf{pk}, m + m'; r + r')$.
- **Key-anonymous**: An adversary cannot distinguish between $(m, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{Commit}(\mathsf{pk}_0, m))$ and $(m, \mathsf{pk}_0, \mathsf{pk}_1, \mathsf{Commit}(\mathsf{pk}_1, m))$ for any honestly generated public keys $\mathsf{pk}_0, \mathsf{pk}_1$ and adversarially chosen message m.

We construct such a scheme using the unconditionally binding commitments of [10]. They are defined in a prime-order p group $(\mathbb{G}, g, p)$ generated by g, where the DDH problem is hard. In essence, ElGamal 'encryption' is used in the exponent where the public keys are of the form $\mathsf{pk} = (g, h) \in \mathbb{G}^2$. Specifically, $\mathsf{Commit}(\mathsf{pk}, m; r)$ yields $\boxed{m} = (c, d)$, where $c = g_i^r$ and $d = g^m h_i^r$.

Using UKPs as the commitment public keys, one can verify and open commitments using the secret key, without needing to know the randomness used.

- $\mathsf{VerifyCom}(\mathsf{sk}, \mathsf{pk}, \mathtt{com}, m)$: Checks if $\mathtt{com} = (c, d)$ is a commitment to m under pk, by checking if $d = g^m c^{\mathsf{sk}}$ holds.
- $\mathsf{OpenCom}(\mathsf{sk}, \boxed{m})$: Given $\boxed{m} = (c, d)$, calculates m by calculating $dc^{-\mathsf{sk}}$ and brute-forcing to obtain m.

3.4 *Σ*-protocols

Let $\mathcal{R}$ be a binary relation for instances x and witnesses w, and let $\mathcal{L}$ be its corresponding language, i.e. $\mathcal{L} = \{x | \exists w : (x, w) \in \mathcal{R}\}$. A Σ-protocol for $\mathcal{R}$ is a three-move public-coin protocol between two PPT algorithms P, V, whose transcript consists of the following phases: (1) **Commit**: P commits to an initial message a and sends it to V (2) **Challenge**: V sends a challenge c to P (3) **Response**: P responds to the challenge with message z.

A Σ-protocol must satisfy the following properties:

- **Completeness**: if $x \in \mathcal{L}$, then if P acts according to the protocol, V always accepts the transcript.
- **Special Soundness**: given two transcripts with the same commitment and different challenges $(a, c, z), (a, c', z')$ one can efficient compute w such that $(x, w) \in \mathcal{R}$.
- **Special honest-verifier zero-knowledge (SHVZK)**: there exists a PPT simulator Sim that on input $x \in L$ and a honestly generated verifier's challenge c, outputs an accepting transcript of the form (a, c, z) with the same probability distribution as a transcript between honest P, V on input x.

AQQUA utilizes the following Σ-protocols: proof of knowledge of discrete logarithm [19], proof of knowledge of DDH tuple [6], Bayer-Groth shuffle [1], and Bulletproofs [4] for range proofs.

Additionally we utilize the following Σ-protocols defined in [10] and repeated below for convenience:

- Σ_{vu}: proof a valid update. Prover shows knowledge of w such that $\mathsf{pk}' = pk^w$.

$$\begin{array}{|lcl|}
\hline
\textbf{Prover}(\mathsf{pk}, \mathsf{pk}', w) & & \textbf{Verifier}(\mathsf{pk}, \mathsf{pk}') \\
s \leftarrow\!\!\$\ \mathbb{F}_p & & \\
\alpha \leftarrow \mathsf{pk}^s = (g^s, h^s) & \xrightarrow{\alpha} & \\
 & \xleftarrow{c} & c \leftarrow\!\!\$\ \{0,1\}^\kappa \\
z \leftarrow cw + s & \xrightarrow{z} & \\
 & & \text{Check } \mathsf{pk}^z = (\mathsf{pk}')^c \cdot \alpha \\
\hline
\end{array}$$

- Σ_{com} : proof of knowledge of two commitments of the same value v under different public keys. Prover shows knowledge of $w = (v, r_1, r_2)$ such that $\mathtt{com}_1 = \mathsf{Commit}(\mathsf{pk}_1, v; r_1), \mathtt{com}_2 = \mathsf{Commit}(\mathsf{pk}_2, v; r_2)$.

$$\begin{array}{|lcl|}
\hline
 & \mathsf{pk}_1 = (g_1, h_1), \mathtt{com}_1 = (c_1, d_1) & \\
 & \mathsf{pk}_2 = (g_2, h_2), \mathtt{com}_2 = (c_2, d_2) & \\
\textbf{Prover}(v, r_1, r_2) & & \textbf{Verifier} \\
v', r_1', r_2' \leftarrow\!\!\$\ \mathbb{F}_p & & \\
(e_1, f_1) \leftarrow (g_1^{r_1'}, g^{v'} h_1^{r_1'}) & & \\
(e_2, f_2) \leftarrow (g_2^{r_2'}, g^{v'} h_2^{r_2'}) & \xrightarrow{\alpha} & \\
 & \xleftarrow{x} & x \leftarrow\!\!\$\ \{0,1\}^\kappa \\
(z_v, z_{r1}, z_{r_2}) \leftarrow x(v, r_1, r_2) + (v', r_1', r_2') & \xrightarrow{(z_v, z_{r1}, z_{r_2})} & \\
 & & \text{Check for } i = 1, 2: \\
 & & g_i^{z_r i} = c_i^x \cdot e_1 \\
 & & g^{z_v} h_i^{z_r i} = d_i^x \cdot f_i \\
\hline
\end{array}$$

4 Definition of Auditable Private Decentralized Payment System

4.1 Entities

- *Registration Authority* ($\mathtt{RA}$): The role of the $\mathtt{RA}$ is to enroll new users into the system. Users register by sending their real-world identity information together with an initial public key that they create on their own. The $\mathtt{RA}$ stores this information off-chain. All the accounts that transact on behalf of this real-world user will originate from this initial public key, through the mechanism of subsection 3.2. The purpose of the registration procedure is essential as it establishes a link between a user's public key and their real identity, to be used for the potential penalization of non-compliant users.

- *Audit Authority* ($\mathtt{AA}$): Its role is to initiate the audit procedure in order to verify that users comply with the system's policies. If a user of the system is found to be non-compliant, the $\mathtt{AA}$ will collaborate with the $\mathtt{RA}$ to enforce the relative penalties.
- *Users* ($\mathtt{U}$): Users of the system that transact with each other.

4.2 State

In an auditable decentralized payment system, the state (denoted state) should at the very least contain the following two sets:

- $\mathsf{UTXOSet}$: A table containing the 'unspent' accounts, i.e. the accounts that are recorded as outputs of a valid transaction, but have not (yet) been used as inputs.
- $\mathsf{UserSet}$: A table containing information about the accounts a user maintains in the system. This table is part of the public state and will be employed by the $\mathtt{AA}$ as a way to connect their privately maintained information about the real identity of the user with their used accounts. The challenge in designing auditable and private payment systems is to maintain privacy despite the existence of this mapping. In our proposal the $\mathsf{UserSet}$ is composed of the user's initial public key and a commitment to the number of accounts owned by the user.

4.3 Functionalities

An auditable private decentralized payment system is a tuple of polynomial-time algorithms defined as below:

- $(\mathsf{state}_0, \mathsf{pp}) \leftarrow \mathsf{Setup}(\lambda)$
  Generates the initial state of the system state_0 and the public parameters pp, which are implicitly given as input to all other algorithms.
- $(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}, \pi) \leftarrow \mathsf{Register}()$
  Used by a user to create the registration information $\mathsf{userInfo}$ and their first account acct.
- $0/1 \leftarrow \mathsf{VerifyRegister}(\mathsf{userInfo}, \mathsf{acct}, \pi, \mathsf{state})$
  Used by the Registration Authority to verify the registration information and the account of a user.
- $\mathsf{state}' \leftarrow \mathsf{ApplyRegister}(\mathsf{userInfo}, \mathsf{acct}, \mathsf{state})$
  Used by the Registration Authority to add a user to the system after their successful registration.
- $\mathsf{tx} = (\{\mathsf{acct}\}_{i=1}^{n}, \{\mathsf{acct}'\}_{i=1}^{n}, \pi) \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \vec{\mathtt{v}}_{\mathtt{S}}, \vec{\mathtt{v}}_{\mathtt{R}}, \mathtt{A})$
  Used by the sender with secret key sk to create a transaction that redistributes their coins from their accounts in $\mathtt{S}$ among the recipients accounts in $\mathtt{R}$. The vectors $\vec{\mathtt{v}}_{\mathtt{S}}, \vec{\mathtt{v}}_{\mathtt{R}}$ describe the changes in the values in $\mathtt{S}, \mathtt{R}$ respectively. To hide the participating accounts, an anonymity set $\mathtt{A}$ is passed as input.
- $\mathsf{tx}_{\mathsf{CA}} = (\mathsf{acct}, \{\mathsf{userInfo}_i\}_{i=1}^{n}, \{\mathsf{userInfo}_i'\}_{i=1}^{n}, \pi) \leftarrow \mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$
  Creates a transaction to create a new account for the owner of $\mathsf{userInfo}.\mathtt{pk}_0$ and appropriately updates the value of the commitment to the number of accounts they own, $\mathsf{userInfo}.\mathtt{com}_{\mathtt{\#accs}}$. To hide the link between the newly created account acct and the corresponding $\mathtt{pk}_0$, an anonymity set $\mathtt{A}$ is given.
  $\mathsf{tx}_{\mathsf{DA}} = (\{\mathsf{acct}\}_{i=1}^{n}, \{\mathsf{acct}'\}_{i=1}^{n}, \{\mathsf{userInfo}\}_{i=1}^{n}, \{\mathsf{userInfo}'\}_{i=1}^{n} \pi) \leftarrow \mathsf{DeleteAcct}(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}_{\mathsf{D}}, \mathsf{acct}_{\mathsf{C}}, \mathtt{A}_1, \mathtt{A}_2)$:
  Delete a zero-balance account $\mathsf{acct}_{\mathsf{D}}$ from the UTXO set from owner of sk, and adding its auditing info $(\mathtt{out}, \mathtt{in})$ to another account $\mathsf{acct}_{\mathsf{C}}$ that shares the same sk. Anonymity sets $\mathtt{A}_1, \mathtt{A}_2$ are included to hide $\mathsf{acct}_{\mathsf{C}}$ and $\mathsf{userInfo}$, respectively.
- $0/1 \leftarrow \mathsf{VerifyTrans}(\mathsf{tx}, \mathsf{state})$
  It is a public verification algorithm that checks the validity of a transaction tx given the current state and outputs 1 if and only if it is valid.
- $\mathsf{state}' \leftarrow \mathsf{ApplyTrans}(\mathsf{tx}, \mathsf{state})$
  Used to apply to the current state a transaction tx, after its verification.
- $\mathsf{auditInfo} = (\pi, \mathtt{\#accs}_1, \{\mathsf{acct}_{1i}\}_{i=1}^{\mathtt{\#accs}_1}, \mathtt{\#accs}_2, \{\mathsf{acct}_{2i}\}_{i=1}^{\mathtt{\#accs}_2}) \leftarrow \mathsf{PrepareAudit}(\mathsf{sk}, \mathtt{pk}_0 \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}))$:
  Used by a user with secret key sk and initial public key $\mathtt{pk}_0$ to generate a proof π for being compliant with policy f, concerning a specific period of time defined by two blockchain snapshots $\mathsf{state}_1, \mathsf{state}_2$. The $\mathtt{aux}$ variable contains the auxiliary information needed for the policy.
- $0/1 \leftarrow \mathsf{VerifyAudit}(\mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}), \mathsf{auditInfo})$
  Used by the Audit Authority to check if the user with initial public key $\mathtt{pk}_0$ is compliant with policy f.

4.4 Policies

An auditable DPS should support a rich set of compliance policies. They can be captured as predicates over an initial public key $\mathtt{pk}_0$, a time period represented by a starting state state_1 and an ending state state_2, and auxiliary information $\mathtt{aux}$ which is dependent on an specific compliance goal. In all the policy predicates, we use the notation A_1, A_2 to denote the set of accounts in $\mathsf{state}_1.\mathsf{UTXOSet}, \mathsf{state}_2.\mathsf{UTXOSet}$ that are owned by the owner of $\mathtt{pk}_0$.

- Sending Limit policy f_{slimit}: The total amount a real-world user can send within a specific period. It can be determined by the $\mathtt{AA}$ off-chain and announced to the user for a specific period, depending on the application. The $\mathsf{state}_1, \mathsf{state}_2$ are the states of the blockchain at the beginning and end of the period, respectively.

$$f_{\mathsf{slimit}}(\mathsf{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), \mathsf{a}_{max}) = 1 \iff \left\{ \left(\sum_{\mathsf{acct} \in A_2} \mathtt{out} - \sum_{\mathsf{acct} \in A_1} \mathtt{out} \right) \leq \mathsf{a}_{max} \right\}$$

where $\mathtt{out}$ is the total amount an account has sent.

- Receiving Limit policy f_{rlimit}: Similarly, the total amount a 'physical' user can receive from other accounts.

$$f_{\mathsf{rlimit}}(\mathsf{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), \mathsf{a}_{max}) = 1 \iff \left\{ \left(\sum_{\mathsf{acct} \in A_2} \mathtt{in} - \sum_{\mathsf{acct} \in A_1} \mathtt{in} \right) \leq \mathsf{a}_{max} \right\}$$

where $\mathtt{in}$ is the total amount an account has received.

- Open policy f_{open}: The value of the amount sent or received by a user in a transaction.

$$f_{\mathsf{open}}(\mathsf{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), v_{\mathsf{open}}) = 1 \iff \left\{ \begin{array}{l} \left(v = \left(\sum_{\mathsf{acct} \in A_2} \mathtt{bl} - \sum_{\mathsf{acct} \in A_1} \mathtt{bl} \right) \in \mathcal{V} \right. \\ \left. \vee\, v = \left(\sum_{\mathsf{acct} \in A_1} \mathtt{bl} - \sum_{\mathsf{acct} \in A_2} \mathtt{bl} \right) \in \mathcal{V} \right) \\ \wedge\, v = v_{\mathsf{open}} \end{array} \right\}$$

where $\mathtt{bl}$ is the balance of an acct.

- Transaction Value Limit f_{txlimit}: Upper bound to the total transferred amount that can be sent in a transaction.

$$f_{\mathsf{txlimit}}(\mathsf{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), v_{\max}) = 1 \iff \left\{ v = \left(\sum_{\mathsf{acct} \in A_1} \mathtt{bl} - \sum_{\mathsf{acct} \in A_2} \mathtt{bl} \right) \leq v_{\max} \right\}$$

- Non-participation f_{np}: Non-participation in a specific transaction $\mathtt{tx}$ or inactivity of the user for a time period. The states $\mathsf{state}_1, \mathsf{state}_2$ are the states before and after a transaction is applied or at the beginning and end of the period.

$$f_{\mathsf{np}}(\mathsf{pk}_0, (\mathsf{state}_1, \mathsf{state}_2)) = 1 \iff \left\{ \begin{array}{l} \wedge \left(\sum_{\mathsf{acct} \in A_1} \mathtt{out} - \sum_{\mathsf{acct} \in A_2} \mathtt{out} \right) = 0 \\ \wedge \left(\sum_{\mathsf{acct} \in A_1} \mathtt{in} - \sum_{\mathsf{acct} \in A_2} \mathtt{in} \right) = 0 \end{array} \right\}$$

5 Security Model

An anonymous payment system should provide *anonymity* and *theft prevention.* Anonymity requires that an observer of the system cannot find the identities of senders and the receivers of a transaction if they don't own the sender's private key, and that even the recipient of a transaction cannot know the sender. Theft prevention means that users can only move funds from accounts they own. For the definitions of the anonymity and theft prevention properties, we adapt the definitions of Quisquis for the corresponding properties to AQQUA. Additionally, an auditable payment system requires the security property of *audit soundness*, which means that there cannot be a successfully verified audit generated by a user who is non-compliant.

We formally define these properties, using security games where the adversary has access to the following oracles.

- $\mathsf{sk} \leftarrow \mathsf{OCorrupt}(\mathsf{pk}, \mathsf{state})$: Returns the secret key that corresponds to a public key. The public key should belong either in an account or a user information entry of the state.
- $\mathsf{state} \leftarrow \mathsf{ORegister}()$: Creates a keypair and registers the public key. Returns the new state.
- $(\mathsf{tx}_{\mathsf{CA}}, \mathsf{state}) \leftarrow \mathsf{OCreateAcct}(\mathsf{userInfo}, \mathtt{A})$: Creates a new account for a $\mathsf{userInfo}$ entry using the anonymity set $\mathtt{A}$. Returns the corresponding transaction and resulting state after the transaction application.
- $(\mathsf{tx}_{\mathsf{DA}}, \mathsf{state}) \leftarrow \mathsf{ODeleteAcct}(\mathsf{userInfo}, \mathsf{acct}_{\mathsf{C}}, \mathsf{acct}_{\mathsf{D}}, \mathtt{A}_1, \mathtt{A}_2)$: Creates and applies a transaction to delete an account by calling $\mathsf{DeleteAcct}$. Returns the transaction and the resulting state after the transaction application.

- $(\mathsf{tx}, \mathsf{state}) \leftarrow \mathsf{OTrans}(\mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}, \mathtt{A})$: Creates and applies a transaction, returns the transaction and the new state.
- $\mathsf{state} \leftarrow \mathsf{OApplyTrans}(\mathsf{tx})$: Checks if a transaction is valid and if so, applies it. Returns the resulting state.
- $\mathsf{auditInfo} \leftarrow \mathsf{OPrepareAudit}(\mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}))$: Creates and returns an audit proof.

Our games make use of bookkeeping functionalities that can be called by the challenger and the available oracles. The bookkeeping keeps a list states of consecutive states created through oracle queries, a set entries containing all the secret keys that control the accounts appearing in these states, and a partition of the keys set into honest and corrupt (controlled by the adversary) keys, honest and corrupt, respectively. The bookkeeping functionalities are:

- $\mathsf{sk} \leftarrow \mathsf{findSecretKey}(\mathsf{pk}, \mathsf{state})$: Finds the secret key corresponding to a public key present in a state.
- $s \leftarrow \mathsf{totalWealth}(\mathsf{set}, \mathsf{state})$: Counts and returns the total amount of funds of the accounts of state that are owned by a set of secret keys ($\mathsf{set} = \mathsf{honest}$ or $\mathsf{set} = \mathsf{corrupt}$).
- $0/1 \leftarrow \mathsf{verifyPolicy}(\mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathsf{aux}))$: Checks whether $\mathtt{pk}_0$ is compliant with policy f for the time period represented by $\mathsf{state}_1, \mathsf{state}_2$.

The bookkeeping functionalities are presented in algorithm 1.1, and the oracles the adversary has access to are presented in algorithm 1.2.

Algorithm 1.1: bookkeeping functionalities

```
entries ← ∅                                                        // set of all secret keys
corrupt ← ∅                                                    // set of corrupt secret keys
honest ← ∅                                                      // set of honest secret keys
states ← []                                      // list of states, updated through oracles
Function findSecretKey(pk, state)
    if state ∉ states then
        return ⊥
    for sk ∈ entries do
        for acct ∈ state.UTXOSet do
            if acct.pk = pk ∧ VerifyKP(sk, acct.pk) = 1 then
                return sk
        for userInfo ∈ state.UserSet do
            if userInfo.pk_0 = pk ∧ VerifyKP(sk, userInfo.pk) = 1 then
                return sk
    return ⊥
Function totalWealth(set, state)
    $s \leftarrow 0$
    for sk ∈ set do
        for acct ∈ state.UTXOSet do
            if VerifyKP(sk, acct.pk) then
                $s \leftarrow s + \mathsf{OpenCom}(\mathsf{sk}, \mathsf{acct.com}_{\mathtt{bl}})$
    return $s$
Function verifyPolicy(pk_0, state_1, state_2, $f$, aux)
    if state_1, state_2 ∉ states ∨ state_1 is not older than state_2 then
        return ⊥
    $A_1, A_2 \leftarrow \emptyset, \emptyset$
    sk ← findSecretKey(pk_0, state_1)
                      // Find accounts owned by sk in state_1.UTXOSet and state_2.UTXOSet resp.
    for acct ∈ state_1.UTXOSet do
        if VerifyKP(sk, acct.pk) then
            $A_1 \leftarrow A_1 \cup \{\mathsf{acct}\}$
    for acct ∈ state_2.UTXOSet do
        if VerifyKP(sk, acct.pk) then
            $A_2 \leftarrow A_2 \cup \{\mathsf{acct}\}$
    if $f$(pk_0, (state_1, state_2), aux) = 1 then
                                                        // Check if $f$ holds using $A_1, A_2$, sk
        return 1
    return 0
```

5.1 Anonymity

In the anonymity game, the challenger first picks a bit $b \leftarrow\$ \{0,1\}$. The adversary, after interacting with the oracles, has to output two sender accounts $\mathsf{acct}_0, \mathsf{acct}_1$, two receiver accounts $\mathsf{acct}'_0, \mathsf{acct}'_1$, two amounts v_0, v_1

Algorithm 1.2: Oracles for security definitions

```
Oracle OCorrupt(pk, state)
        // pk should be a key of an account or user information in state, aborts otherwise
    sk ← findSecretKey(pk, state)
    honest ← honest \ {sk}
    corrupt ← corrupt ∪ {sk}
    return sk
Oracle ORegister()
    state ← bookkeeping.states[−1]                          // most recent state of bookkeeping
    (sk, userInfo, acct, $\pi$) ← Register()
    if VerifyRegister(userInfo, acct, $\pi$, state) = 0 then
        return ⊥                                  // cannot be registered given current state
    entries ← entries ∪ {sk}
    honest ← honest ∪ {sk}
    state′ ← ApplyRegister(userInfo, acct, state); states ← states ∪ [state′]
    return state′
Oracle OCreateAcct(userInfo, A)
    state ← bookkeeping.states[−1]                          // most recent state of bookkeeping
    $tx_{CA}$ ← CreateAcct(userInfo, A)
    if VerifyTrans($tx_{CA}$, state) = 0 then
        return ⊥                                   // transaction cannot be applied to state
    state′ ← ApplyTrans($tx_{CA}$, state); states ← states ∪ [state′]
    return $tx_{CA}$, state′
Oracle ODeleteAcct(userInfo, $acct_C$, $acct_D$, $A_1$, $A_2$)
    state ← bookkeeping.states[−1]
    sk ← findSecretKey($acct_C$)
    $tx_{DA}$ ← DeleteAcct(sk, userInfo, $acct_C$, $acct_D$, $A_1$, $A_2$)
    if VerifyTrans($tx_{DA}$, state) = 0 then
        return ⊥                                   // transaction cannot be applied to state
    state′ ← ApplyTrans($tx_{DA}$, state); states ← states ∪ [state′]
    return $tx_{DA}$, state′
Oracle OTrans(S, R, $\vec{v_S}$, $\vec{v_R}$, A)
    state ← bookkeeping.states[−1]                          // most recent state of bookkeeping
    for sk ∈ entries do
        Take an arbitrary acct ∈ S
        if VerifyKP(sk, acct.pk) = 1 then
            tx ← Trans(S, R, $\vec{v_S}$, $\vec{v_R}$A)     // If sk is not the owner of all accounts in S, the transaction
             will not be created.
            if VerifyTrans(tx, state) = 0 then
                return ⊥                           // transaction cannot be applied to state
            state′ ← ApplyTrans(tx, state); states ← states ∪ [state′]
            return tx, state′
    return ⊥
Oracle OApplyTrans(tx)
    if VerifyTrans(tx, state) = 0 then
        return ⊥
    state′ ← ApplyTrans(tx, state)
    states ← states ∪ [state′]; return state′
Oracle OPrepareAudit($pk_0$, $state_1$, $state_2$, $f$, aux)
    sk ← findSecretKey($pk_0$, $state_1$)
    if $state_1$, $state_2$ ∈ states ∧ $state_1$ is older than $state_2$ then
        auditInfo ← PrepareAudit(sk, $pk_0$, $state_1$, $state_2$, $f$, aux)
        if VerifyAudit($pk_0$, $state_1$, $state_2$, ($f$, aux), auditInfo) then
            return auditInfo
    return ⊥        // $pk_0$ was invalid for the snapshots, $state_1$,$state_2$ were not valid or $f$ was not
     satisfied
```

and an anonymity set $\mathtt{A}$. Then, the challenger creates a transaction in which acct_b sends amount v_b to acct'_b using $\mathtt{A} \cup \{\mathsf{acct}_{1-b}\}$ as the anonymity set. Finally, the adversary has to guess b, and if they guess correctly, they win the game.

In the anonymity game, the following rules must be enforced or else the adversary could trivially guess b.

– Both senders must be honest. If one of the senders were corrupted, the adversary would be able to see whose account's balance decreases.
– Both receivers are honest. If both were corrupted then $\mathsf{acct}'_0 = \mathsf{acct}'_1$ and $v_0 = v_1$. If one is corrupted, the adversary would be able to see which account's balance increased or the amount by which it increased.

The anonymity game is presented in Game 1.3.

Game 1.3: Anonymity game $\mathsf{Exp}^{\text{anon}}_{\mathcal{A}}(\lambda)$

```
Input  : $\lambda$
Output: $\{0,1\}$
$b \leftarrow \{0,1\}$
$(\mathsf{state}_0, \mathsf{pp}) \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1, \mathtt{A}, v_0, v_1) \leftarrow \mathcal{A}^{\mathsf{OCorrupt},\mathsf{ORegister},\mathsf{OCreateAcct},\mathsf{ODeleteAcct},\mathsf{OTrans},\mathsf{OApplyTrans}}(\mathsf{state}_0)$
$\mathsf{state} \leftarrow \mathsf{states}[-1]$                    // most recent state of bookkeeping
$\mathsf{sk}_0 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}_0.\mathsf{pk}, \mathsf{state})$; $\mathsf{sk}_1 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}_1.\mathsf{pk}, \mathsf{state})$
$\mathsf{sk}'_0 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}'_0.\mathsf{pk}, \mathsf{state})$; $\mathsf{sk}'_1 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}'_1.\mathsf{pk}, \mathsf{state})$
if $(\mathsf{sk}_0 \in \mathsf{corrupt} \vee \mathsf{sk}_1 \in \mathsf{corrupt}) \vee ((\mathsf{sk}'_0 \in \mathsf{corrupt} \vee \mathsf{sk}'_1 \in \mathsf{corrupt}) \wedge ((\mathsf{acct}'_0 \neq \mathsf{acct}'_1) \vee (\mathsf{acct}'_0 = \mathsf{acct}'_1 \wedge v_0 \neq v_1))) \vee (\mathsf{acct}_0.\mathtt{bl} < v_0 \vee \mathsf{acct}_1.\mathtt{bl} < v_1)$ then
    return $\perp$
for $y \in \{0,1\}$ do
    $\mathtt{A}_y \leftarrow \mathtt{A}$
    if $\mathsf{sk}_0 \neq \mathsf{sk}_1$ then
        $\mathtt{A}_y \leftarrow \mathtt{A} \cup \{\mathsf{acct}_{1-y}\}$
    if $\mathsf{sk}'_0 \neq \mathsf{sk}'_1$ then
        $\mathtt{A}_y \leftarrow \mathtt{A} \cup \{\mathsf{acct}'_{1-y}\}$
    $\mathsf{tx}_y \leftarrow \mathsf{Trans}(\mathsf{sk}_y, \{\mathsf{acct}_y\}, \{\mathsf{acct}'_y\}, (-v_y), (v_y), \mathtt{A}_y)$
    if $\mathsf{VerifyTrans}(\mathsf{tx}_y, \mathsf{state}) = 0$ then
        return $\perp$
$\mathsf{state}' \leftarrow \mathsf{ApplyTrans}(\mathsf{tx}_b, \mathsf{state})$
$b' \leftarrow \mathcal{A}(\mathsf{state}')$
return $(b = b')$
```

Definition 1. *The* advantage *of the adversary in winning the anonymity game is defined as:* $\mathsf{Adv}^{\text{anon}}_{\mathcal{A}}(\lambda) = \mid \Pr[\mathsf{Exp}^{\text{anon}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \mid$. *A DPS satisfies* anonymity *if for every PPT adversary* $\mathcal{A}$, $\mathsf{Adv}^{\text{anon}}_{\mathcal{A}}(\lambda)$ *is negligible in* λ.

5.2 Theft Prevention

In order for the adversary to win the theft prevention game, they have to output a valid transaction that, when applied, either increases the wealth of the users they control, decreases the wealth of the honest parties, or alters the total wealth of all the users (i.e. the adversary's transaction either created or destroyed wealth). The theft prevention game is presented in Game 1.4.

Definition 2. *The* advantage *of the adversary in winning the theft prevention game is defined as* $\mathsf{Adv}^{\text{theft}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Exp}^{\text{theft}}_{\mathcal{A}}(\lambda) = 1]$ *A DPS satisfies* theft prevention *if for every PPT adversary* $\mathcal{A}$, $\mathsf{Adv}^{\text{theft}}_{\mathcal{A}}(\lambda)$ *is negligible in* λ.

5.3 Audit soundness

In order for the adversary to win the audit soundness game for a policy f, they have to output a valid audit proof for a user that is non-compliant regarding the particular policy. The audit soundness game is presented in Game 1.5.

Definition 3. *The* advantage *of the adversary in winning the audit soundness game for policy* f *is defined as:* $\mathsf{Adv}^{\text{ausound}}_{\mathcal{A},f}(\lambda) = \Pr[\mathsf{Exp}^{\text{ausound}}_{\mathcal{A},f}(\lambda) = 1]$ *A DPS satisfies* audit soundness *for a policy* f *if for every PPT adversary* $\mathcal{A}$, $\mathsf{Adv}^{\text{ausound}}_{\mathcal{A},f}(\lambda)$ *is negligible in* λ.

Game 1.4: Theft prevention game $\mathsf{Exp}^{\text{theft}}_{\mathcal{A}}(\lambda)$

```
Input  : $\lambda$
Output: $\{0,1\}$
$(\mathsf{state}_0, \mathsf{pp}) \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{tx} \leftarrow \mathcal{A}^{\mathsf{OCorrupt,ORegister,OCreateAcct,ODeleteAcct,OTrans,OApplyTrans}}(\mathsf{state}_0)$
$\mathsf{state} \leftarrow \mathsf{states}[-1]$                              // most recent state of bookkeeping
$s_h \leftarrow \mathsf{totalWealth}(.\mathsf{honest}, \mathsf{state})$
$s_c \leftarrow \mathsf{totalWealth}(\mathsf{corrupt}, \mathsf{state})$
if $\mathsf{VerifyTrans}(\mathsf{tx}, \mathsf{state}) = 0$ then
    return $\perp$
$\mathsf{state}' \leftarrow \mathsf{ApplyTrans}(\mathsf{tx}, \mathsf{state})$
$s'_h \leftarrow \mathsf{totalWealth}(\mathsf{honest}, \mathsf{state}')$
$s'_c \leftarrow \mathsf{totalWealth}(\mathsf{corrupt}, \mathsf{state}')$
return $(s'_h < s_h) \vee (s'_c > s_c) \vee (s'_c + s'_h \neq s_c + s_h)$
```

Game 1.5: Audit soundness game $\mathsf{Exp}^{\text{ausound}}_{\mathcal{A},f}(\lambda)$

```
Input  : $\lambda$
Output: $\{0,1\}$
$b \leftarrow \{0,1\}$
$\mathsf{state}_0, \mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{pk}_0, \mathsf{state}_1, \mathsf{state}_2, f, \mathsf{aux}, \mathsf{auditInfo}) \leftarrow \mathcal{A}^{\mathsf{OCorrupt,ORegister,OCreateAcct,ODeleteAcct,OTrans,OApplyTrans,OPrepareAudit}}(\mathsf{state}_0)$
if $\mathsf{VerifyAudit}(\mathsf{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathsf{aux}), \mathsf{auditInfo}) = 1$ then
                    // run bookkeeping and check if $f$ is satisfied and that $\mathsf{state}_1, \mathsf{state}_2$ are valid
    if $\mathsf{verifyPolicy}(\mathsf{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathsf{aux})) = 1$ then
        return 0
    else
        return 1
else
    return $\perp$
```

6 Our construction

6.1 Setup

The Setup algorithm takes as input the security parameter λ and returns the output of UPK.Setup and the initial state which contains an empty UserSet and UTXOSet.

6.2 Accounts

User accounts are of the form $\mathsf{acct} = (\mathsf{pk}, \boxed{\mathtt{bl}}, \boxed{\mathtt{out}}, \boxed{\mathtt{in}})$, where $\mathtt{bl}$ is the account balance and $\mathtt{out}, \mathtt{in}$ is the total amount that the account has sent and received, respectively. Each user may own multiple accounts which are stored in the UTXOSet. The following functionalities create, verify and update accounts.

– $\mathsf{acct} \leftarrow \mathsf{NewAcct}(\mathsf{pk}_0; r_1, r_2, r_3, r_4)$: takes as input a public key pk_0 and outputs a new account of the form $\mathsf{acct} = (\mathsf{pk}, \boxed{\mathtt{bl}}, \boxed{\mathtt{out}}, \boxed{\mathtt{in}})$, where $\mathsf{pk} = \mathsf{Update}(\mathsf{pk}_0; r_1)$, $\boxed{\mathtt{bl}} = \mathsf{Commit}(\mathsf{pk}, 0; r_2)$, $\boxed{\mathtt{out}} = \mathsf{Commit}(\mathsf{pk}, 0; r_3)$ and $\boxed{\mathtt{in}} = \mathsf{Commit}(\mathsf{pk}, 0; r_4)$.

– $0/1 \leftarrow \mathsf{VerifyAcct}(\mathsf{acct}, \mathsf{sk}, \mathtt{bl}, \mathtt{out}, \mathtt{in})$: Parses acct as $(\mathsf{pk}, \mathsf{com}_1, \mathsf{com}_2, \mathsf{com}_3)$ and outputs 1 if

$$\mathsf{VerifyCom}(\mathsf{sk}, \mathsf{pk}, \mathsf{com}_1, \mathtt{bl}) \wedge \mathsf{VerifyCom}(\mathsf{sk}, \mathsf{pk}, \mathsf{com}_2, \mathtt{out}) \wedge$$
$$\mathsf{VerifyCom}(\mathsf{sk}, \mathsf{pk}, \mathsf{com}_3, \mathtt{in}) \wedge (\mathtt{bl}, \mathtt{out}, \mathtt{in} \in \mathcal{V})$$

– $\{\mathsf{acct}'_i\}_{i=1}^n \leftarrow \mathsf{UpdateAcct}(\{\mathsf{acct}_i, \mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{in}i}, \mathtt{v}_{\mathtt{out}i}\}_{i=1}^n; r_1, r_2, r_3, r_4)$
takes as input a set of accounts $\mathsf{acct}_i = (\mathsf{pk}_i, \mathsf{com}_{\mathtt{bl}i}, \mathsf{com}_{\mathtt{out}i}, \mathsf{com}_{\mathtt{in}i})$ and values $\mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{out}i}, \mathtt{v}_{\mathtt{in}i} \in \mathcal{V}$ and outputs a new set of accounts $\{\mathsf{acct}'_i\}_{i=1}^n$, where

$$\mathsf{acct}'_i \leftarrow (\mathsf{Update}(\mathsf{pk}; r_1), \mathsf{com}_{\mathtt{bl}i} \odot \mathsf{Commit}(\mathsf{pk}, \mathtt{v}_{\mathtt{bl}i}; r_2),$$
$$\mathsf{com}_{\mathtt{out}i} \odot \mathsf{Commit}(\mathsf{pk}, \mathtt{v}_{\mathtt{out}i}; r_3), \mathsf{com}_{\mathtt{in}i} \odot \mathsf{Commit}(\mathsf{pk}, \mathtt{v}_{\mathtt{in}i}; r_4)).$$

– $0/1 \leftarrow \mathsf{VerifyUpdateAcct}(\{\mathsf{acct}'_i, \mathsf{acct}_i, \mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{out}i}, \mathtt{v}_{\mathtt{in}i}\}_{i=1}^n; r_1, r_2, r_3, r_4)$: outputs 1 if

$$\{\mathsf{acct}'_i\}_{i=1}^n = \mathsf{UpdateAcct}(\{\mathsf{acct}_i, \mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{out}i}, \mathtt{v}_{\mathtt{in}i}\}_{i=1}^n; r_1, r_2, r_3, r_4) \wedge (|\mathtt{v}_{\mathtt{bl}}|, \mathtt{v}_{\mathtt{out}}, \mathtt{v}_{\mathtt{in}} \in \mathcal{V}).$$

6.3 User information

Each real-world user is associated with a tuple $\mathsf{userInfo} = (\mathtt{pk}_0, \boxed{\mathtt{\#accs}})$, stored in the $\mathsf{UserSet}$. The public key $\mathtt{pk}_0$ is an initial public key provided at the time of registration. The public key of every account owned by the user will share the same secret key with $\mathtt{pk}_0$.

The value $\mathtt{\#accs}$ is the number of accounts in the $\mathsf{UTXOSet}$ that are owned by the user, and is stored as a commitment so that it remains hidden. Keeping track of the number of accounts a user owns is necessary in order to support policies related to value limits, such as the total amount a user has received or sent in a period of time. Otherwise, such policies could be easily bypassed through the creation of sybil identities [5]. The opening of the commitment $\boxed{\mathtt{\#accs}}$ will be revealed only to the $\mathtt{AA}$ during the auditing procedure.

The following functions create, verify and update $\mathsf{userInfo}$ entries of the $\mathsf{UserSet}$.

– $(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}) \leftarrow \mathsf{GenUser}()$: Picks $r_1, r_2, r_3, r_4, r_5 \leftarrow\$ \mathcal{R}$ and let $\vec{r} = (r_1, r_2, r_3, r_4)$. Then runs $(\mathsf{sk}, \mathtt{pk}_0) \leftarrow \mathsf{KGen}()$, $\mathsf{acct} \leftarrow \mathsf{NewAcct}(\mathtt{pk}_0; \vec{r})$, calculates the tuple $\mathsf{userInfo} = (\mathtt{pk}_0, \mathsf{Commit}(\mathtt{pk}_0, 1; r_5))$ and returns $(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct})$.
– $0/1 \leftarrow \mathsf{VerifyUser}((\mathtt{pk}_0, \mathtt{com}), (\mathsf{sk}, \mathtt{\#accs}))$: outputs 1 if $\mathsf{VerifyCom}(\mathsf{sk}, \mathtt{pk}_0, \mathtt{com}, \mathtt{\#accs}) \wedge (\mathtt{\#accs} \in \mathcal{V})$
– $\{\mathsf{userInfo}'_i\}_{i=1}^n \leftarrow \mathsf{UpdateUser}(\{\mathsf{userInfo}_i, \mathtt{v}_{\#accs_i}\}_{i=1}^n; r)$ takes as input a set of user-value pairs where $\mathsf{userInfo}_i = (\mathsf{pk}_{0_i}, \mathsf{com}_{\mathtt{\#accs}i})$ and $\mathtt{v}_{\#accs_i} \in \mathcal{V}$ and outputs a new set of users $\{\mathsf{userInfo}'_i\}_{i=1}^n = \{(\mathsf{pk}_{0_i}, \mathsf{com}'_{\mathtt{\#accs}i})\}_{i=1}^n$ where

$$\mathsf{com}'_{\mathtt{\#accs}i} = \mathsf{com}_{\mathtt{\#accs}i} \odot \mathsf{Commit}(\mathtt{pk}_0, \mathtt{v}_{\#accs}; r)$$

– $0/1 \leftarrow \mathsf{VerifyUpdateUser}(\{\mathsf{userInfo}'_i, \mathsf{user}_i, \mathtt{v}_{\#accs_i}\}_{i=1}^n; r)$ outputs 1 if

$$\{\mathsf{userInfo}'\}_{i=1}^n = \mathsf{UpdateUser}(\{\mathsf{userInfo}_i, \mathtt{v}_{\#accs_i}\}_{i=1}^n; r) \wedge (\mathtt{v}_{\#accs} \in \mathcal{V})$$

6.4 Registration

In order for users to register in the system, they first use the $\mathsf{Register}$ algorithm to create a secret key, a $\mathsf{userInfo}$ entry and a first empty account acct. The $\mathsf{Register}$ algorithm also provides proofs that $\mathsf{userInfo}, \mathsf{acct}$ have been properly created. Then, the user sends $\mathsf{userInfo}, \mathsf{acct}$ and the proofs to the $\mathtt{RA}$ and the $\mathtt{RA}$ verifies the proofs using the $\mathsf{VerifyRegister}$ algorithm. If the proofs verify, the $\mathtt{RA}$ adds $\mathsf{userInfo}$ to the $\mathsf{UserSet}$ and acct to the $\mathsf{UTXOSet}$ using the $\mathsf{ApplyRegister}$ algorithm.

Register The $\mathsf{Register}$ algorithm creates a secret key sk, the entry $\mathsf{userInfo} = (\mathtt{pk}_0, \boxed{1})$ that will be later stored in the $\mathsf{UserSet}$, the user's first account $\mathsf{acct} = (\mathsf{pk}, \boxed{0}, \boxed{0}, \boxed{0})$ and a zero-knowledge proof π for the fact that the commitments $\boxed{1}$ of $\mathsf{userInfo}$ and $\boxed{0}, \boxed{0}, \boxed{0}$ of acct are indeed commitments to the correct values. The proof π can be posted on-chain for public verification.

The user must keep sk secret, and sends through a secure channel $\mathsf{userInfo}, \mathsf{acct}, \pi$ to the $\mathtt{RA}$, together with their real-world identity information. The detailed description of the $\mathsf{Register}$ algorithm is depicted in Figure 1.

The $\mathsf{Register}$ algorithm performs the following steps:

1. Run $(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}) \leftarrow \mathsf{GenUser}()$.
2. Create a zero-knowledge proof π of the relation $R(x, w)$, where $x = (\mathsf{acct}, \mathsf{userInfo})$, $w = (\mathsf{sk})$ and $R(x, w) = 1$ if:

$$\begin{aligned}
&\mathsf{VerifyCom}(\mathsf{userInfo.pk}_0, \mathsf{userInfo.com}_{\mathtt{\#accs}}, (\mathsf{sk}, 1)) = 1\\
&\wedge \mathsf{VerifyKP}(\mathsf{userInfo.pk}_0, \mathsf{sk}) = 1\\
&\wedge \mathsf{VerifyKP}(\mathsf{acct.pk}, \mathsf{sk}) = 1\\
&\wedge \mathsf{VerifyCom}(\mathsf{acct.pk}, \mathsf{acct.com}_{\mathtt{bl}}, (\mathsf{sk}, 0)) = 1\\
&\wedge \mathsf{VerifyCom}(\mathsf{acct.pk}, \mathsf{acct.com}_{\mathtt{out}}, (\mathsf{sk}, 0)) = 1\\
&\wedge \mathsf{VerifyCom}(\mathsf{acct.pk}, \mathsf{acct.com}_{\mathtt{in}}, (\mathsf{sk}, 0)) = 1
\end{aligned}$$

3. Return $(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}, \pi)$.

Fig. 1. The $\mathsf{Register}$ algorithm.

Verify Register The $\mathsf{VerifyRegister}(\mathsf{userInfo}, \mathsf{acct}, \pi, \mathsf{state})$ algorithm guarantees the validity of the registration information. It first checks that the $\mathsf{userInfo.pk}_0$ does not already exist in a $\mathsf{userInfo}$ entry of $\mathsf{UserSet}$. Afterwards, it executes the verification algorithm for the NIZK argument π and returns its result.

Apply Register The $\mathsf{ApplyRegister}(\mathsf{userInfo}, \mathsf{acct}, \mathsf{state})$ algorithm runs after the registration verification and adds a new record to the $\mathsf{UserSet}$ containing the $\mathsf{userInfo}$ as well as a new record to the $\mathsf{UTXOSet}$ containing the newly created account acct.

6.5 Transactions

Trans Algorithm Transactions enable a sender to redistribute their wealth to one or more recipients. Similarly to Quisquis [10], transactions are composed of input and output sets, which both include the sender and the intended recipients, and a NIZK proof that the output list has been computed according to the protocol specification. We assume that the size of each of the inputs and outputs sets is a predetermined number $\mathtt{n}$.

The Trans algorithm is used to create a transaction that redistributes a number of coins from a set of sender accounts, which are owned by the same secret key, to a set of receiver accounts. In order to substract an amount from a sender account or add an amount to a receiver account, the homomorphic property of the commitment scheme is used. Furthermore, in the algorithm the total amount sent and total amount received of the sender and receiver accounts is also updated appropriately. Finally, the account public keys are re-randomized in order to hide the connection between the input and output accounts.

In order to hide the participating accounts, an anonymity set is included. The balances of the accounts belonging to the anonymity set do not change, however the commitments and the public keys are re-randomized in order to be indistinguishable from the actual participating accounts. The account updates happen though the invocation of the $\mathsf{UpdateAcct}$ algorithm, and the outputs set is composed of these updated accounts.

The ordering of the accounts in the input and output sets should not remain the same, since this trivially reveals the link between every account and its update. Therefore, the input and output lists are always ordered in some canonical order. This can be thought of as applying a random permutation to shuffle the updated accounts.

The detailed description of the Trans algorithm is depicted in Figure 2. It takes as input the sender's secret key sk, the set of sender accounts $\mathtt{S}$, the set of receiver accounts $\mathtt{R}$, two vectors $\overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}$ containing the desired changes to the balances of the sender and receiver accounts respectively, and an anonymity set $\mathtt{A}$. It returns a transaction $\mathsf{tx} = (\mathtt{inputs}, \mathtt{outputs}, \pi)$, where π is a zero-knowledge proof that $\mathtt{outputs}$ is created correctly.

Due to the way transactions are generated, every address appears at most twice: once when it is created in the output of some transaction, and once when included in the inputs of another transaction (regardless of whether it serves as the actual sender or is only included for anonymity).

Our transaction construction is similar to the one of Quisquis [10], with the difference that we introduce the vectors $\overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}$ to perform the updates to the associated total amount sent and total amount received of the accounts.

Proof of transaction correctness In each transaction created from Trans algorithm a prover essentially has to prove in zero-knowledge that:

1. accounts in $\mathtt{outputs}$ are proper updates of $\mathtt{inputs}$
2. the updates of balances satisfy preservation of value
3. balances in accounts of recipients and anonymity set do not decrease
4. the sender account in $\mathtt{outputs}$ contain a balance in $\mathcal{V}$
5. the vectors $\overrightarrow{\mathtt{v_{bl}}}', \overrightarrow{\mathtt{v_{out}}}'$ have the same values for the sender accounts and $\overrightarrow{\mathtt{v_{bl}}}', \overrightarrow{\mathtt{v_{in}}}'$ for the receivers accounts and $(\overrightarrow{\mathtt{v_{out}}}', \overrightarrow{\mathtt{v_{in}}}')$ have zero value for the rest.

Properties 3,4 can be proved by range proofs and we implement them with Bulletproofs [4]. For the properties 1,2,5 we are doing the following analysis similar to Quisquis[10].

Let the sender's accounts be $\mathtt{inputs}_1, \ldots, \mathtt{inputs}_s$ and the receivers' accounts be $\mathtt{inputs}_{s+1}, \ldots, \mathtt{inputs}_t$.

In order to easily verify the validity of the updates, the prover creates accounts $\overrightarrow{\mathsf{acct}}_\delta$, where $\mathsf{acct}_{\delta,i} = (\mathsf{pk}_i, \boxed{\mathtt{v_{bl}}_i}, \boxed{\mathtt{v_{out}}_i}, \boxed{\mathtt{v_{in}}_i})$. Now in order to prove property 5, the prover shows that for $\mathsf{acct}_{\delta,1}, \ldots, \mathsf{acct}_{\delta,s}$ the values under the $\boxed{\mathtt{v_{bl}}_i}$ and $\boxed{\mathtt{v_{out}}_i}$ are the same. Respectively for the recipients, for $\mathsf{acct}_{\delta,s+1}, \ldots, \mathsf{acct}_{\delta,t}$ the values under the $\boxed{\mathtt{v_{bl}}_i}$ and $\boxed{\mathtt{v_{in}}_i}$ are the same.

Since the sender-prover knows all the values of the acct_δ, they can create commitments for the same values under a different public key $\mathsf{pk}_\epsilon = (g, h)$. So the prover creates $\overrightarrow{\mathsf{acct}}_\epsilon$ where $\mathsf{acct}_{\epsilon i} = ((g, h), \boxed{\mathtt{v_{bl}}_i}_\epsilon, \boxed{\mathtt{v_{out}}_i}_\epsilon, \boxed{\mathtt{v_{in}}_i}_\epsilon)$. Then they use the homomorphic property of the commitment in order to prove the preservation of value, since $\sum_i \mathtt{v_{bl}}_i = 0 \iff \prod_i \boxed{\mathtt{v_{bl}}_i}_\epsilon$ is a commitment of 0 under $\mathsf{pk}_\epsilon = (g, h)$. The values in $\mathsf{acct}_{\epsilon,s+1}, \ldots, \mathsf{acct}_{\epsilon,t}$ will be used to prove that balances of recipients set and anonymity set is not decreased, meaning $\mathtt{v_{bl}}_{\epsilon,s+1}, \ldots, \mathtt{v_{bl}}_{\epsilon,n} \in \mathcal{V}$.

Now in order to hide the sender's and the receiver's position in $\mathtt{inputs}$ and $\mathtt{outputs}$ we first shuffle $\mathtt{inputs}$ list to $\mathtt{inputs}'$ before the updates, then we execute the updates to produce $\mathtt{outputs}'$, and finally we shuffle again after the updates to get $\mathtt{outputs}$ in arbitrary order. The first shuffle uses the aforementioned permutation

The algorithm $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}, \mathtt{A})$ performs the following steps:

1. Ensure that for each $\mathsf{acct} \in \mathtt{S}$, $\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct.pk}) = 1$, and that $|\mathtt{S}| = |\overrightarrow{\mathtt{v_S}}|, |\mathtt{R}| = |\overrightarrow{\mathtt{v_R}}|$.
2. Let $\mathtt{I_S} = \{1, \dots, |\mathtt{S}|\}$. For all $i \in \mathtt{I_S}$, calculate the opening of the committed balance $\boxed{\mathtt{bl}_i}$ of $\mathsf{acct}_i \in \mathtt{S}$, denoted $\mathtt{bl}_i$.
3. Let $\overrightarrow{\mathtt{v_{bl}}} = \overrightarrow{\mathtt{v_S}} || \overrightarrow{\mathtt{v_R}}$, where $||$ denotes vector concatenation. Let also $\mathtt{I_R} = \{|\mathtt{S}|+1, \dots, |\mathtt{S}| + |\mathtt{R}|\}$. Ensure that:
   (a) $\sum_{i \in \mathtt{I_S} \cup \mathtt{I_R}} \mathtt{v_{bl}}_i = 0$
   (b) $\forall i \in \mathtt{I_R} : \mathtt{v_{bl}}_i \in \mathcal{V}$
   (c) $\forall i \in \mathtt{I_S} : -\mathtt{v_{bl}}_i \in \mathcal{V} \land \mathtt{bl}_i + \mathtt{v_{bl}}_i \in \mathcal{V}$
4. Construct $\overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}$ as follows:
   (a) $\overrightarrow{\mathtt{v_{out}}} = \overrightarrow{\mathtt{v_S}} || \underbrace{(0, \dots, 0)}_{\text{length } |\mathtt{R}|} || \underbrace{(0, \dots, 0)}_{\text{length } |\mathtt{A}|}$
   (b) $\overrightarrow{\mathtt{v_{in}}} = \underbrace{(0, \dots, 0)}_{\text{length } |\mathtt{S}|} || \overrightarrow{\mathtt{v_R}} || \underbrace{(0, \dots, 0)}_{\text{length } |\mathtt{A}|}$.

   Furthermore, expand $\overrightarrow{\mathtt{v_{bl}}}$ too with zero values for each $\mathsf{acct} \in \mathtt{A}$.
5. Order $\mathtt{P} \cup \mathtt{A}$ in some canonical order and let $\mathtt{inputs}$ be the result. Let also $\overrightarrow{\mathtt{v_{bl}}}', \overrightarrow{\mathtt{v_{out}}}', \overrightarrow{\mathtt{v_{in}}}'$ be the permutation of $\overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}$ in the same order. Let $\mathtt{I}^*_\mathtt{S}, \mathtt{I}^*_\mathtt{R}, \mathtt{I}^*_\mathtt{A}$ denote the indices of the respective accounts of the sender, the recipients and the anonymity set in this list.
6. Pick $r_1, r_2, r_3, r_4 \leftarrow\$ \mathcal{R}$ and let $\overrightarrow{r} = (r_1, r_2, r_3, r_4)$.
   Perform $\mathsf{UpdateAcct}(\mathtt{inputs}, \overrightarrow{\mathtt{v_{bl}}}', \overrightarrow{\mathtt{v_{out}}}', \overrightarrow{\mathtt{v_{in}}}'; \overrightarrow{r})$, order the result in some canonical order, and denote by $\mathtt{outputs}$ the final result.
7. Let $\psi : [\mathtt{n}] \to [\mathtt{n}]$ be the implicit permutation mapping $\mathtt{inputs}$ into $\mathtt{outputs}$; such that accounts $\mathsf{acct}_i \in \mathtt{inputs}$ and $\mathsf{acct}'_{\psi(i)} \in \mathtt{outputs}$ share the same secret key.
8. Form a zero-knowledge proof π of the relation $R(x, w)$, where $x = (\mathtt{inputs}, \mathtt{outputs}), w = (\mathsf{sk}, \{\mathtt{bl}_i, \mathtt{out}_i, \mathtt{in}_i\}_{i \in \mathtt{I}^*_\mathtt{S}}, \overrightarrow{\mathtt{v_{bl}}}', \overrightarrow{\mathtt{v_{out}}}', \overrightarrow{\mathtt{v_{in}}}', \overrightarrow{r}, \psi, \mathtt{I}^*_\mathtt{S}, \mathtt{I}^*_\mathtt{R}, \mathtt{I}^*_\mathtt{A})$, and $R(x, w) = 1$ if

$$
\begin{aligned}
&\mathsf{VerifyUpdateAcct}(\mathsf{acct}'_{\psi(i)}, \mathsf{acct}_i, 0, 0, 0; \overrightarrow{r}) = 1 \ \forall i \in \mathtt{I}^*_\mathtt{A} \\
&\land (\mathsf{VerifyUpdateAcct}(\mathsf{acct}'_{\psi(i)}, \mathsf{acct}_i, \mathtt{v_{bl}}'_i, \mathtt{v_{out}}'_i, \mathtt{v_{in}}'_i; \overrightarrow{r}) = 1 \land \mathtt{v_{bl}}'_i, \mathtt{v_{out}}'_i, \mathtt{v_{in}}'_i \in \mathcal{V}) \ \forall i \in \mathtt{I}^*_\mathtt{R} \\
&\land \mathsf{VerifyUpdateAcct}(\mathsf{acct}'_{\psi(i)}, \mathsf{acct}_i, \mathtt{v_{bl}}'_i, \mathtt{v_{out}}'_i, \mathtt{v_{in}}'_i; \overrightarrow{r}) = 1 \ \forall i \in \mathtt{I}^*_\mathtt{S} \\
&\land \mathsf{VerifyAcct}(\mathsf{acct}'_{\psi(i)}, \mathsf{sk}, \mathtt{bl}_i + \mathtt{v_{bl}}'_i, \mathtt{out}_i + \mathtt{v_{out}}'_i, \mathtt{in}_i + \mathtt{v_{in}}'_i) = 1 \ \forall i \in \mathtt{I}^*_\mathtt{S} \\
&\land \sum_{i \in \mathtt{I}^*_\mathtt{S} \cup \mathtt{I}^*_\mathtt{R} \cup \mathtt{I}^*_\mathtt{A}} \mathtt{v_{bl}}'_i = 0 \\
&\land \mathtt{v_{bl}}'_i = \mathtt{v_{out}}'_i \ \forall i \in \mathtt{I}^*_\mathtt{S} \\
&\land \mathtt{v_{bl}}'_i = \mathtt{v_{in}}'_i \ \forall i \in \mathtt{I}^*_\mathtt{R} \\
&\land \mathtt{v_{out}}'_i = \mathtt{v}'_{\mathtt{in}i} = 0 \ \forall i \in \mathtt{I}^*_\mathtt{A}
\end{aligned}
$$

The transaction created is $\mathsf{tx} = (\mathtt{inputs}, \mathtt{outputs}, \pi)$.

Fig. 2. The Trans algorithm.

where senders' accounts are first,followed by recipients' accounts and then the anonymity set. The second shuffle uses a permutation in order to order the `outputs` lexicographically.

Therefore, we need some auxiliary functions for the proof that are defined as following:

- $\mathsf{CreateDelta}(\{\mathsf{acct}_i\}_{i=1}^n, \{\mathtt{v}_{\mathtt{bl}i}\}_{i=1}^n, \{\mathtt{v}_{\mathtt{out}i}\}_{i=1}^n, \{\mathtt{v}_{\mathtt{in}i}\}_{i=1}^n)$: Creates a set of accounts that contains the differences between accounts' variables `bl`, `out`, `in` in the input and output accounts, and another set of accounts that also contains these differences but all with the global public key (g, h):
  1. Parse $\mathsf{acct}_i = (\mathsf{pk}_i, \mathsf{com}_{\mathtt{bl},i}, \mathsf{com}_{\mathtt{out},i}, \mathsf{com}_{\mathtt{in},i})$. Sample $r_{(\mathtt{bl}|\mathtt{out}|\mathtt{in}),1}, \dots, r_{(\mathtt{bl}|\mathtt{out}|\mathtt{in}),n-1} \leftarrow\!\!\$\ \mathbb{F}_p$ and set $r_{(\mathtt{bl}|\mathtt{out}|\mathtt{in}),n} = -\sum_{i=1}^{n-1} r_{(\mathtt{bl}|\mathtt{out}|\mathtt{in}),i}$.
  2. Set $\mathsf{acct}_{\delta,i} = (\mathsf{pk}_i, \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{bl}i}; r_{\mathtt{bl},i}), \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{out}i}; r_{\mathtt{out},i}), \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{in}i}; r_{\mathtt{in},i}))$
  3. Set $\mathsf{acct}_{\epsilon,i} = ((g,h), \mathsf{Commit}((g,h), \mathtt{v}_{\mathtt{bl}i}; r_{\mathtt{bl},i}), \mathsf{Commit}((g,h), \mathtt{v}_{\mathtt{out}i}; r_{\mathtt{out},i}), \mathsf{Commit}((g,h), \mathtt{v}_{\mathtt{in}i}; r_{\mathtt{in},i}))$
  4. Output $(\{\mathsf{acct}_{\delta,i}\}_{i=1}^n, \{\mathsf{acct}_{\epsilon,i}\}_{i=1}^n, \vec{r_{\mathtt{bl}}}, \vec{r_{\mathtt{out}}}, \vec{r_{\mathtt{in}}})$

- $\mathsf{VerifyDelta}(\{\mathsf{acct}_{\delta,i}\}_{i=1}^n, \{\mathsf{acct}_{\epsilon,i}\}_{i=1}^n, \vec{\mathtt{v}_{\mathtt{bl}}}, \vec{\mathtt{v}_{\mathtt{out}}}, \vec{\mathtt{v}_{\mathtt{in}}}, \vec{r_{\mathtt{bl}}}, \vec{r_{\mathtt{out}}}, \vec{r_{\mathtt{in}}})$: Verifies that accounts created using $\mathsf{CreateDelta}$ are consistent:
  1. Parse $\mathsf{acct}_{\delta,i} = (\mathsf{pk}_i, \boxed{\mathtt{v}_{\mathtt{bl}i}}, \boxed{\mathtt{v}_{\mathtt{out}i}}, \boxed{\mathtt{v}_{\mathtt{in}i}})$ and $\mathsf{acct}_{\epsilon,i} = (\mathsf{pk}_{\epsilon,i}, \mathsf{com}_{\epsilon,i})$
  2. If $\prod_{i=1}^n \mathsf{com}_{\epsilon,i} = (1,1)$ and $\forall i\ \boxed{\mathtt{v}_{\mathtt{bl}i}} = \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{bl}i}; r_{\mathtt{bl},i}) \wedge \boxed{\mathtt{v}_{\mathtt{out}i}} = \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{out}i}; r_{\mathtt{out},i}) \wedge \boxed{\mathtt{v}_{\mathtt{in}i}} = \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{in}i}; r_{\mathtt{in},i}) \wedge \mathsf{acct}_{\epsilon,i} = ((g,h), \mathsf{Commit}((g,h), (\mathtt{v}_{\mathtt{bl}i}; r_{\mathtt{bl},i}))$ output 1. Else output 0.

- $\mathsf{VerifyNonNegative}(\mathsf{acct}_\epsilon, \mathtt{v}_{\mathtt{bl}}, r_{\mathtt{bl}})$: Verifies that an account contains a balances in $\mathcal{V}$:
  1. If $\mathsf{acct}_\epsilon = ((g,h), (g^r, g^v h^r)) \wedge v \in \mathcal{V}$ outputs 1. Else output 0.

- $\mathsf{UpdateDelta}(\{\mathsf{acct}_i\}_{i=1}^n, \{\mathsf{acct}_{\delta,i}\}_{i=1}^n)$: Updates the input accounts by $\mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{out}i}, \mathtt{v}_{\mathtt{in}i}$ but with the public key unchanged:
  1. Parse $\mathsf{acct}_i = (\mathsf{pk}_i, \mathsf{com}_{\mathtt{bl},i}, \mathsf{com}_{\mathtt{out},i}, \mathsf{com}_{\mathtt{in},i})$ and $\mathsf{acct}_{\delta,i} = (\mathsf{pk}'_i, \boxed{\mathtt{v}_{\mathtt{bl}i}}, \boxed{\mathtt{v}_{\mathtt{out}i}}, \boxed{\mathtt{v}_{\mathtt{in}i}})$.
  2. If $\mathsf{pk}_i = \mathsf{pk}'_i\ \forall i$ output $\{(\mathsf{pk}_i, \mathsf{com}_{\mathtt{bl},i} \cdot \boxed{\mathtt{v}_{\mathtt{bl}i}}, \mathsf{com}_{\mathtt{out},i} \cdot \boxed{\mathtt{v}_{\mathtt{out}i}}, \mathsf{com}_{\mathtt{in},i} \cdot \boxed{\mathtt{v}_{\mathtt{in}i}})\}$, else output $\perp$.

- $\mathsf{VerifyUD}(\mathsf{acct}, \mathsf{acct}', \mathsf{acct}_\delta)$: Verifies that $\mathsf{UpdateDelta}$ was performed correctly:
  1. Parse $\mathsf{acct} = (\mathsf{pk}, \mathsf{com}_{\mathtt{bl}}, \mathsf{com}_{\mathtt{out}}, \mathsf{com}_{\mathtt{in}}), \mathsf{acct}' = (\mathsf{pk}, \mathsf{com}'_{\mathtt{bl}}, \mathsf{com}'_{\mathtt{out}}, \mathsf{com}'_{\mathtt{in}})$ and $\mathsf{acct}_\delta = (\mathsf{pk}_\delta, \boxed{\mathtt{v}_{\mathtt{bl}}}, \boxed{\mathtt{v}_{\mathtt{out}}}, \boxed{\mathtt{v}_{\mathtt{in}}})$.
  2. Check that $\mathsf{pk} = \mathsf{pk}' = \mathsf{pk}_\delta \wedge \mathsf{com}'_{\mathtt{bl}} = \mathsf{com}_{\mathtt{bl}} \cdot \boxed{\mathtt{v}_{\mathtt{bl}}} \wedge \mathsf{com}'_{\mathtt{out}} = \mathsf{com}_{\mathtt{out}} \cdot \boxed{\mathtt{v}_{\mathtt{out}}} \wedge \mathsf{com}'_{\mathtt{in}} = \mathsf{com}_{\mathtt{in}} \cdot \boxed{\mathtt{v}_{\mathtt{in}}}$.

- $\mathsf{VerifyDeltaSender}(\mathsf{acct}_\delta, v, r_{\mathtt{bl}}, r_{\mathtt{out}})$: Verifies that sender's value `out` is correct.
  1. Parse $\mathsf{acct}_\delta = (\mathsf{pk}_\delta, \boxed{\mathtt{v}_{\mathtt{bl}}}, \boxed{\mathtt{v}_{\mathtt{out}}}, \boxed{\mathtt{v}_{\mathtt{in}}})$.
  2. If $\boxed{\mathtt{v}_{\mathtt{bl}}} = \mathsf{Commit}(\mathsf{pk}_\delta, v; r_{\mathtt{bl}}) \wedge \boxed{\mathtt{v}_{\mathtt{out}}} = \mathsf{Commit}(\mathsf{pk}_\delta, v; r_{\mathtt{out}})$ then return 1. Else return 0.

- $\mathsf{VerifyDeltaReceiver}(\mathsf{acct}_\delta, v, r_{\mathtt{bl}}, r_{\mathtt{in}})$: Verifies that receiver's value `in` is correct.
  1. Parse $\mathsf{acct}_\delta = (\mathsf{pk}_\delta, \boxed{\mathtt{v}_{\mathtt{bl}}}, \boxed{\mathtt{v}_{\mathtt{out}}}, \boxed{\mathtt{v}_{\mathtt{in}}})$.
  2. If $\boxed{\mathtt{v}_{\mathtt{bl}}} = \mathsf{Commit}(\mathsf{pk}_\delta, v; r_{\mathtt{bl}}) \wedge \boxed{\mathtt{v}_{\mathtt{in}}} = \mathsf{Commit}(\mathsf{pk}_\delta, v; r_{\mathtt{in}})$ then return 1. Else return 0.

Then the $\mathsf{NIZK.Prove}_{\mathsf{Trans}}(x, w)$ performs the following steps:

1. Parse $x = (\mathtt{inputs}, \mathtt{outputs}), w = (\mathsf{sk}, \{\mathtt{bl}_i, \mathtt{out}_i, \mathtt{in}_i\}_{i \in \mathtt{I}^*_{\mathtt{S}}}, \overrightarrow{\mathtt{v}_{\mathtt{bl}}}', \overrightarrow{\mathtt{v}_{\mathtt{out}}}', \overrightarrow{\mathtt{v}_{\mathtt{in}}}', \vec{r}, \psi, \mathtt{I}^*_{\mathtt{S}}, \mathtt{I}^*_{\mathtt{R}}, \mathtt{I}^*_{\mathtt{A}})$. If $R(x,w) = 0$ abort;
2. Let ψ_1 be a permutation such that $\psi_1(\mathtt{I}^*_{\mathtt{S}}) = [1, s], \psi_1(\mathtt{I}^*_{\mathtt{R}}) = [s+1, t]$ and $\psi_1(\mathtt{I}^*_{\mathtt{A}}) = [t+1, n]$;
3. Sample $\rho_1, \rho_2, \rho_3, \rho_4 \leftarrow\!\!\$\ \mathbb{F}_p$ and let $\vec{\rho} = (\rho_1, \rho_2, \rho_3, \rho_4)$;
4. Set $\mathtt{inputs}' = \mathsf{UpdateAcct}(\{\mathtt{inputs}_{\psi_1(i)}, 0, 0, 0\}_i; \vec{\rho})$;
5. Set vectors $\overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}$ such that $\mathtt{v}_{\mathtt{bl}i} = \mathtt{v}_{\mathtt{bl}}{}'_{\psi(i)}$, $\mathtt{v}_{\mathtt{out}i} = \mathtt{v}_{\mathtt{out}}{}'_{\psi(i)}$,$\mathtt{v}_{\mathtt{in}i} = \mathtt{v}_{\mathtt{in}}{}'_{\psi(i)}$;
6. Set $(\{\mathsf{acct}_{\delta,i}\}, \{\mathsf{acct}_{\epsilon,i}\}, \vec{r_{\mathtt{bl}}}, \vec{r_{\mathtt{out}}}, \vec{r_{\mathtt{in}}}) \leftarrow\!\!\$\ \mathsf{CreateDelta}(\mathtt{inputs}', \overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}})$;
7. Update $\mathtt{outputs}' \leftarrow \mathsf{UpdateDelta}(\mathtt{inputs}', \{\mathsf{acct}_{\delta,i}\})$;
8. Let $\psi_2 = \psi_1^{-1} \circ \psi, \rho'_1 = \frac{r_1}{\rho_1}, \vec{\rho'_2} = \frac{r_2 - \rho_2}{\rho_1} - r_{\mathtt{bl}i}, \vec{\rho'_3} = \frac{r_3 - \rho_3}{\rho_1} - r_{\mathtt{out}i}, \vec{\rho'_4} = \frac{r_4 - \rho_4}{\rho_1} - r_{\mathtt{in}i}$ and let $\vec{\rho'} = (\rho'_1, \vec{\rho'_2}, \vec{\rho'_3}, \vec{\rho'_4})$.
9. Update $\mathtt{outputs} = \mathsf{UpdateAcct}(\{\mathtt{outputs}'_{\psi_2(i)}, 0, 0, 0\}_i; \vec{\rho'})$

10. Generate a ZK proof $\pi = (\mathtt{inputs}', \mathtt{outputs}', \mathsf{acct}_\delta, \mathsf{acct}_\epsilon, \pi_1, \pi_2, \pi_3)$ for the relation $R_1 \wedge R_2 \wedge R_3$ where:

$$
\begin{aligned}
R_1 = \{&(\mathtt{inputs}, \mathtt{inputs}', (\psi_1, \vec{\rho}))| \\
&\mathsf{VerifyUpdateAcct}(\{\mathtt{inputs}'_i, \mathtt{inputs}_{\psi_1(i)}, 0, 0, 0\}_i; \vec{\rho}) = 1\}, \\
R_2 = \{&((\mathtt{inputs}', \mathtt{outputs}', \mathsf{acct}_\delta, \mathsf{acct}_\epsilon), (\mathsf{sk}, \{\mathtt{bl}, \mathtt{out}, \mathtt{in}\}_{i=0}^{s}, \overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}, \overrightarrow{r_{\mathtt{bl}}}, \overrightarrow{r_{\mathtt{out}}}, \overrightarrow{r_{\mathtt{in}}}))| \\
&\mathsf{VerifyUD}(\mathtt{inputs}'_i, \mathtt{outputs}'_i, \mathsf{acct}_{\delta,i}) = 1 \ \forall i \\
&\wedge \mathsf{VerifyUpdateAcct}(\mathtt{inputs}'_i, \mathtt{outputs}'_i, 0, 0, 0; 1, r_{\mathtt{bl},i}, r_{\mathtt{out},i}, r_{\mathtt{in},i}) = 1 \ \forall i \in [t+1, n] \\
&\wedge \mathsf{VerifyNonNegative}(\mathsf{acct}_{\epsilon,i}, \mathtt{v_{bl}}_i, r_{\mathtt{bl},i}) = 1 \ \forall i \in [s+1, t] \\
&\wedge \mathsf{VerifyAcct}(\mathtt{outputs}'_i, (\mathsf{sk}, \mathtt{bl}_i + \mathtt{v_{bl}}_i)) = 1 \ \forall i \in [1, s] \\
&\wedge \mathsf{VerifyDelta}(\{\mathsf{acct}_{\delta,i}\}, \{\mathsf{acct}_{\epsilon,i}\}, \overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}, \overrightarrow{r_{\mathtt{bl}}}, \overrightarrow{r_{\mathtt{out}}}, \overrightarrow{r_{\mathtt{in}}}) = 1 \\
&\wedge \mathsf{VerifyDeltaSender}(\mathsf{acct}_{\delta,i}, \mathtt{v_{bl}}_i, r_{\mathtt{bl},i}, r_{\mathtt{out},i}) = 1 \ \forall i \in [1, s] \\
&\wedge \mathsf{VerifyDeltaReceiver}(\mathsf{acct}_{\delta,i}, \mathtt{v_{bl}}_i, r_{\mathtt{bl},i}, r_{\mathtt{in},i}) = 1 \ \forall i \in [s+1, t]\}, \\
R_3 = \{&(\mathtt{outputs}', \mathtt{outputs}, (\psi_2, \vec{\rho'}))| \\
&\mathsf{VerifyUpdateAcct}(\{\mathtt{outputs}_i, \mathtt{outputs}'_{\psi_1(2)}, 0, 0, 0\}_i; \vec{\rho'}) = 1\}
\end{aligned}
$$

Now R_1, R_3 can be proven using a slight modification of the Bayer-Groth shuffle argument [1]. The Σ_2 protocol that proves R_2 consists of the following sub-protocols:

1. Σ_{vu}: trivial check of $\mathsf{VerifyUD}$.
2. Σ_δ: prover shows knowledge of $\overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}, \overrightarrow{r_{\mathtt{bl}}}, \overrightarrow{r_{\mathtt{out}}}, \overrightarrow{r_{\mathtt{in}}}$ such that
$\mathsf{VerifyDelta}(\{\mathsf{acct}_{\delta,i}\}_{i=1}^n, \{\mathsf{acct}_{\epsilon,i}\}_{i=1}^n, \overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}, \overrightarrow{r_{\mathtt{bl}}}, \overrightarrow{r_{\mathtt{out}}}, \overrightarrow{r_{\mathtt{in}}}) = 1$.
Σ_δ can be implemented by using Σ_{com}:
$\Sigma_\delta = \wedge_{i=1}^n \Sigma_{com}((\mathsf{pk}_{\delta,i}, \mathsf{com}_{\delta,i}), (\mathsf{pk}_{\epsilon,i}, \mathsf{com}_{\epsilon,i}); (\mathtt{v_{bl}}, r_{\mathtt{bl},i}, r_{\mathtt{bl},i}))$
$\wedge_{i=1}^n \Sigma_{com}((\mathsf{pk}_{\delta,i}, \mathsf{com}_{\delta,i}), (\mathsf{pk}_{\epsilon,i}, \mathsf{com}_{\epsilon,i}); (\mathtt{v_{out}}, r_{\mathtt{out},i}, r_{\mathtt{out},i}))$
$\wedge_{i=1}^n \Sigma_{com}((\mathsf{pk}_{\delta,i}, \mathsf{com}_{\delta,i}), (\mathsf{pk}_{\epsilon,i}, \mathsf{com}_{\epsilon,i}); (\mathtt{v_{in}}, r_{\mathtt{in},i}, r_{\mathtt{in},i}))$, but the verifier additionally checks that $\mathsf{pk}_{\epsilon,i} = (g, h) \ \forall i$ and that $\prod_{i=1}^n \boxed{\mathtt{v_{bl}}_i}_\epsilon = (1, 1)$.
3. Σ_{zero}^i: prover shows knowledge of $r_{\mathtt{bl},i}, r_{\mathtt{out},i}, r_{\mathtt{in},i}$ such that
$\mathsf{VerifyUpdateAcct}(\mathtt{inputs}'_i, \mathtt{outputs}'_i, 0, 0, 0; (1, r_{\mathtt{bl},i}, r_{\mathtt{out},i}, r_{\mathtt{in},i})) = 1$.
The sub-argument can be written as follows:
given $\mathsf{acct}_1 = (\mathsf{pk}, \boxed{\mathtt{v_{bl}}}_1, \boxed{\mathtt{v_{out}}}_1, \boxed{\mathtt{v_{in}}}_1), \mathsf{acct}_2 = (\mathsf{pk}, \boxed{\mathtt{v_{bl}}}_2, \boxed{\mathtt{v_{out}}}_2, \boxed{\mathtt{v_{in}}}_2)$, the prover knows $r_{\mathtt{bl}}, r_{\mathtt{out}}, r_{\mathtt{in}}$ such that $\boxed{\mathtt{v_{bl}}}_1 = \boxed{\mathtt{v_{bl}}}_2 \cdot \mathsf{pk}^{r_{\mathtt{bl}}}, \boxed{\mathtt{v_{out}}}_1 = \boxed{\mathtt{v_{out}}}_2 \cdot \mathsf{pk}^{r_{\mathtt{out}}}, \boxed{\mathtt{v_{in}}}_1 = \boxed{\mathtt{v_{in}}}_2 \cdot \mathsf{pk}^{r_{\mathtt{in}}}$. The equation is equivalent to: $\wedge_{i=\{\mathtt{bl},\mathtt{out},\mathtt{in}\}} \mathsf{VerifyUpdate}(\mathsf{pk}, \frac{com_{2,i}}{com_{1,i}}, r_i) = 1$, hence can be done using AND-proofs of Σ_{vu}.
4. Σ_{vds}^i: prover shows knowledge of $v, r_{\mathtt{bl},i}, r_{\mathtt{out},i}$ such that $\mathsf{acct}_{\delta,i}$ has the same value under commitments $\boxed{\mathtt{v_{bl}}}, \boxed{\mathtt{v_{out}}}$. Σ_{vds}^i can be implemented by using $\Sigma_{com}((\mathsf{pk}_{\delta,i}, \boxed{\mathtt{v_{bl}}}_i), (\mathsf{pk}_{\delta,i}, \boxed{\mathtt{v_{out}}}_i); (\mathtt{v_{bl}}_i, r_{\mathtt{bl},i}, r_{\mathtt{out},i}))$.
5. Σ_{vdr}^i: prover shows knowledge of $v, r_{\mathtt{bl},i}, r_{\mathtt{in},i}$ such that $\mathsf{acct}_{\delta,i}$ has the same value under commitments $\boxed{\mathtt{v_{bl}}}, \boxed{\mathtt{v_{in}}}$. Σ_{vds}^i can be implemented by using $\Sigma_{com}((\mathsf{pk}_{\delta,i}, \boxed{\mathtt{v_{bl}}}_i), (\mathsf{pk}_{\delta,i}, \boxed{\mathtt{v_{in}}}_i); (\mathtt{v_{bl}}_i, r_{\mathtt{bl},i}, r_{\mathtt{in},i}))$.
6. Σ_{range}: prover shows knowledge of $\mathsf{acct}_\epsilon, v, r$ such that $\mathsf{VerifyNonNegative}(\mathsf{acct}_\epsilon, v, r) = 1$. In order to implement this we use Bulletproofs [4].
7. Finally in order to prove $\mathsf{VerifyAcct}(\mathsf{acct}, \mathsf{sk}, \mathtt{bl})$:
   (a) the prover shows knowledge of sk using Σ_{dlog}.
   (b) Since sender may not know the randomness used to open his commitment, the prover opens the commitment with the sk and finds the value $\mathtt{bl}$.
   (c) Chooses a new randomness $r \leftarrow\$ \mathbb{F}_p$ and constructs $\mathsf{acct}_\epsilon = ((g, h), \mathsf{Commit}((g, h), \mathtt{bl}; r))$.
   (d) Proves using Σ_{com} that these two accounts has the same $\mathtt{bl}$.
   (e) Proves using $\Sigma_{range}(\mathsf{acct}_\epsilon, \mathtt{bl}, r)$ that $\mathtt{bl} \in \mathcal{V}$.
   So $\Sigma_{range,\mathsf{sk}} = \Sigma_{dlog} \ \wedge \ \Sigma_{com} \ \wedge \ \Sigma_{range}$.

Hence $\Sigma_2 = \Sigma_{vud} \wedge \Sigma_\delta \wedge \left(\wedge_{i=s+1}^t \Sigma_{range}(\mathsf{acct}_{\delta,i}, \mathtt{v}'_{\mathtt{bl}i}, r_{\mathtt{bl},i})\right) \wedge \left(\wedge_{i=t+1}^n \Sigma_{zero}^i\right) \wedge (\wedge_{i=1}^s \Sigma_{range,\mathsf{sk}}(\mathtt{outputs}'_i, \mathtt{bl}_i + \mathtt{v_{bl}}_i, \mathsf{sk})) \wedge \left(\wedge_{i=1}^s \Sigma_{vds}^i\right) \ \wedge \left(\wedge_{i=s+1}^t \Sigma_{vdr}^i\right)$. Σ_2 is a public-coin SHVZK argument of knowledge of the relation R_2 as follows from the properties of AND-proofs.

The full SHVZK argument knowledge of Trans is then $\Sigma := \Sigma_1 \wedge \Sigma_2 \wedge \Sigma_3$.

Transaction Verification The $\mathsf{VerifyTrans}(\mathsf{tx}, \mathsf{state})$ algorithm guarantees the validity of transaction tx. Depending on the transaction type $(\mathsf{tx}, \mathsf{tx_{CA}}, \mathsf{tx_{DA}})$ performs the following steps:

– if tx is an output of the Trans algorithm, then it first checks that all the accounts listed in $\mathsf{tx}.\mathtt{inputs}$ are deemed unspent in the current state, meaning for each $\mathsf{acct} \in \mathsf{tx}.\mathtt{inputs}, \mathsf{acct} \in \mathsf{state}.\mathsf{UTXOSet}$. Afterwards, it executes the verification algorithm for the NIZK argument π and returns its result.

- if $\mathsf{tx}_{\mathsf{CA}}$ is an output of the CreateAcct algorithm, then it first checks that all the userInfo listed in $\mathsf{tx}_{\mathsf{CA}}.\mathtt{inputs}$ are registered, meaning, for each $\mathsf{userInfo} \in \mathsf{tx}_{\mathsf{CA}}.\mathtt{inputs}$ we have that $\mathsf{userInfo} \in \mathsf{state.UserSet}$. It also ensures that $\mathsf{tx}_{\mathsf{CA}}.\mathsf{acct} \notin \mathsf{state.UTXOSet}$. Afterwards, it executes the verification algorithm for the NIZK argument π and returns its result.
- if $\mathsf{tx}_{\mathsf{DA}}$ is an output of the DeleteAcct algorithm, then it first checks that all the accounts listed in $\mathsf{tx}.\mathtt{inputs}_{\mathsf{UTXOSet}}$ belong to state.UTXOSet and similarly for $\mathtt{inputs}_{\mathsf{userInfo}}$. Afterwards, it executes the verification algorithm for the NIZK argument π and returns its result.

Create Account Algorithm Within the system every user can create a new account for any other registered user, which improves the efficiency of the system [10]. Since each account can appear only once as input in a transaction, if two concurrent transactions include the same account in their input set, one of them should be rejected. As the number of accounts within the system increases, the probability of a non-empty intersection between two transaction input sets decreases. In addition, creating new accounts allows users to own a fixed key that can be used to receive funds, instead of the key constantly changing. Therefore, it improves the overall communication overhead.

New accounts are composed of updates of the initial public key stored in the user's userInfo and commitments to zero values for the other attributes related to $\mathtt{bl}, \mathtt{out}, \mathtt{in}$. Moreover, userInfo is updated, by increasing the committed value for the number of accounts the user owns. This is achieved by using the homomorphic property of the commitment scheme.

In order to hide the userInfo that corresponds to the user, an anonymity set $\mathtt{A}$ is used. The values of the commitments of the userInfo that belong to the anonymity set are re-randomized without changing their committed values. That is, transactions that create new accounts are composed of input and output sets, which both include the intended user's userInfo, and also the newly created account. The userInfo updates happen through the invocation of the UpdateUser algorithm, and the outputs set is composed of these updated userInfo.

The detailed description of the CreateAcct algorithm is depicted in Figure 3. It takes as input the userInfo of the intended user and an anonymity set $\mathtt{A}$. It returns a transaction $\mathsf{tx}_{\mathsf{CA}} = (\mathsf{acct}, \mathtt{inputs}, \mathtt{outputs}, \pi)$.

The algorithm $\mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$ performs the following steps:

1. Pick $r_1, r_2, r_3, r_4 \leftarrow\$ \mathcal{R}$ and let $\vec{r} = (r_1, r_2, r_3, r_4)$. Let $\mathsf{acct} = (\mathsf{pk}, \boxed{0}, \boxed{0}, \boxed{0})$ be the output of $\mathsf{NewAcct}(\mathsf{userInfo.pk}_0; \vec{r})$.
2. Let $\mathtt{inputs} = \{\mathsf{userInfo}\} \cup \mathtt{A}$ in some canonical order. Let $\mathtt{c}, \mathtt{I}_\mathtt{A}$ be the indices of the chosen initial public key for which we wish to construct the new account, and the anonymity set in this list.
3. Construct $\vec{\mathtt{v}}$ as follows: $\mathtt{v}_i = 0\ \forall i \in \mathtt{I}_\mathtt{A}$ and $\mathtt{v}_\mathtt{c} = 1$.
4. Pick $r_5 \leftarrow\$ \mathcal{R}$ and let $\mathtt{outputs}$ be the output of $\mathsf{UpdateUser}(\mathtt{inputs}, \vec{\mathtt{v}}; r_5)$.
5. Form a zero-knowledge proof π of the relation $R(x, w)$, where $x = (\mathsf{acct}, \mathtt{inputs}, \mathtt{outputs}), w = (c, \vec{\mathtt{v}}, \vec{r}, r_5)$ and $R(x, w) = 1$ if $\forall i \in \{\mathtt{c}\} \cup \mathtt{I}_\mathtt{A},\ \mathsf{userInfo}_i \in \mathtt{inputs}, \mathsf{userInfo}'_i \in \mathtt{outputs}$ we have that:

$$\begin{aligned}&\mathsf{VerifyUpdateUser}(\mathsf{userInfo}'_i, \mathsf{userInfo}_i, 0; r_5) = 1\ \forall i \in \mathtt{I}_\mathtt{A}\\ &\wedge \mathsf{VerifyUpdateUser}(\mathsf{userInfo}'_\mathtt{c}, \mathsf{userInfo}_\mathtt{c}, 1; r_5) = 1\\ &\wedge \mathsf{VerifyUpdate}(\mathsf{acct.pk}, \mathsf{userInfo}_c.\mathsf{pk}_0, r_1) = 1\\ &\wedge \mathsf{Commit}(\mathsf{acct.pk}, 0; r_2) = \mathsf{acct.com}_{\mathtt{bl}}\\ &\wedge \mathsf{Commit}(\mathsf{acct.pk}, 0; r_3) = \mathsf{acct.com}_{\mathtt{out}} \wedge \mathsf{Commit}(\mathsf{acct.pk}, 0; r_4) = \mathsf{acct.com}_{\mathtt{in}}\end{aligned}$$

The final transaction returned by the algorithm is $\mathsf{tx}_{\mathsf{CA}} = (\mathsf{acct}, \mathtt{inputs}, \mathtt{outputs}, \pi)$.

Fig. 3. The CreateAcct algorithm.

Create Account Verification The $\mathsf{VerifyCreateAcct}(\mathsf{tx}_{\mathsf{CA}}, \mathsf{state})$ algorithm guarantees the validity of transaction $\mathsf{tx}_{\mathsf{CA}}$. First, it checks that all the userInfo listed in $\mathsf{tx}_{\mathsf{CA}}.\mathtt{inputs}$ are registered and compliant in the current state, meaning for each $\mathsf{userInfo} \in \mathsf{tx}_{\mathsf{CA}}.\mathtt{inputs}, \mathsf{userInfo} \in \mathsf{UserSet}$. Afterwards, it executes the verification algorithm for the NIZK argument π and returns its result.

Delete Account Algorithm Allowing users to delete zero-balance accounts reduces the storage overhead of AQQUA, since accounts that have no balance left to spend might be abandoned and thus not needed to be stored in the UTXOSet. Furthermore, due to the fact that senders usually create new accounts for their intended recipients, the number of accounts in the UTXOSet increases if the option to remove zero-balance accounts is not given. We note that users should be incentivized to delete the zero-balance accounts they own and don't need to keep. The mechanism to do so is left for future work.

In order to delete an account, the information containing the total amount $\mathtt{out}, \mathtt{in}$ sent and received by the account must be transferred to another account acct_C of the corresponding owner. In order to hide acct_C an anonymity set is included.

The detailed description of the $\mathsf{DeleteAcct}$ algorithm is depicted in Figure 4. The algorithm takes as input the secret key sk, the account to be deleted acct_D, the account acct_C to which $\mathtt{out}, \mathtt{in}$ of acct_D will be transferred, and anonymity sets $\mathtt{A}_1$ for the $\mathsf{UTXOSet}$ and $\mathtt{A}_2$ for the $\mathsf{UserSet}$ respectively. It returns a transaction $\mathsf{tx}_\mathsf{DA} = (\mathtt{inputs}, \mathtt{outputs}, \pi)$.

The algorithm $\mathsf{DeleteAcct}(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}_\mathsf{D}, \mathsf{acct}_\mathsf{C}, \mathtt{A}_1, \mathtt{A}_2)$ performs the following steps:

1. For the account acct_D, calculate the opening of the commitments $\mathsf{acct}_\mathsf{D}.\mathsf{com}_{\mathtt{out}}, \mathsf{acct}_\mathsf{D}.\mathsf{com}_{\mathtt{in}}$, denoted $\mathtt{out}_\mathsf{D}, \mathtt{in}_\mathsf{D}$, using the secret key sk.
2. Let $\mathtt{inputs}_{\mathsf{UTXOSet}} = \{\mathsf{acct}_\mathsf{C}\} \cup \mathtt{A}_1$ in some canonical order. Let $c^*, \mathtt{I}_{\mathtt{A}1}$ denote the indices of the account to be added the information and the accounts of the anonymity set in this list.
3. Construct $\overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}$ as follows:
   – $\overrightarrow{\mathtt{v_{bl}}} = 0\ \forall i \in \{c^*\} \cup \mathtt{I}_{\mathtt{A}1}$
   – $\overrightarrow{\mathtt{v_{out}}} = 0\ \forall i \in \mathtt{I}_{\mathtt{A}1}$ and $\mathtt{v_{out}}_{c^*} = \mathtt{out}_\mathsf{D}$
   – $\overrightarrow{\mathtt{v_{in}}} = 0\ \forall i \in \mathtt{I}_{\mathtt{A}1}$ and $\mathtt{v_{in}}_{c^*} = \mathtt{in}_\mathsf{D}$
4. Pick $r_1, r_2, r_3, r_4 \leftarrow\!\!\$\ \mathcal{R}$. and let $\vec{r} = (r_1, r_2, r_3, r_4)$. Let $\mathtt{outputs}_{\mathsf{UTXOSet}}$ be the output of $\mathsf{UpdateAcct}(\mathtt{inputs}_{\mathsf{UTXOSet}}, \overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}; \vec{r})$ in some canonical order.
5. Let $\psi : [\mathtt{n}] \to [\mathtt{n}]$ be the implicit permutation mapping $\mathtt{inputs}_{\mathsf{UTXOSet}}$ into $\mathtt{outputs}_{\mathsf{UTXOSet}}$; such that accounts $\mathsf{acct}_i \in \mathtt{inputs}_{\mathsf{UTXOSet}}$ and $\mathsf{acct}'_{\psi(i)} \in \mathtt{outputs}_{\mathsf{UTXOSet}}$ share the same secret key.
6. Form a zero-knowledge proof π_1 of the relation $R(x, w)$, where $x = (\mathsf{acct}_\mathsf{D}, \mathtt{inputs}_{\mathsf{UTXOSet}}, \mathtt{outputs}_{\mathsf{UTXOSet}}), w = (\mathsf{sk}, \mathtt{out}_\mathsf{D}, \mathtt{in}_\mathsf{D}, \vec{r}, \psi, c^*, \mathtt{I}_{\mathtt{A}1})$, and $R(x, w) = 1$ if

$$\begin{aligned}
&\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct}_\mathsf{D}.\mathsf{pk}) = 1 \wedge \mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct}_{c^*}.\mathsf{pk}) = 1\\
&\wedge \mathsf{VerifyUpdateAcct}(\mathsf{acct}'_{\psi(i)}, \mathsf{acct}_i, 0, 0, 0; \vec{r}) = 1\ \forall i \in \mathtt{I}_{\mathtt{A}1}\\
&\wedge \mathsf{VerifyUpdateAcct}(\mathsf{acct}'_{\psi(c^*)}, \mathsf{acct}_{c^*}, 0, \mathtt{out}_\mathsf{D}, \mathtt{in}_\mathsf{D}; \vec{r}) = 1\\
&\wedge \mathsf{VerifyCom}(\mathsf{acct}_\mathsf{D}.\mathsf{pk}, \mathsf{acct}_\mathsf{D}.\mathsf{com}_{\mathtt{bl}}, (\mathsf{sk}, 0)) = 1
\end{aligned}$$

7. Let $\mathtt{inputs}_{\mathsf{UserSet}} = \{\mathsf{userInfo}\} \cup \mathtt{A}_2$ in some canonical order. Let $s^*, \mathtt{I}_{\mathtt{A}2}$ denote the indices of the chosen initial public key for which we wish to construct the new account, and the anonymity set in this list.
8. Construct $\vec{\mathtt{v}}$ as follows: $\mathtt{v}_i = 0\ \forall i \in \mathtt{I}_{\mathtt{A}2}$ and $\mathtt{v}_{s^*} = -1$.
9. Pick $r \leftarrow\!\!\$\ \mathcal{R}$ and let $\mathtt{outputs}_{\mathsf{UserSet}}$ be the output of $\mathsf{UpdateUser}(\mathtt{inputs}_{\mathsf{UserSet}}, \vec{\mathtt{v}}; r)$.
10. Form a zero-knowledge proof π_2 of the relation $R(x, w)$, where $x = (\mathtt{inputs}_{\mathsf{UserSet}}, \mathtt{outputs}_{\mathsf{UserSet}}), w = (\mathsf{sk}, r, s^*, \mathtt{I}_{\mathtt{A}2})$ and $R(x, w) = 1$ if $\forall i \in \{s^*\} \cup \mathtt{I}_{\mathtt{A}2}$ $\mathsf{userInfo}_i \in \mathtt{inputs}_{\mathsf{UserSet}}, \mathsf{userInfo}'_i \in \mathtt{outputs}_{\mathsf{UserSet}}$ we have that:

$$\begin{aligned}
&\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{userInfo}_{s^*}.\mathsf{pk}_0) = 1\\
&\wedge \mathsf{VerifyUpdateUser}(\mathsf{userInfo}'_i, \mathsf{userInfo}_i, 0; r) = 1\ \forall i \in \mathtt{I}_{\mathtt{A}2}\\
&\wedge \mathsf{VerifyUpdateUser}(\mathsf{userInfo}'_{s^*}, \mathsf{userInfo}_{s^*}, -1; r) = 1
\end{aligned}$$

The final transaction returned by the algorithm is
$\mathsf{tx}_\mathsf{DA} = (\mathtt{inputs}_{\mathsf{UTXOSet}}, \mathtt{outputs}_{\mathsf{UTXOSet}}, \mathtt{inputs}_{\mathsf{UserSet}}, \mathtt{outputs}_{\mathsf{UserSet}}, \pi = (\pi_1, \pi_2))$.

Fig. 4. The $\mathsf{DeleteAcct}$ algorithm.

Delete Account Verification The $\mathsf{VerifyDeleteAcct}(\mathsf{tx}_\mathsf{CA}, \mathsf{state})$ algorithm guarantees the validity of transaction tx_DA. Firstly, it checks that all the accounts listed in $\mathsf{tx}.\mathtt{inputs}$ are part of the $\mathsf{UTXOSet}$. Afterwards, it executes the verification algorithm for the NIZK argument π and returns its result.

Apply Transaction The $\mathsf{ApplyTrans}(\mathsf{tx}, \mathsf{state})$ algorithm is executed after the verification of the transaction. It applies the transaction tx by updating the current state, adding $\mathsf{tx}.\mathtt{outputs}$ and removing $\mathsf{tx}.\mathtt{inputs}$.

– If tx is the result of the Trans algorithm, it updates only the $\mathsf{state.UTXOSet}$ with the new accounts.
– If tx is the result of the $\mathsf{CreateAcct}$ algorithm, it updates the $\mathsf{state.UserSet}$ and adds the newly created account in the $\mathsf{state.UTXOSet}$.
– If tx is the result of the $\mathsf{DeleteAcct}$ algorithm, it updates both $\mathsf{state.UserSet}$ and $\mathsf{state.UTXOSet}$.

Similarly to [10], upon receiving a new state, users whose accounts are included in a tranction's `inputs` should identify their updated accounts in `outputs`. This can be accomplished by iterating through every $\mathsf{acct} \in$ `outputs` and using $\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct.pk})$. Once the user identifies an updated account, they can check whether their account was used as part of the anonymity set or as a recipient, by running $\mathsf{VerifyCom}(\mathsf{sk}, \mathsf{acct.pk}, \mathsf{acct.com}_{\mathtt{bl}}, \mathtt{bl})$, passing as input the account's previous balance `bl`. If the result is 1, then the account was used as part of the anonymity set. Otherwise, the user must find out the new value for the balance. The value is small enough so that the computation of its discrete logarithm takes place in a reasonable time.

6.6 Audit

Audit Algorithm In the audit procedure, the `AA` selects a user by their initial public key $\mathtt{pk}_0$ and a time period which is represented by two snapshots of the blockchain $(\mathsf{state}_1, \mathsf{state}_2)$. For the policies that are applied to transactions (namely $f_{\mathsf{txlimit}}, f_{\mathsf{open}}$), the snapshot state_2 should be the state that results from applying the transaction to state_1. In the case where the policy is applied to a specified period (for example in $f_{\mathsf{slimit}}, f_{\mathsf{rlimit}}, f_{\mathsf{np}}$), the snapshots $\mathsf{state}_1, \mathsf{state}_2$ should be the states right before the beginning and after the end of the period, respectively.

The user which participates in the auditing should open for each of the two snapshots the committed value of the number of accounts they own ($\boxed{\texttt{\#accs}}$ field of $\mathsf{userInfo}$). Then, they should reveal their accounts in each of the two snapshots' $\mathsf{UTXOSet}$. The number of accounts they reveal in each snapshot should be equal to the opening of the corresponding commitment. Revealing the accounts does not hurt the anonymity of the user, since from the indistinguishability property of the UPK scheme and the hiding property of the commitment scheme, the `AA` cannot link the accounts that will be revealed with updated versions of them that will appear as a result of the user participating in any new transaction.

After opening the commitment and revealing the account, the user creates a zero-knowledge proof that the sets of accounts satisfy the required policy predicate, as defined in subsection 4.4.

The detailed description of the $\mathsf{PrepareAudit}$ algorithm is depicted in Figure 5. It takes as input the user's secret key sk, the two blockchain snapshots $(\mathsf{state}_1, \mathsf{state}_2)$, and the policy f along with the necessary information `aux`.

Both the $\mathsf{Register}$ and $\mathsf{PrepareAudit}$ functionalities need a zero-knowledge proof for the statements:

– $\mathsf{VerifyKP}(\mathsf{pk}, \mathsf{sk})$: prover shows knowledge of a valid $(\mathsf{pk}, \mathsf{sk})$ key-pair. The corresponding language can be written as:

$$L_{vu} := \{pk = (X = g^r, Y = g^{r \cdot sk}) \mid \exists \mathsf{sk} \text{ s.t. } \; Y = X^{\mathsf{sk}}\}$$

That can be proven through Σ_{dlog} with arguments (X, Y, sk).

– $\mathsf{VerifyCom}(\mathsf{pk}, \mathtt{com}, \mathsf{sk}, v)$: prover shows knowledge of secret key sk that opens the commitment `com` to value v. The corresponding language can be written as:

$$L_{open(sk)} := \{(\mathtt{com} = (X = h^r, Y = g^v h^{sk \cdot r}), v) \mid \exists \mathsf{sk} \text{ s.t. } \; Y/g^v = X^{\mathsf{sk}}\}$$

That can be proven through Σ_{dlog} with arguments $(X, Y/g^v, \mathsf{sk})$.

The proof needed for $\mathsf{Register}$ results from the composition of these Σ-protocols and a range proof for showing that $bl \in \mathcal{V}$.

The $\mathsf{PrepareAudit}$ proof uses the same combination of these Σ-protocols and appropriate range proofs for each policy $f_{\mathsf{slimit}}, f_{\mathsf{rlimit}}, f_{\mathsf{open}}, f_{\mathsf{txlimit}}, f_{\mathsf{np}}$.

Audit Verification The $\mathsf{VerifyAudit}$ algorithm is executed by the `AA` to check the compliance of the user with a specific policy. Initially, the algorithm checks that the user has revealed `#accs` accounts that belongs to each selected snapshot, calculate the necessary values (i.e. multiplication of committed amounts), and then runs the verification algorithm for the NIZK argument π and returns its result.

7 Security Analysis

7.1 Anonymity

Intuitively, we argue that any PPT adversary $\mathcal{A}$ capable of distinguishing between $\mathsf{tx}_0, \mathsf{tx}_1$ in the anonymity game (find if $b' = b$) can be used to break either the indistinguishability of UPK scheme, the hiding property of commitment scheme, or the zero-knowledge property of the NIZK proofs.

Transactions consist of `inputs`, `outputs`, and a zk-proof π (and if it is $\mathsf{CreateAcct}$ or $\mathsf{DeleteAcct}$ a newly created account acct). One way $\mathcal{A}$ could determine b is based on π, but that violates the zero-knowledge property

The algorithm $\mathsf{auditInfo} \leftarrow \mathsf{PrepareAudit}(\mathsf{sk}, \mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}))$ performs the following steps:

1. Ensure that $\mathsf{VerifyKP}(\mathsf{sk}, \mathtt{pk}_0)$. For each snapshot $\mathsf{state}_j, j = 1, 2$ find the $\mathsf{userInfo}_j$ that contains $\mathtt{pk}_0$, and calculate $\mathtt{\#accs}_j = \mathsf{OpenCom}(\mathsf{sk}, \mathsf{userInfo}.\boxed{\mathtt{\#accs}_j})$
2. For each snapshot find the set of accounts $A_j = \{\mathsf{acct}_i\}_{i=1}^{\mathtt{\#accs}_j}$ that belong to the user. That is, $\forall \mathsf{acct} \in \mathsf{state}_j.\mathsf{UTXOSet}$, if $\mathsf{VerifyKP}(\mathsf{acct}.\mathsf{pk}, \mathsf{sk}) = 1$, then add acct to A_j.
3. Form a zero-knowledge proof π_1 of the relation $R(x, w)$, where $x = (\mathtt{pk}_0, \{\mathtt{\#accs}_j, \boxed{\mathtt{\#accs}_j}, \{\mathsf{acct}_{ji}\}_{i=1}^{\mathtt{\#accs}_j}\}_{j=1}^{2}), w = (\mathsf{sk})$ and $R(x, w) = 1$ if:

$$\begin{aligned}
&\mathsf{VerifyCom}(\mathtt{pk}_0, \boxed{\mathtt{\#accs}_j}, (\mathsf{sk}, \mathtt{\#accs}_j)) = 1 \ \forall j \in \{1, 2\}\\
&\wedge \mathsf{VerifyKP}(\mathtt{pk}_0, \mathsf{sk}) = 1\\
&\wedge \mathsf{VerifyKP}(\mathsf{acct}_{ji}.\mathsf{pk}, \mathsf{sk}) = 1 \ \forall i \in \{1, \dots, \mathtt{\#accs}_j\}, \ \forall j \in \{1, 2\}
\end{aligned}$$

If $f \in \{f_{\mathsf{slimit}}, f_{\mathsf{rlimit}}, f_{\mathsf{np}}\}$ then:

4. For each snapshot calculate $\boxed{\mathtt{out}_j^*} = \prod_{i=1}^{\mathtt{\#accs}_j} \mathsf{acct}_{ji}.\boxed{\mathtt{out}}, \boxed{\mathtt{in}_j^*} = \prod_{i=1}^{\mathtt{\#accs}_j} \mathsf{acct}_{ji}.\boxed{\mathtt{in}}$.

   Then calculate $\boxed{\mathtt{out}^*} = \boxed{\mathtt{out}_2^*} \cdot \left(\boxed{\mathtt{out}_1^*}\right)^{-1}, \boxed{\mathtt{in}^*} = \boxed{\mathtt{in}_2^*} \cdot \left(\boxed{\mathtt{in}_1^*}\right)^{-1}$.

   Finally, calculate $\mathtt{out}^* = \mathsf{OpenCom}(\mathsf{sk}, \boxed{\mathtt{out}^*}), \mathtt{in}^* = \mathsf{OpenCom}(\mathsf{sk}, \boxed{\mathtt{in}^*})$. These values represent the total amount of coins that the user spent/received in the selected period of time.
5. Form a zero-knowledge proof π_2 of the relation $R(x, w)$ where $x = (\{\mathsf{acct}_{1i}\}_{i=1}^{\mathtt{\#accs}_j}, \{\mathsf{acct}_{2i}\}_{i=1}^{\mathtt{\#accs}_j}, \boxed{\mathtt{out}^*}, \boxed{\mathtt{in}^*}, \mathtt{aux}), w = (\mathtt{out}^*, \mathtt{in}^*)$ and $R(x, w) = 1$ if:

$$f(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), \mathtt{aux}) = 1$$

If $f \in \{f_{\mathsf{txlimit}}, f_{\mathsf{open}}\}$ then:

4. For each snapshot calculate $\boxed{\mathtt{bl}_j^*} = \prod_{i=1}^{\mathtt{\#accs}_j} \mathsf{acct}_{ji}.\boxed{\mathtt{bl}}$. Then calculate $\boxed{\mathtt{bl}^*} = \boxed{\mathtt{bl}_2^*} \cdot \left(\boxed{\mathtt{bl}_1^*}\right)^{-1}$ and $\mathtt{bl}^* = \mathsf{OpenCom}(\mathsf{sk}, \boxed{\mathtt{bl}^*})$.
5. Form a zero-knowledge proof π_2 of the relation $R(x, w)$ where $x = (\{\mathsf{acct}_{1i}\}_{i=1}^{\mathtt{\#accs}_j}, \{\mathsf{acct}_{2i}\}_{i=1}^{\mathtt{\#accs}_j}, \boxed{\mathtt{bl}^*}, \mathtt{aux}), w = (\mathtt{bl}^*)$ and $R(x, w) = 1$ if:

$$f(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), \mathtt{aux}) = 1$$

The final output is $\mathsf{auditInfo} = (\pi = (\pi_1, \pi_2), \mathtt{\#accs}_1, \{\mathsf{acct}_{1i}\}_{i=1}^{\mathtt{\#accs}}, \mathtt{\#accs}_2, \{\mathsf{acct}_{2i}\}_{i=1}^{\mathtt{\#accs}})$.

Fig. 5. The PrepareAudit algorithm.

of the NIZK proofs. Another way that $\mathcal{A}$ could determine b is to distinguish between $\mathsf{tx}_0, \mathsf{tx}_1$ through the `outputs` sets of each tx. The only differences in the two `outputs` sets tx_0.`outputs`, tx_1.`outputs` are the accounts which are used in P and in A as well as the amount v used to increase/decrease the variables in the accounts of P. However, since both the accounts' amounts and transferred value v are presented in a committed form, if $\mathcal{A}$ can determine b based on the different values v_0, v_1 then the hiding property of the commitment scheme is violated. In addition, since all the accounts participating in the transaction are updated and randomly permuted, $\mathcal{A}$ cannot use $\mathtt{P}_0, \mathtt{A}_0, \mathtt{P}_1, \mathtt{A}_1$ to distinguish the two transactions without violating the indistinguishability property of UPK scheme.

Before we give the proof of anonymity, we first recall a definition for indistinguishability of UPK scheme [10].

Definition 4. *The* advantage *of the adversary in winning the indistinguishability game is defined as:*

$$\mathsf{Adv}^{\mathrm{ind}}_{\mathcal{A}}(\lambda) =\mid \Pr[\mathsf{Exp}^{ind}_{\mathcal{A}}((\lambda)) = 1] - \frac{1}{2} \mid$$

A DPS satisfies indistinguishability *if for every PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{ind}}_{\mathcal{A}}(\lambda)$ is negligible in λ.*

Game 1.6: Indistinguishability game $\mathsf{Exp}^{ind}_{\mathcal{A}}(\lambda)$

```
Input  : $\lambda$
Output: $\{0,1\}$
$b \leftarrow \{0,1\}$
$(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KGen}()$
$r \leftarrow\$ \mathcal{R}$
$\mathsf{pk}_0 \leftarrow \mathsf{Update}(\mathsf{pk}^*; r)$
$(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KGen}()$
$b' \leftarrow \mathcal{A}(\mathsf{pk}^*, \mathsf{pk}_b)$
return $(b = b')$
```

Note that in indistinguishability game the challenger can update many times the pk^* before creating pk_0 due to the fact that even with more updates the pk_0 can be described as an update of pk^* with a different randomness.

Lemma 1. *The constructed UPK scheme satisfies 4 if the DDH assumption holds in $(\mathbb{G}, g, p)$.*

Proof of this lemma can be found in [10].

Theorem 1. *AQQUA satisfies anonymity, as defined in Definition 1*

Proof. We prove the theorem using a sequence of 14 hybrid games, as follows. Hybrid 0 and Hybrid 7 are the anonymity game for $b = 0, b = 1$ respectively. Each of the rest hybrids differs in oracles' functionalities in a way that the successive hybrids are indistinguishable from the view of the adversary. We use these hybrids to prove that the adversary cannot distinguish anonymity game for $b = 0$ and anonymity game with $b = 1$.
Hybrid 0. The anonymity game for $b = 0$.
Hybrid 1. Same as Hybrid 0, but here we run the NIZK extractor on each transaction generated by the adversary. That means, when $\mathcal{A}$ runs the $\mathsf{OApplyTrans}(\mathsf{tx})$ Oracle, the Oracle verifies tx by running $\mathsf{VerifyTrans}(\mathsf{tx}, \mathsf{state})$ depending on the transaction tx and if it is successful the oracle runs $\mathsf{state}' \leftarrow \mathsf{ApplyTrans}(\mathsf{state}, \mathsf{tx})$, as well as uses the NIZK extractor to extract the witness used to generate tx, including sk.
Hybrid 2. Same as Hybrid 1, but here the zero-knowledge arguments of the each transaction is replaced with the output of the corresponding simulator of the zero-knowledge property of NIZK. In order to achieve this we change the following oracles' functionality:

– when $\mathcal{A}$ or the challenger creates tx through the $\mathsf{OTrans}(\mathtt{S}, \mathtt{R}, \vec{v_{\mathtt{S}}}, \vec{v_{\mathtt{R}}}, \mathtt{A})$ Oracle, the Oracle runs $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \vec{v_{\mathtt{S}}}, \vec{v_{\mathtt{R}}}, \mathtt{A})$, but replaces the zero-knowledge arguments in tx with a simulated argument.

– when $\mathcal{A}$ or the challenger creates tx through the $\mathsf{OCreateAcct}(\mathsf{userInfo}, \mathtt{A})$ Oracle, the Oracle runs $\mathsf{tx} \leftarrow \mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$, but replaces the zero-knowledge arguments in tx with a simulated argument.

Hybrid 3. Same as Hybrid 2, but now the challenger replaces the potential senders' and receivers' accounts of the challenge transaction tx_0 $(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1)$, with new accounts that have a freshly created key pair $(\mathsf{sk}, \mathsf{pk})$ derived from the output of the $\mathsf{KGen}()$. In order to achieve this we change the following oracles' functionality:

- when $\mathcal{A}$ creates one of these accounts acct_i through the OTrans Oracle (these accounts are presented in $\mathtt{tx.outputs}$), the Oracle runs $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}, \mathtt{A})$, $(\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{KGen}$ and then return tx', where $\mathsf{tx}' = \mathsf{tx}$ except that each $\mathsf{acct}_i \in \{\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1\}$ is replaced with $\mathsf{acct}'_i = (\mathsf{pk}'_i, \mathtt{com}_{\mathtt{bl}i}, \mathtt{com}_{\mathtt{out}i}, \mathtt{com}_{\mathtt{in}i})$.

- when $\mathcal{A}$ creates one of these accounts acct_i through the $\mathsf{OCreateAcct}$ Oracle, the Oracle runs $\mathsf{tx} \leftarrow \mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$, $(\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{KGen}$ and then return tx', where $\mathsf{tx}' = \mathsf{tx}$ except that each $\mathsf{acct}_i \in \{\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1\}$ is replaced with $\mathsf{acct}'_i = (\mathsf{pk}'_i, \boxed{0}, \boxed{0}, \boxed{0})$.

Hybrid 4. Same as Hybrid 3, but here the challenger replaces also the commitments of the accounts $(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1)$ with newly created commitments to the same values with different randomness. In order to achieve this we change the following oracles' functionality:

- when $\mathcal{A}$ creates one of these accounts acct_i through the OTrans Oracle (these accounts are presented in $\mathtt{tx.outputs}$), the Oracle runs $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}, \mathtt{A})$, $(r_1, r_2, r_3) \leftarrow\!\$\ \mathcal{R}$, $\mathtt{bl}_i \leftarrow \mathsf{OpenCom}(\mathsf{sk}, \mathsf{acct}_i.\mathtt{com}_{\mathtt{bl}})$, $\mathtt{out}_i \leftarrow \mathsf{OpenCom}(\mathsf{sk}, \mathsf{acct}_i.\mathtt{com}_{\mathtt{out}})$, $\mathtt{in}_i \leftarrow \mathsf{OpenCom}(\mathsf{sk}, \mathsf{acct}_i.\mathtt{com}_{\mathtt{in}})$, $\mathtt{com}'_{\mathtt{bl}} \leftarrow \mathsf{Commit}(\mathsf{pk}', \mathtt{bl}_i; r_1)$, $\mathtt{com}'_{\mathtt{out}} \leftarrow \mathsf{Commit}(\mathsf{pk}', \mathtt{out}_i; r_2)$, $\mathtt{com}'_{\mathtt{in}} \leftarrow \mathsf{Commit}(\mathsf{pk}', \mathtt{in}_i; r_3)$ and then return tx', where $\mathsf{tx}' = \mathsf{tx}$ except that each $\mathsf{acct}_i \in \{\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1\}$ is replaced with $\mathsf{acct}' = (\mathsf{pk}, \mathtt{com}'_{\mathtt{bl}}, \mathtt{com}'_{\mathtt{out}}, \mathtt{com}'_{\mathtt{in}})$. $(\mathsf{pk} = \mathsf{pk}'$ as in the Hybrid 3).

- when $\mathcal{A}$ creates one of these accounts acct_i through the $\mathsf{OCreateAcct}$ Oracle, the Oracle runs $\mathsf{tx} \leftarrow \mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$, $(r_1, r_2, r_3) \leftarrow\!\$\ \mathcal{R}$, $\mathtt{com}'_{\mathtt{bl}} \leftarrow \mathsf{Commit}(\mathsf{pk}'0; r_1)$, $\mathtt{com}'_{\mathtt{out}} \leftarrow \mathsf{Commit}(\mathsf{pk}', 0; r_2)$, $\mathtt{com}'_{\mathtt{in}} \leftarrow \mathsf{Commit}(\mathsf{pk}', 0; r_3)$ and then return tx', where $\mathsf{tx}' = \mathsf{tx}$ except that each $\mathsf{acct}_i \in \{\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1\}$ is replaced with $\mathsf{acct}' = (\mathsf{pk}, \mathtt{com}'_{\mathtt{bl}}, \mathtt{com}'_{\mathtt{out}}, \mathtt{com}'_{\mathtt{in}})$. $(\mathsf{pk} = \mathsf{pk}'$ as in the Hybrid 3).

Hybrid 5. Same as Hybrid 4, but here also the updated accounts of $(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1)$ in the challenge $\mathtt{tx.outputs}$ are replaced by accounts with freshly created public key pk'.
Hybrid 6. Same as Hybrid 5, but here also the updated accounts of $(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1)$ in the challenge $\mathtt{tx.outputs}$ are replaced by accounts with freshly created commitments to the same value.

Afterwards, we create Hybrids 7-13 that are the same with Hybrids 0-6 with the difference that are made for the anonymity game with $b = 1$.

Note that in Hybrid 6 and in Hybrid 13 all accounts of the potential senders' and receivers' accounts of the challenge transaction tx_b (both in $\mathtt{inputs}$ and $\mathtt{outputs}$) are fresh accounts, where in $\mathtt{outputs}$ have been generated with values corresponding to the case $b = 0$ — $b = 1$.

Now we will prove that $\mathcal{A}$ has negligible advantage of distinguish Hybrid 0 and Hybrid 7.

Lemma 2. *Hybrid 0 and Hybrid 1 are indistinguishable.*

Corollary 1. *Hybrid 7 and Hybrid 8 are indistinguishable.*

Proof. The adversary's view in the two hybrids' game are identical.

Lemma 3. *Hybrid 1 and Hybrid 2 are indistinguishable.*

Corollary 2. *Hybrid 8 and Hybrid 9 are indistinguishable.*

Proof. Let $\mathcal{A}$ be an adversary that can distinguish Hybrid 1 and Hybrid 2 with advantage ϵ. We construct an adversary $\mathcal{B}$ that breaks the zero-knowledge property of the NIZK proof π of transaction tx with probability ϵ.

Let $\mathsf{O_{zk}}(\cdot)$ be an oracle that on input $(\mathtt{tx.inputs}, \mathtt{tx.outputs})$ creates a valid zero-knowledge proof for the transaction. Then $\mathcal{B}$ wins if they can decide wether $\mathsf{O_{zk}}(\cdot)$ is a prover or simulator oracle.

$\mathcal{B}$ takes as input the $\mathsf{O_{zk}}(\cdot)$ and runs as follows:

1. $\mathcal{B}$ generates $\mathsf{state} \leftarrow \mathsf{Setup}(\lambda)$;
2. When $\mathcal{A}$ queries the $\mathsf{OTrans}(\mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}, \mathtt{A})$ oracle then $\mathcal{B}$ runs $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}, \mathtt{A})$ with the difference that $\mathcal{B}$ replace the proof with the output of $\mathsf{O_{zk}}(\mathsf{tx}[\mathtt{inputs}], \mathsf{tx}[\mathtt{outputs}])$
3. When $\mathcal{A}$ queries the $\mathsf{OCreateAcct}(\mathsf{userInfo}, \mathtt{A})$ oracle then $\mathcal{B}$ runs $\mathsf{tx} \leftarrow \mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$ with the difference that $\mathcal{B}$ replace the proof with the output of $\mathsf{O_{zk}}(\mathsf{tx}[\mathtt{inputs}], \mathsf{tx}[\mathtt{outputs}])$
4. $\mathcal{B}$ runs $b \leftarrow \mathcal{A}(state)$;

If $\mathcal{A}$ answers Hybrid 0 then $\mathsf{O_{zk}}(\cdot)$ is a prover oracle. If $\mathcal{A}$ answers Hybrid 1 then $\mathsf{O_{zk}}(\cdot)$ is a simulator oracle. So $\mathcal{B}$ wins with probability ϵ.

Lemma 4. *Hybrid 2 and Hybrid 3 are indistinguishable.*

Corollary 3. *Hybrid 9 and Hybrid 10 are indistinguishable.*

Proof. Note that $\mathcal{A}$ cannot distinguish Hybrid 2 and Hybrid 3 from the fact that commitments are under different public key on the grounds that this breaks the key-anonymous property of the commitment scheme. Let $\mathcal{A}$ be an adversary that can distinguish Hybrid 2 and Hybrid 3 with advantage ϵ. We construct an adversary $\mathcal{B}$ that breaks the indistinguishability property of the UPK scheme with probability ϵ.

In order to create $\mathcal{B}$, we define five sub-hybrids. Let h_0 be Hybrid 2 and for each $i \in \{1, 2, 3, 4\}$ h_i would be a sub-hybrid where we replace the account $\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1$ respectively. In hybrid h_4 all of the accounts will be changed, therefore h_4 is Hybrid 3. Lets $\mathcal{A}$ be an adversary that can distinguish h_i from h_{i+1}. Let acct_c be the account that we are replacing in this hybrid. Then:
$\mathcal{B}$ gets as input the tuple $(\mathsf{acct}^*, \mathsf{acct}_b)$ from the indistinguishability game and runs as follows:

1. $\mathcal{B}$ generates $\mathsf{state} \leftarrow \mathsf{Setup}(\lambda)$.
2. when $\mathcal{A}$ uses the $\mathsf{ORegister}$ Oracle to create the initial account that share the same secret key with acct_c, $\mathcal{B}$ replaces this account with acct^*.
3. when $\mathcal{A}$ uses OTrans or $\mathsf{OCreateAcct}$ Oracle to create the account acct_c, $\mathcal{B}$ replaces acct_c with acct_b.
4. $\mathcal{B}$ reply to all other queries in the oracles as in the Hybrid h_0.
5. $\mathcal{B}$ outputs $b' \leftarrow \mathcal{A}(\mathsf{state})$.

We know that $\mathcal{A}$ did not query the corrupt oracle on acct_c or on any other account that shares the same secret key with acct_c cause it would have immediately lost the anonymity game. Note that if $b = 0$ then the distribution of the game is the same as hybrid h_i and if $b = 1$ then the game has the same distribution as hybrid h_{i+1}. Hence $\mathcal{B}$ answer b' and solves the indistinguishability game with probability ϵ.

Lemma 5. *Hybrid 3 and Hybrid 4 are indistinguishable.*

Corollary 4. *Hybrid 10 and Hybrid 11 are indistinguishable.*

Proof. The only difference from this two Hybrids are the randomness to the commitments of the real participants accounts. Therefore, they produce a computationally indistinguishable distribution, due to the hiding property if the used commitment scheme.

Corollary 5. *Hybrid 4 and Hybrid 5 are indistinguishable.*
Hybrid 11 and Hybrid 12 are indistinguishable.
It can be proven the same way as Hybrid 2 and Hybrid 3 are indistinguishable.

Corollary 6. *Hybrid 5 and Hybrid 6 are indistinguishable.*
Hybrid 12 and Hybrid 13 are indistinguishable.
It can be proven the same way as Hybrid 3 and Hybrid 4 are indistinguishable.

Lemma 6. *Hybrid 6 and Hybrid 13 are indistinguishable.*

Proof. Hybrid 6 and Hybrid 13 differ to (1) the accounts that are included in $\mathtt{P}$ and in $\mathtt{A}$ as well as to (2) the balances that are stored in the real participants' accounts in the challenge query $(\mathsf{acct}_i = \in \{\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1\})$. Concerning the former (1), in both Hybrids the `inputs` that $\mathcal{A}$ sees is obtained by permuting $(\mathtt{P}_x \cup \mathtt{A}_x)$ with a random permutation ψ. But the union of these set in both cases $(x = \{0, 1\})$ produces identical distributions. As a result $\mathcal{A}$ cannot distinguish the two Hybrids from (1). The second change (2) produces a computationally indistinguishable distribution, due to the hiding property of the commitment scheme. Therefore, if $\mathcal{A}$ could distinguish these Hybrids based on (2) then $\mathcal{A}$ could break the hiding property of Commit.

Using the above lemmas and the triangle inequality, we prove that there is not a PPT adversary $\mathcal{A}$ that can distinguish Hybrid 0 and Hybrid 7 with more than negligible advantage.

7.2 Theft prevention

Intuitively, we argue that any PPT adversary $\mathcal{A}$ capable of winning the theft-prevention game can be used to break either the unforgeability property of UPK scheme, the binding property of commitment scheme, or the soundness property of the NIZK proofs.

In order to win the theft-prevention game, $\mathcal{A}$ should submit a transaction tx that either increases the total balance of the corrupted users, decreases the balance of honest users, or does not maintain preservation of value. This can happen in the following ways: The first way is if the adversary is able to transfer some amount from a honest user's account. However, this means that $\mathcal{A}$ can compute the sk of the honest account, thus the unforgeability property of the UPK scheme is violated. Secondly, if $\mathcal{A}$ manages to transfer more coins than the corrupted account holds. But in order for such a transaction to be valid, the adversary should either be able to make a zk-proof that violates the soundness property, or to compute an opening to a commitment with balance different from the real one, hence breaking the binding property of the commitment scheme. The third way is by creating a transaction that breaks preservation of value, but in order for such a transaction to be valid, $\mathcal{A}$ should again be able to construct an unsound zk-proof or break the binding property of the commitment scheme.

Theorem 2. *AQQUA satisfies theft prevention, as defined in Definition 2.*

Proof. Assume that there exists a PPT $\mathcal{A}$ that wins the theft prevention game of Game 1.4 with non-negligible probability. Using the notation of the game, we have that $\mathcal{A}$ outputted a valid transaction tx that verifies and that results in one of the three winning conditions of the game being satisfied.

We have that $\mathsf{tx} = (\mathtt{inputs}, \mathtt{outputs}, \pi)$, where π is a ZK-proof for the relation $R(x, w)$ as defined in Figure 2, with $x = (\mathtt{inputs}, \mathtt{outputs})$ and $w = (\mathsf{sk}, \mathtt{bl}, \mathtt{out}, \mathtt{in}, \overrightarrow{\mathtt{v_{bl}}}, \overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}, \vec{r}, \psi, \mathtt{I}^*_{\mathtt{S}}, \mathtt{I}^*_{\mathtt{R}}, \mathtt{I}^*_{\mathtt{A}})$.

From the soundness property of the NIZK argument of the Trans algorithm, we can extract a witness $w^* = (\mathsf{sk}^*, \mathtt{bl}^*, \cdots, \overrightarrow{\mathtt{v}^{*\prime}_{\mathtt{bl}}}, \cdots, \overrightarrow{r^*}, \cdots)$ such that $R(x, w^*) = 1$.

Let $\mathsf{acct} \in \mathtt{inputs}$ be the account such that $\mathsf{VerifyKP}(\mathsf{sk}^*, \mathsf{acct.pk}) = 1$. We divide into two cases.

1. It holds that $\mathsf{sk}^* \in \mathsf{honest}$. In this case, we construct an adversary $\mathcal{B}$ that breaks the unforgeability property of the UPK scheme with non-negligible probability.
   The reduction works as follows. The adversary $\mathcal{B}$ takes as input a public key pk^*. It also keeps a directed tree with root $(\mathsf{pk}^*, 1)$ and whose nodes will be tuples of the form (pk, r). The tree will be updated so that for every edge of the form $((\mathsf{pk}_1, \cdot), (\mathsf{pk}_2, r_2))$ it will hold that $\mathsf{VerifyUpdate}(\mathsf{pk}_2, \mathsf{pk}_1, r_2) = 1$.
   $\mathcal{B}$ answers to $\mathcal{A}$'s oracle queries as follows.
   - When $\mathcal{A}$ queries the $\mathsf{ORegister}$ oracle and this query results in the $\mathsf{Register}$ algorithm to generate sk^*, $\mathcal{B}$ replaces $\mathsf{userInfo.pk}_0$ with pk^*, and when $\mathsf{NewAcct}$ is called in the procedure, $\mathcal{B}$ gives as input pk^*. The adversary $\mathcal{B}$ stores the public key of the newly created account and the randomness used as a child of $(\mathsf{pk}^*, 1)$ in the tree. For the rest of the $\mathsf{ORegister}$ queries, $\mathcal{B}$ answers honestly.
   - When $\mathcal{A}$ queries the $\mathsf{OCreateAcct}$ oracle for an account whose public key pk is contained in a leaf of the tree, $\mathcal{B}$ answers honestly and adds a child to the leaf, composed of the updated public key of the updated account and the randomness used.
   - When $\mathcal{A}$ queries the OTrans oracle, the adversary $\mathcal{B}$ acts as follows.
     * If the public keys of the accounts in $\mathtt{S}$ are contained in leaves of the tree, $\mathcal{B}$ creates an outputs set and creates a simulated proof for the transaction. $\mathcal{B}$ also updates the tree by creating new children containing the updates of the public keys and the randomness.
     * If there exist public keys of accounts in the anonymity set that are contained in leaves of the tree, $\mathcal{B}$ creates new children containing the updates of the public keys and the randomness.
   - When $\mathcal{A}$ queries the $\mathsf{OApplyTrans}$ with a transaction whose inputs contain a leaf of the tree, $\mathcal{B}$ uses the proof contained in the transaction to extract the witness. Then, $\mathcal{B}$ creates new children for the updates of the public keys, storing also the randomness of the witness.
   - For the rest of the oracle queries, $\mathcal{B}$ answers honestly.

   Finally, when $\mathcal{A}$ outputs the transaction tx of the theft prevention game, $\mathcal{B}$ finds the $\mathsf{acct} \in \mathtt{inputs}$ for which $\mathsf{VerifyKP}(\mathsf{sk}^*, \mathsf{acct.pk}) = 1$, and finds the leaf (pk, r) of the tree for which $\mathsf{acct.pk} = \mathsf{pk}$. Let r' be the multiplication of all randomnesses stored in the path from that leaf to the root. $\mathcal{B}$ returns (pk, r').
   If $\mathcal{A}$ wins the theft prevention game, we have that $\mathsf{VerifyKP}(\mathsf{pk}, \mathsf{sk}^*) = 1$ and $\mathsf{VerifyUpdate}(\mathsf{pk}, \mathsf{pk}^*, r') = 1$. Since $\mathcal{A}$ can win with non-negligible probability, $\mathcal{B}$ breaks unforgeability with non-negligible probability.
2. It holds that $\mathsf{sk}^* \in \mathsf{corrupt}$.
   Assume w.l.o.g. that the transaction tx that $\mathcal{A}$ outputs is the first transaction that results in winning the game (that is, there is no transaction submitted to $\mathsf{OApplyTrans}$ oracle prior to this point that would result in $\mathcal{A}$ winning).
   Since $\mathcal{A}$ wins the game, we have that the sum of the openings of the committed balances of all the accounts (stored in the bookkeeping) of $\mathtt{inputs}$ is different from those of $\mathtt{outputs}$.
   From the soundness property of the NIZK argument of the Trans algorithm, we have that for every sender account acct' of $\mathtt{outputs}$, $\mathsf{VerifyAcct}(\mathsf{acct}', \mathsf{sk}^*, \mathtt{bl}^* + \mathtt{v}'^*_{\mathtt{bl}}, \cdot, \cdot) = 1$.
   Since $\mathsf{VerifyAcct}$ returns 1, and also $\sum_{\mathtt{v}'^*_{\mathtt{bl}} \in \overrightarrow{\mathtt{v}'^*_{\mathtt{bl}}}} \mathtt{v}'^*_{\mathtt{bl}} = 0$, and since $\mathcal{A}$ wins the game, there exists an account $\mathsf{acct} \in \mathtt{outputs}$ for which $\mathsf{acct.com}_{\mathtt{bl}}$ has two different openings: one resulting from the bookkeeping, and one derived from the extracted witness (one of the values of the form $\mathtt{bl}^* + \mathtt{v}'^*_{\mathtt{bl}}$ for some sender account). This trivially breaks the binding property of the commitment scheme.

7.3 Audit soundness

Intuitively, we argue that any PPT adversary $\mathcal{A}$ capable of winning the audit soundness game can be used to break either the binding property of commitment scheme or the soundness property of the NIZK proofs.

In order to win the the audit soundness game, $\mathcal{A}$ should either create a valid zero-knowledge proof without knowing the corresponding witness, or hide some of their accounts from the AA. However, the former attack violates the soundness property of the zero-knowledge proof. The latter requires the $\mathcal{A}$ to be able to open their commitment $\boxed{\texttt{\#accs}}$ to a different value, but this breaks again the binding property of the commitment scheme.

Theorem 3. *AQQUA satisfies audit soundness, as of Definition 3*

Proof. Assume that there exist a PPT $\mathcal{A}$ that wins the audit soundness game of Game 1.5 with non-negligible probability. Using the notation of the game, we have that $\mathcal{A}$ outputted a proof $\pi = (\pi_1, \pi_2)$ that verifies but $\mathcal{A}$ is not compliant with the specified policy.

$\mathcal{A}$ choose a policy f with its auxiliary parameters $\texttt{aux}$, an initial public key $\texttt{pk}_0$ and two snapshots from the blockchain $\mathsf{state}_1, \mathsf{state}_2$. Then $\mathcal{A}$ constructs $\pi = (\pi_1, \pi_2)$ which as defined in Figure 5 is a ZK-proof for the relations $R_1(x, w)$, with $x = (\texttt{pk}_0, \{\texttt{\#accs}_j, \boxed{\texttt{\#accs}_j}, \{\mathsf{acct}_{ji}\}_{i=1}^{\texttt{\#accs}_j}\}_{j=1}^{2})$ and $w = (\mathsf{sk})$ and $R_2(x, w)$, with $x = (\{\mathsf{acct}_{1i}\}_{i=1}^{\texttt{\#accs}_j}, \{\mathsf{acct}_{2i}\}_{i=1}^{\texttt{\#accs}_j}, \boxed{\texttt{v}}, \texttt{aux})$ and $w = (\mathsf{sk}, \texttt{v})$, where $\texttt{v}, \texttt{aux}$ are values that depend on the policy.

From the soundness property of the NIZK argument of the π_1, we can extract a witness $w^* = \mathsf{sk}^*$ such that $R_1(x, w^*) = 1$. We have that every $\mathsf{pk} \in \{\texttt{pk}_0\} \cup \{\mathsf{acct}_{ji}.\mathsf{pk}\}_{i=1}^{\texttt{\#accs}_j}$, $\mathsf{VerifyKP}(\mathsf{sk}^*, \mathsf{pk})$. Therefore similarly to theft-prevention proof we can prove that if $\mathsf{sk}^* \in \mathsf{honest}$ then $\mathcal{A}$ can be used to break the unforgeability property of UPK scheme. Else if $\mathsf{sk}^* \in \mathsf{corrupt}$ then since $\mathcal{A}$ wins the game, we have that the opening to the commitment of $\boxed{\texttt{\#accs}}$ is different from the one that resulting from bookkeeping. This trivially breaks the binding property of the commitment scheme.

From the soundness property of the NIZK argument of the π_2, we can extract a witness $w^* = \texttt{v}^*$ such that $R_1(x, w^*) = 1$. Again since $\mathcal{A}$ wince the game the sum of the openings of the commited value of all the accounts that belongs to $\mathcal{A}$ is different from the one that resulting from bookkeeping, so this breaks the binding property of the commitment scheme.

8 Conclusion and Future Work

We presented AQQUA, a decentralized private and auditable payment system. Our accounts extend Quisquis accounts in order to record (in hidden form) the total influx and outflux. While we introduce two authorities in AQQUA, it remains decentralized since the RA and AA do not intervene in the normal flow of transactions. A major direction for possible future research involves strengthening the privacy provided by AQQUA even more. Firstly, the fact that the audit proofs leak account information between the audit states could be addressed. Secondly, another direction could be to convert audit proofs to be designated-verifier [12]. As a result, the AA will be able to simulate them, and thus it will be the only entity convinced about the audit results. This may increase the privacy of the participants, but it will interfere with the trust dynamics of the system. As a result, further research is needed for this integration.

References

[1] Stephanie Bayer and Jens Groth. "Efficient Zero-Knowledge Argument for Correctness of a Shuffle". In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 263–280. ISBN: 978-3-642-29011-4.

[2] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[3] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. "Mixcoin: Anonymity for Bitcoin with Accountable Mixes". In: *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Vol. 8437. Lecture Notes in Computer Science. Springer, 2014, pp. 486–504. DOI: 10.1007/978-3-662-45472-5_31. URL: https://doi.org/10.1007/978-3-662-45472-5_31.

[4] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. "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*. IEEE Computer Society, 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020. URL: https://doi.org/10.1109/SP.2018.00020.

[5] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. "SoK: Auditability and Accountability in Distributed Payment Systems". In: *Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part II*. Kamakura, Japan: Springer-Verlag, 2021, 311–337. ISBN: 978-3-030-78374-7. DOI: 10.1007/978-3-030-78375-4_13. URL: https://doi.org/10.1007/978-3-030-78375-4_13.

[6] David Chaum and Torben P. Pedersen. "Wallet Databases with Observers". In: *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*. Ed. by Ernest F. Brickell. Vol. 740. Lecture Notes in Computer Science. Springer, 1992, pp. 89–105. DOI: 10.1007/3-540-48071-4_7. URL: https://doi.org/10.1007/3-540-48071-4_7.

[7] Yu Chen, Xuecheng Ma, Cong Tang, and Man Ho Au. "PGC: Decentralized Confidential Payment System with Auditability". In: *Computer Security - ESORICS 2020 - 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part I*. Ed. by Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider. Vol. 12308. Lecture Notes in Computer Science. Springer, 2020, pp. 591–610. DOI: 10.1007/978-3-030-58951-6_29. URL: https://doi.org/10.1007/978-3-030-58951-6_29.

[8] Gaby G. Dagher, Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. "Provisions: Privacy-preserving Proofs of Solvency for Bitcoin Exchanges". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: Association for Computing Machinery, 2015, 720–731. ISBN: 9781450338325. DOI: 10.1145/2810103.2813674. URL: https://doi.org/10.1145/2810103.2813674.

[9] Maya Dotan, Ayelet Lotem, and Margarita Vald. "Haze: A Compliant Privacy Mixer". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1152. URL: https://eprint.iacr.org/2023/1152.

[10] Prastudy Fauzi, Sarah Meiklejohn, Rebekah Mercer, and Claudio Orlandi. "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*. Springer. 2019, pp. 649–678.

[11] Christina Garman, Matthew Green, and Ian Miers. "Accountable Privacy for Decentralized Anonymous Payments". In: *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*. Ed. by Jens Grossklags and Bart Preneel. Vol. 9603. Lecture Notes in Computer Science. Springer, 2016, pp. 81–98. DOI: 10.1007/978-3-662-54970-4_5. URL: https://doi.org/10.1007/978-3-662-54970-4_5.

[12] Markus Jakobsson, Kazue Sako, and Russell Impagliazzo. "Designated Verifier Proofs and Their Applications". In: *Advances in Cryptology — EUROCRYPT '96*. Ed. by Ueli Maurer. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 143–154. ISBN: 978-3-540-68339-1.

[13] Aggelos Kiayias, Markulf Kohlweiss, and Amirreza Sarencheh. "PEReDi: Privacy-Enhanced, Regulated and Distributed Central Bank Digital Currencies". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*. Ed. by Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi. ACM, 2022, pp. 1739–1752. DOI: 10.1145/3548606.3560707. URL: https://doi.org/10.1145/3548606.3560707.

[14] Ya-Nan Li, Tian Qiu, and Qiang Tang. "Pisces: Private and Compliable Cryptocurrency Exchange". In: *IACR Cryptol. ePrint Arch.* (2023), p. 1317. URL: https://eprint.iacr.org/2023/1317.

[15] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. "A fistful of Bitcoins: characterizing payments among men with no names". In: *Commun. ACM* 59.4 (2016), pp. 86–93. DOI: 10.1145/2896384. URL: https://doi.org/10.1145/2896384.

[16] Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: (May 2009). URL: http://www.bitcoin.org/bitcoin.pdf.

[17] Neha Narula, Willy Vasquez, and Madars Virza. "zkLedger: Privacy-Preserving Auditing for Distributed Ledgers". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 65–80. ISBN: 978-1-939133-01-4. URL: https://www.usenix.org/conference/nsdi18/presentation/narula.

[18] Shen Noether. *Ring Signature Confidential Transactions for Monero*. Cryptology ePrint Archive, Paper 2015/1098. 2015. URL: https://eprint.iacr.org/2015/1098.

[19] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 239–252. DOI: 10.1007/0-387-34805-0_22. URL: https://doi.org/10.1007/0-387-34805-0_22.

[20] Alin Tomescu, Adithya Bhat, Benny Applebaum, Ittai Abraham, Guy Gueta, Benny Pinkas, and Avishay Yanai. "UTT: Decentralized Ecash with Accountable Privacy". In: *IACR Cryptol. ePrint Arch.* (2022), p. 452. URL: https://eprint.iacr.org/2022/452.