AQQUA: Augmenting Quisquis with Auditability

George Papadoulis, Danai Balla, Panagiotis Grontas, and Aris Pagourtzis

National Technical University of Athens and Archimedes, Athena Research Center, Greece
geopapadoulis@gmail.com, balla.danai@gmail.com, pgrontas@corelab.ntua.gr, pagour@cs.ntua.gr

Abstract. We present AQQUA, a permissionless, private, and auditable payment system built on top of Quisquis. Unlike other auditable payment systems, AQQUA supports auditing, while maintaining privacy. It allows users to hold multiple accounts, perform concurrent transactions, and features a non-increasing state. AQQUA achieves auditability by introducing two authorities: one for registration and one for auditing. These authorities cannot censor transactions, thus preserving the decentralized nature of the system. Users create an initial account with the registration authority and then privately transact by using provably unlinkable updates of it. Audits can be voluntarily initiated by the users or requested by the audit authority at any time. Compliance is proved in zero-knowledge against a set of policies which include a maximum limit in the amount sent/received during a time period or in a single transfer, non-participation in a specific transaction or selective disclosure of the value exchanged. To analyze the security of AQQUA we formally define a security model for private and auditable decentralized payment systems. Using this model, we prove that AQQUA satisfies anonymity towards both the public and the auditor, theft prevention, and audit soundness.

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

1 Introduction

Fifteen years after the introduction of Bitcoin [23], the integration of blockchain-based cryptocurrencies - or more formally Distributed Payment Systems (DPS) - with the traditional financial system is still underwhelming, despite their huge popularity. This state of affairs is attributed [29] to the lack of three factors: performance, privacy, and regulation. While the lack of performance is a consequence of the underlying consensus mechanism, privacy and regulation present a deeper issue due to the inherent conflict between these two desiderata.

Regarding privacy, the inadequacy of Bitcoin's renewable pseudonyms to protect user privacy was demonstrated early on [22]. To overcome this problem, privacy-enhanced cryptocurrencies (e.g. Zerocash [4], Monero [25], Zether [8], Quisquis [17], Nopenena [1]) arose. These systems hide transaction identities and/or amounts exchanged, thus providing privacy in a provable cryptographic manner. At the same time, however, they make it easier for malicious users to conduct illegal activities (e.g. money laundering, unauthorized funds transfer, tax evasion), thus diminishing any regulation prospects and making mainstream adoption even more unlikely.

Consequently, auditable privacy solutions [18, 11, 20, 21, 15, 24, 29, 13, 2, 28, 7] arose to make private cryptocurrencies regulation-friendly without (entirely) compromising user privacy. 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 to the total value that can be exchanged in a single transaction.

In this work, we address privacy and auditability through AQQUA.

Overview AQQUA equips Quisquis [17] with auditability without changing its decentralized, permissionless, and trustless nature, while maintaining transaction anonymity and confidentiality. To this end, we introduce two new entities: A Registration Authority (RA) to enroll users in the system, and an Audit Authority (AA) to perform audits. In order to transact in AQQUA, users must first register with the RA and provide their real-world credentials, thus fulfilling KYC. Then they acquire a cryptographic pseudonym in the form of an initial updatable public key as in [17], which is used to create new accounts in the system.

AQQUA accounts consist of a public key, and hiding commitments for the balance, the total amount of coins spent, and the total amount of coins received. New accounts can be created by any registered user by updating the public key in a provably unlinkable manner as in [17], without requiring interaction with the RA. As a result, users can own as many accounts as they wish, contrary to other private and auditable DPS. The number of accounts per user is recorded in the state of the system, which is split between two sets: the UTXOSet contains user accounts, and the UserSet maintains a mapping between a registered public key and the number of accounts (in committed form) that have been created by updating this public key. For each new account this commitment must be updated. The addition of a new public key to the UserSet can happen only after approval by the RA. Nevertheless, the RA cannot censor or identify user transactions after enrollment, since the process of updating an initial public key to create new accounts operates independently of it.

In AQQUA, transactions can be thought of as 'wealth redistribution' between inputs and outputs, an idea from Quisquis [17]. Input accounts include the set of senders, the set of recipients as well as an anonymity set. Each account participating in these sets is an update of an account from the UserSet. Output accounts are new, updated but unlikable accounts for the senders, recipients, and decoys. To enforce 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.

A user can produce a proof of compliance with a policy for a specific transaction or all their exchanges for a particular time period of interest. To do so their initial public key is required, which may be disclosed to the AA voluntarily by the user themselves, or the AA might acquire it in cooperation with the RA. Then, the user proves in zero-knowledge that they are compliant with the audited policy, using data that are only stored on-chain.

AQQUA enables voluntarily auditing, i.e. honest users may initiate the audit, in order to enjoy some advantage [7]. For example, consider the case where the AA is a cryptocurrency exchange. A user can prove to the AA that their AQQUA accounts are compliant with a policy, before being allowed to use the exchange for AQQUA coins. Another example is when the user is among the set of suspects for some illicit activity. Compliant users can prove their 'innocence' without giving up their privacy. On the other hand, audits may be initiated by the collaboration of the AA and RA. In such cases, auditing can be effectively made mandatory if combined with off-chain penalties to non-compliant or irresponsive users.

2 Related Work

AQQUA takes the auditability route to regulation; there is an external auditor (the AA) who can request an 'explanation' of the data stored on the blockchain at will. The other option is accountability [9], where policies are evaluated at the system level when certain predicates are met, and non-conforming transactions never make it to the blockchain. Auditability is better suited to the Quisquis setting, since it imposes no extra burden to its consensus layer.

Most works that combine privacy and regulation, apply to the permissioned setting [20, 27, 11, 13, 24, 28], which is considered appropriate for Central Bank Digital Currencies (CBDC). These approaches use a distributed ledger to record transactions between banks or large financial organizations, which might also play the role of validators. In contrast, AQQUA makes no assumptions about its users. In order to limit the power of the validators while enforcing compliance, [20, 27] use secure multi-party computation techniques to distribute the application of regulation policies between different parties. PGC [11] provides a generalized design and an implementation that combines confidentiality with auditability, altogether skipping anonymity. Their proposal supports a rich set of regulation policies to limit money laundering and enable taxation. AQQUA tries to apply the expressiveness of PGC's policies to the permissionless setting, while also being anonymous to outsiders and to auditors in-between audits. UTT [27] has the unique approach of allowing users a privacy budget to spend in order to satisfy KYC policies. In AQQUA, there are no limits either in spending or in the number of accounts that can be created.

In the permissionless setting, there are some works that combine auditability and privacy utilizing points of concentration like privacy mixers [5, 14, 7] or exchanges [21]. In Haze [14], compliance amounts to approving only the transactions which do not originate from a black list of banned addresses, even if the address is banned after funds were deposited on the mixer. Pisces [21] achieves anonymity for all asset types that might be traded in an exchange, while supporting aggregation of statistics that allow tax calculation. Privacy pools [7] allows users to prove that their withdrawals (do not) originate from (black-) white-listed deposits. Our approach in AQQUA has two advantages over [5, 14, 21, 7]: Firstly, it is censorship-resistant since neither the RA nor the AA can censor transactions of registered users. Secondly, auditing in AQQUA has a complete view of the blockchain, which means that a richer set of data can be given to as input to compliance policies.

To the best of our knowledge, works with a philosophy close to AQQUA are [18, 15]. The former modifies the Zerocash [4] coin format by adding counters that allow the data aggregation that may be used for auditing. However, [18] is plagued with the same problems as Zerocash; a monotonically increasing UTXO set that affects performance which is exacerbated by the addition of auxiliary information. AQQUA inherits from Quisquis [17] where transactions don't increase the size of the UTXO set. Furthermore, in contrast to [18], AQQUA does not have an option for transaction tracing, and while account information is revealed during an audit, user privacy is immediately restored afterwards. [15] can be used to create a private cryptocurrency that allows for the creation of multiple accounts per user and may be extended to support auditing. It is built on type of accountable ring signature, which can reveal the spender identity *for a single transaction* through an extractable commitment. It is not specified how this could be extended to provide compliance proofs for policies that span multiple transactions. Furthermore, [15] does not provide a complete system architecture nor a security model and analysis. Its difficult thus to evaluate its security as a complete DPS (e.g against collusion attacks).

In conclusion, a distinctive aspect of AQQUA lies in its ability to allow each user to maintain multiple accounts while simultaneously supporting compliance proofs for policies such as sending limits. This introduces a significant technical challenge: ensuring that no account involved in sending or receiving is maliciously omitted from the compliance proof. While prior work has examined the intersection of payment privacy and auditing, there has been inadequate exploration of this specific challenge. AQQUA addresses this gap by proposing a complete solution that ensures both strong privacy guarantees and robust compliance.

3 Preliminaries

Notation. We denote by λ the security parameter. $\mathcal{M}$ is the message space of our cryptographic schemes and $\mathcal{V} = \{0, \ldots, V\}$ defines the range of valid currency values, where V is an upper bound on the maximum possible number of coins ($|\mathcal{V}| \ll |\mathcal{M}|$). 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 employ the dot notation, i.e. $t.a$ or $t.b$ and denote $t^x = (a, b)^x = (a^x, b^x)$ and $t_1 \cdot t_2 = (t_1.a \cdot t_2.a, t_1.b \cdot t_2.b)$. Our cryptographic primitives operate in a group $\mathbb{G}$ of prime order p generated by $g \in \mathbb{G}$ $(\mathbb{G}, p, g)$ where the DDH is hard, with corresponding field $\mathbb{F}_p$.

Updatable Public Keys. An Updatable Public Key [17] (UPK) can be updated while remaining indistinguishable from freshly generated keys. We utilize the construction of [17] where a public key pk_i is a tuple (g_i, g_i^{sk}) where $sk \leftarrow\!\!\$\ \mathbb{F}_p$ is the secret key and $g_i \in \mathbb{G}$. UPKs can be updated through $\mathsf{Update}(\{\mathsf{pk}_i\}; r)$ which computes $\left\{\mathsf{pk}'_i = \mathsf{pk}_i^r\right\}, r \leftarrow\!\!\$\ \mathbb{F}_p^*$ for all i. Using the secret key one can verify if a keypair $\mathsf{pk} = (g', h')$ is an UPK by calling $\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{pk})$ which checks if $(g')^{\mathsf{sk}} = h'$. Finally, one can use $\mathsf{VerifyUpdate}(\mathsf{pk}', \mathsf{pk}, r)$ to verify if pk' is a valid update of pk using r, by checking if $\mathsf{Update}(\{\mathsf{pk}\}; r) = \mathsf{pk}'$. The security properties of UPKs prevent an adversary from distinguish between a new public key and an updated one (indistinguishability), and from explaining a public key update without the secret key and the randomness. We provide a formal description in Appendix A.1.

Commitments. We use a *computationally hiding*, *unconditionally binding*, *additively homomorphic* (thus *re-randomizable*), and *key anonymous* commitment scheme over $(\mathbb{G}, p, g)$ with $\boxed{m} \overset{\text{def}}{=} \mathsf{Commit}(\mathsf{pk}, m; r) = (c, d) = (g_i^r, g^m h_i^r)$.

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

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

***Σ*-protocols.** AQQUA utilizes well known Σ-protocols for proving discrete logarithm knowledge [26], and DDH tuples [10]. We also use the variation of the Bayer-Groth shuffle proof [3] from [17], and Bulletproofs [6] for range proofs. Furthermore, we utilize two specific Σ-protocols from [17] (cf. Appendix A.2):

- Σ_{vu} proves the validity of a UPK update, i.e. knowledge of w: $\mathsf{pk}' = \mathsf{pk}^w$.
- Σ_{com} for proving that two commitments hide the same value, i.e. knowledge of $w = (v, r_1, r_2)$ such that $\mathtt{com}_1 = \mathsf{Commit}(\mathsf{pk}_1, v; r_1), \mathtt{com}_2 = \mathsf{Commit}(\mathsf{pk}_2, v; r_2)$

4 AQQUA Architecture

Entities. In AQQUA there are the following types of entities:

- *Registration Authority* (RA): Enrolls users by linking their real-world identity to an initial public key ($\mathtt{pk}_0$). All users' accounts within the system will originate from $\mathtt{pk}_0$, through updates. RA stores identity data off-chain for enforcement penalties on non-compliant users.
- *Audit Authority* (AA): Initiates the audits to verify user compliance with system's policies. The AA cooperates with the RA to penalize non-compliant users.
- *Users* (U): Participants that own and transact through multiple accounts.

State. In AQQUA the state (denoted state) is composed of two sets:

- $\mathsf{UTXOSet}$: Contains the ’unspent’ user accounts that are outputs of valid transactions but have not yet been used as inputs.
- $\mathsf{UserSet}$: Stores a mapping between the user’s initial public key and a commitment to the number of accounts they own. This ensures that the user cannot withhold information from the $\mathtt{AA}$ during audits. While all users can update existing information in the $\mathsf{UserSet}$, only the $\mathtt{RA}$ may add new entries.

Accounts. Users may own multiple accounts of the form $\mathsf{acct} = (\mathsf{pk}, \boxed{\mathtt{bl}}, \boxed{\mathtt{out}}, \boxed{\mathtt{in}})$, where $\mathtt{bl}$ is the account balance and $\mathtt{out}, \mathtt{in}$ are the total amounts the account has sent and received, respectively. Accounts functionalities include:

- $\mathsf{acct} \leftarrow \mathsf{NewAcct}(\mathtt{pk}_0; \vec{r} = (r_1, r_2, r_3, r_4))$: Outputs a new account with an updated public key $\mathsf{pk} = \mathsf{Update}(\mathtt{pk}_0; r_1)$ and zero-value commitments for balance, sent, and received amounts, using randomness r_2, r_3, r_4 respectively. Namely, $\mathsf{acct} = (\mathsf{pk}, \boxed{\mathtt{bl}}, \boxed{\mathtt{out}}, \boxed{\mathtt{in}})$, where $\mathsf{pk} = \mathsf{Update}(\mathtt{pk}_0; r_1)$, $\boxed{\mathtt{bl}} = \mathsf{Commit}(\mathsf{pk}, 0; r_2)$, $\boxed{\mathtt{out}} = \mathsf{Commit}(\mathsf{pk}, 0; r_3)$ and $\boxed{\mathtt{in}} = \mathsf{Commit}(\mathsf{pk}, 0; r_4)$.
- $0/1 \leftarrow \mathsf{VerifyAcct}(\mathsf{acct}, \mathsf{sk}, \mathtt{bl}, \mathtt{out}, \mathtt{in})$: Verifies that the commitments in the account correspond to the provided values. It parses $\mathsf{acct} = (\mathsf{pk}, \mathtt{com}_1, \mathtt{com}_2, \mathtt{com}_3)$ and outputs 1 if

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

- $\{\mathsf{acct}'_i\}_{i=1}^n \leftarrow \mathsf{UpdateAcct}(\{\mathsf{acct}_i, \mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{in}i}, \mathtt{v}_{\mathtt{out}i}\}_{i=1}^n; \vec{r})$: Outputs a new set of unlinkable accounts by updating their public key and commitments. It parses $\mathsf{acct}_i = (\mathsf{pk}_i, \mathtt{com}_{\mathtt{bl}i}, \mathtt{com}_{\mathtt{out}i}, \mathtt{com}_{\mathtt{in}i})$ and $\vec{r} = (r_1, r_2, r_3, r_4)$ and outputs $\{\mathsf{acct}'_i\}_{i=1}^n$ where

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

- $0/1 \leftarrow \mathsf{VerifyUpdateAcct}(\{\mathsf{acct}'_i, \mathsf{acct}_i, \mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{out}i}, \mathtt{v}_{\mathtt{in}i}\}_{i=1}^n; \vec{r})$: Verifies that an account update was performed correctly by comparing the original and updated accounts, balances, and randomness. Outputs 1 if

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

User information. Each user is associated with a tuple $\mathsf{userInfo} = (\mathtt{pk}_0, \boxed{\mathtt{\#accs}})$, which is stored in the $\mathsf{UserSet}$, where $\mathtt{pk}_0$ is the initial public key assigned at registration, and $\boxed{\mathtt{\#accs}}$ is a commitment to the number of accounts the user owns in the $\mathsf{UTXOSet}$. This commitment ensures that users cannot conceal any accounts during audits, and its opening is disclosed only to the $\mathtt{AA}$. The functionalities for user information are the following.

- $(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}) \leftarrow \mathsf{GenUser}()$: Creates a new user by producing a new keypair $(\mathsf{sk}, \mathtt{pk}_0)$, a new account acct for $\mathtt{pk}_0$ and a $\mathsf{userInfo}$ tuple. The algorithm picks $r_1, r_2, r_3, r_4, r_5 \twoheadleftarrow\$ \ \mathbb{F}_p^*$ and let $\vec{r} = (r_1, r_2, r_3, r_4)$. Then runs $(\mathsf{sk}, \mathtt{pk}_0) \leftarrow \mathsf{KGen}()$, $\mathsf{acct} \leftarrow \mathsf{NewAcct}(\mathtt{pk}_0; \vec{r})$ and calculates $\mathsf{userInfo} = (\mathtt{pk}_0, \mathsf{Commit}(\mathtt{pk}_0, 1; r_5))$.
- $0/1 \leftarrow \mathsf{VerifyUser}(\mathsf{userInfo}, (\mathsf{sk}, \mathtt{\#accs}))$: verifies the correctness of $\mathsf{userInfo} = (\mathtt{pk}_0, \mathtt{com})$ by checking if

$$\mathsf{VerifyUpdate}(\mathsf{sk}, \mathtt{pk}_0) \wedge \mathsf{VerifyCom}(\mathsf{sk}, \mathtt{pk}_0, \mathtt{com}, \mathtt{\#accs}) \wedge (\mathtt{\#accs} \in \mathcal{V}).$$

- $\{\mathsf{userInfo}'_i\}_{i=1}^n \leftarrow \mathsf{UpdateUser}(\{\mathsf{userInfo}_i, \mathtt{v}_{\#accs_i}\}_{i=1}^n; r)$: Updates a set of user information tuples by modifying the commitment to reflect the updated number of accounts. It takes as input n tuples $\mathsf{userInfo}_i = (\mathtt{pk}_{0i}, \mathtt{com}_{\mathtt{\#accs}i})$ and $\mathtt{v}_{\#accs_i} \in \mathcal{V}$ and outputs the updated set $\{\mathsf{userInfo}'_i\}_{i=1}^n = \{(\mathtt{pk}_{0i}, \mathtt{com}_{\mathtt{\#accs}}{}'_i)\}_{i=1}^n$, where

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

- $0/1 \leftarrow \mathsf{VerifyUpdateUser}(\{\mathsf{userInfo}'_i, \mathsf{user}_i, \mathtt{v}_{\#accs_i}\}_{i=1}^n; r)$: Verifies the correctness of the updated user information by comparing the original and updated values and commitments. Outputs 1 if

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

Policies. AQQUA supports policies that address Anti-Money Laundering (AML) requirements and selective disclosure, allowing for compliance with regulations, while preserving user privacy. We express policies 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 $\mathtt{aux}$ specific to a compliance objective.

We introduce our supported policy predicates, where A_1, A_2 denote the sets of accounts owned by the user with $\mathtt{pk}_0$ in $\mathsf{state}_1.\mathsf{UTXOSet}, \mathsf{state}_2.\mathsf{UTXOSet}$, respectively, and $\mathtt{bl}, \mathtt{out}, \mathtt{in}$ represent the variables $\mathsf{acct}.\mathtt{bl}, \mathsf{acct}.\mathtt{out}, \mathsf{acct}.\mathtt{in}$ for a specific account acct, accordingly.

- Predicate $f_{\{\mathsf{slimit},\mathsf{rlimit}\}}(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), \mathsf{a}_{max})$ for *sending/receiving limit*: Restricts the amount that a user can send/receive within a given time period, thus helping the enforcement of AML regulations. The sent and received amounts should not exceed a predefined threshold a_{max}, i.e.: $\sum_{\mathsf{acct} \in A_2} \mathtt{out} - \sum_{\mathsf{acct} \in A_1} \mathtt{out} \leq \mathsf{a}_{max}$. For f_{rlimit} we use $\mathtt{in}$ instead of $\mathtt{out}$ respectively.
- Predicate $f_{\mathsf{txlimit}}(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), v_{\max})$ for *transaction value limit*: Set an upper bound $v_{\max}$ to the value transferred in a single transaction, i.e.: $\sum_{\mathsf{acct} \in A_1} \mathtt{bl} - \sum_{\mathsf{acct} \in A_2} \mathtt{bl} \leq v_{\max}$.
- Predicate $f_{\mathsf{np}}(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2))$ for *non-participation*: Verifies that a user has not participated in any transaction during a given time period, i.e.: $(\sum_{\mathsf{acct} \in A_1} \mathtt{out} - \sum_{\mathsf{acct} \in A_2} \mathtt{out} = 0) \wedge (\sum_{\mathsf{acct} \in A_1} \mathtt{in} - \sum_{\mathsf{acct} \in A_2} \mathtt{in} = 0)$.
- Predicate $f_{\mathsf{open}}(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), v_{\mathsf{open}})$ for *open transaction*: Selective disclosure of value sent or received (v_{open}) in a specific transaction, i.e.: $(v = (\sum_{\mathsf{acct} \in A_2} \mathtt{bl} - \sum_{\mathsf{acct} \in A_1} \mathtt{bl}) \in \mathcal{V}) \wedge (v = v_{\mathsf{open}} \vee v = -v_{\mathsf{open}})$.

Tracking the number of accounts a user owns is essential for enforcing value-limit policies. Without this, users could bypass these policies by creating multiple sybil identities [9]. For this reason AQQUA includes an `RA`.

5 AQQUA Functionalities

AQQUA consists of functionalities to set up the system, register users, issue transactions and undergo audits. Registration functionalities are used to create, verify, and add registration data. Transaction functionalities are used to send funds, create and delete accounts, verify transactions and apply transactions to the state. Audit functionalities create and verify audit proofs. An overview of the system is shown in Figure 1.

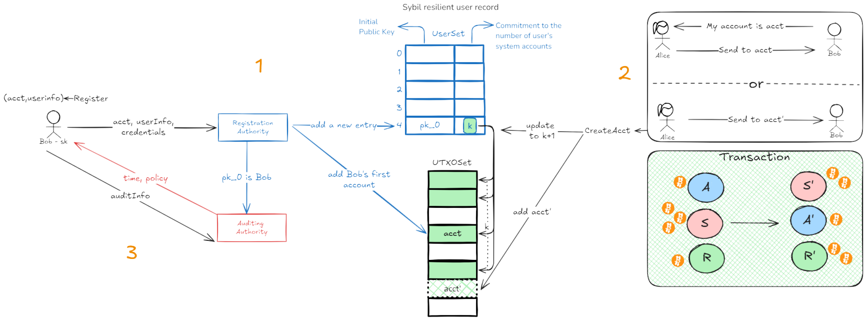

Fig. 1. Overview 1.*Registration.* Bob first registers with the `RA`, which verifies the registration information and enrolls him to the system. $\mathsf{UserSet}$ is a sybil-resilient record of the registered users information. 2.*Transaction.* Alice wants to send coins to Bob. She can either contact Bob to get one of his accounts or she can create a new account for Bob if she knows his initial public key (the `RA` is not involved in this procedure). Transactions can be thought of as 'wealth redistribution' [17]. Input accounts include the set of senders, the set of recipients as well as an anonymity set. 3.*Audit.* The `AA` can cooperate with the `RA` to find which initial public key corresponds to Bob and then initiates the audit procedure with Bob.

5.1 Setup

The $(\mathsf{state}_0, \mathsf{pp}) \leftarrow \mathsf{Setup}(\lambda)$ algorithm takes as input the security parameter λ and returns the public parameters pp and the initial state state_0 which contains an empty $\mathsf{UserSet}$ and $\mathsf{UTXOSet}$.

5.2 Registration

In order for users to register in AQQUA, they use the $(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}, \pi) \leftarrow \mathsf{Register}()$ algorithm to create an initial updatable public key pk_0, their first account acct and the corresponding secret key sk.

The intial public key $\mathtt{pk}_0$ is generated by the user and will be sent to $\mathtt{RA}$, while sk is kept secret. All the valid initial public keys should be constructed with randomness 1, meaning $\mathtt{pk}_0 = (g, g^{\mathsf{sk}})$. This is to prevent a collision attack, where two different users may create an initial public key that derives from the same secret key.

Using $\mathtt{pk}_0$, the algorithm creates the user's first account $\mathsf{acct} = (\mathsf{pk}, \boxed{0}, \boxed{0}, \boxed{0})$, where pk is an update of $\mathtt{pk}_0$, and the corresponding $\mathsf{userInfo} = (\mathtt{pk}_0, \boxed{1})$ entry. It also creates a NIZK argument π of correct construction, starting that both $\mathtt{pk}_0, \mathsf{pk}$ correspond to sk, that $\mathsf{userInfo}$ contains a commitment to 1 as the initial number of accounts, and that the acct commitments contain 0 as the initial balance, total amount sent/received.

Then, the user sends their real-world identification information along with $\mathsf{userInfo}, \mathsf{acct}$ and π to the $\mathtt{RA}$, which verifies them.

The registration algorithm is depicted in Figure 2.

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

$$\mathsf{VerifyCom}(\mathsf{userInfo}.\mathtt{pk}_0, \mathsf{userInfo}.\mathsf{com}_{\mathtt{\#accs}}, (\mathsf{sk}, 1)) = 1 \wedge$$
$$\mathsf{VerifyKP}(\mathsf{userInfo}.\mathtt{pk}_0, \mathsf{sk}) = 1 \wedge \mathsf{VerifyAcct}(\mathsf{acct}, \mathsf{sk}, 0, 0, 0) = 1$$

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

Fig. 2. The Register algorithm.

To verify, the $\mathtt{RA}$ invokes $0/1 \leftarrow \mathsf{VerifyRegister}(\mathsf{userInfo}, \mathsf{acct}, \pi, \mathsf{state})$, which first checks that $\mathsf{userInfo}.\mathtt{pk}_0 = (g, \cdot)$ and that it does not already exist in a $\mathsf{userInfo}$ entry of $\mathsf{UserSet}$. It then executes the verification algorithm for the NIZK argument π and returns its result. If the arguments verify, the $\mathtt{RA}$ notifies validators for $(\mathsf{userInfo}, \mathsf{acct}, \pi)$ through an authenticated channel, so they can update the state using the $\mathsf{state}' \leftarrow \mathsf{ApplyRegister}(\mathsf{userInfo}, \mathsf{acct}, \mathsf{state})$ algorithm. The algorithm adds $\mathsf{userInfo}$ to $\mathsf{state}.\mathsf{UserSet}$ and acct to $\mathsf{state}.\mathsf{UTXOSet}$, and returns the resulting new state state'.

It is important to note that the $\mathtt{RA}$'s sole purpose is to maintain a sybil-resilient record of users. The only additional information the $\mathtt{RA}$ holds, compared to other entities, is the real-world credentials of users and their correlation with the initial public keys. This information, however, has no bearing on the accounts created and used within the system, which remain entirely anonymous and unlinkable post-registration.

5.3 Transactions

Transactions are used to exchange money and create or remove accounts.AQQUA's transaction algorithm extends the respective one in Quisquis [17] by introducing additional fields to update the total amount sent from and received by each participating account.

Transaction Algorithm. When a user wants to send coins to one or more recipients they invoke $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}, \mathtt{A})$. The accounts in the set $\mathtt{S}$ are owned by sk, while $\mathtt{R}$ contains the receivers. Since accounts are anonymous, defining $\mathtt{R}$ requires the sender to either contact the recipients off-chain to request their accounts or create a new account on their behalf using the $\mathsf{CreateAcct}$ algorithm. The vectors $\overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}$ contain the amounts to be subtracted from and added to the balance of each account in $\mathtt{S}$, $\mathtt{R}$, respectively. Finally, the anonymity set $\mathtt{A}$ is used to hide the identity of the sender and receiver accounts. The Trans algorithm which is specified in Appendix B.1 works as follows:

- Ensures that each account in $\mathtt{S}$ is owned by sk by using $\mathsf{VerifyKP}$, and that $|\mathtt{S}| = |\overrightarrow{\mathtt{v_S}}|, |\mathtt{R}| = |\overrightarrow{\mathtt{v_R}}|$.
- Ensures that the sum of entries of $\overrightarrow{\mathtt{v_S}}, \overrightarrow{\mathtt{v_R}}$ is zero, values in $\overrightarrow{\mathtt{v_R}}$ are positive, values in $\overrightarrow{\mathtt{v_S}}$ are negative, and after adding the value of $\overrightarrow{\mathtt{v_S}}$ to the balance of the corresponding account, the result stays non-negative (i.e. each account has enough funds to send).
- Sorts $\mathtt{S} \cup \mathtt{R} \cup \mathtt{A}$ in some canonical order and stores the result in `inputs`.
- Using $\mathsf{UpdateAcct}$, re-randomizes the public keys of accounts of `inputs` and re-randomizes and updates their balances, total amount sent and total amount received. The balances of accounts in $\mathtt{S}$ are reduced by the value in the matching entry in $\overrightarrow{\mathtt{v_S}}$ and the ones in $\mathtt{R}$ are increased according to $\overrightarrow{\mathtt{v_R}}$. Balances of accounts in $\mathtt{A}$ remain unaltered, only re-randomized. The total amount received and sent are updated appropriately, i.e. if an account is a sender/receiver account, the total amount sent/received is increased by the amount sent from/received by the account. Finally it sorts the updated accounts in some canonical order. The results are assigned to `outputs`.
- Forms a NIZK argument π which proves that the accounts in `outputs` are created with the above procedure.
- Returns $\mathsf{tx} = (\mathtt{inputs}, \mathtt{outputs}, \pi)$.

Every account can appear in at most two transactions; once when it is created as an output and once when included in the inputs of another transaction, regardless of whether it is the actual sender or is included for anonymity only.

Create Account. Within the system, every user can create a new account for any other registered user, which improves efficiency [17]. 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 a result, owning multiple accounts enables a user to send or receive funds by transactions created in parallel. Furthermore, increasing the number of accounts decreases the probability of including the same acount in the anonymity sets of two concurrent transactions.

The $\mathsf{tx}_{\mathsf{CA}} = (\mathsf{acct}, \mathtt{inputs}, \mathtt{outputs}, \pi) \leftarrow \mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$ algorithm can be used by any user to construct a transaction that creates a new account for the owner of $\mathsf{userInfo}$. It takes as input an anonymity set containing entries of $\mathsf{UserSet}$, used to hide $\mathsf{userInfo}$. The algorithm creates the new account $\mathsf{acct} = (\mathsf{pk}, \boxed{0}, \boxed{0}, \boxed{0})$, where pk is an update of $\mathsf{userInfo.pk}_0$. Using $\mathsf{UpdateUser}$, it updates $\mathsf{userInfo}$ by increasing by one and re-randomizing the committed value for the number of accounts the user owns, and re-randomizes all commitments of entries of $\mathtt{A}$. Thus, $\mathtt{inputs} = \{\mathsf{userInfo}\} \cup \mathtt{A}$ in some canonical order and $\mathtt{outputs}$ consists of the re-randomized and updated $\mathtt{inputs}$. Finally, π is a NIZK argument that $\mathsf{tx}_{\mathsf{CA}}$ has been constructed according to the above procedure. The detailed description of the $\mathsf{CreateAcct}$ algorithm is depicted in Appendix B.2.

Delete Account. Allowing users to delete zero-balance accounts reduces the storage overhead of AQQUA, since accounts that have no balance left might be abandoned and thus not needed to be stored in the $\mathsf{UTXOSet}$. Furthermore, due to the fact that senders usually create new accounts for their intended recipients, the number of accounts in the $\mathsf{UTXOSet}$ increases if the option to remove zero-balance accounts is not given. Users should be incentivized to delete the zero-balance accounts they own and don't need to keep.

The $\mathsf{tx}_{\mathsf{DA}} = (\mathtt{inputs}, \mathtt{outputs}, \pi) \leftarrow \mathsf{DelAcct}(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}_{\mathsf{D}}, \mathsf{acct}_{\mathsf{C}}, \mathtt{A}_1, \mathtt{A}_2)$ takes as input the user's secret key sk, user information $\mathsf{userInfo}$, the zero-balance account to be deleted $\mathsf{acct}_{\mathsf{D}}$, an account $\mathsf{acct}_{\mathsf{C}}$ to transfer the information containing the total amount sent and received of $\mathsf{acct}_{\mathsf{D}}$, and anonymity sets $\mathtt{A}_1, \mathtt{A}_2$ to hide $\mathsf{acct}_{\mathsf{D}}, \mathsf{acct}_{\mathsf{C}}$ and $\mathsf{userInfo}$, respectively. In $\mathsf{tx}_{\mathsf{DA}}$, the set $\mathtt{inputs}$ consists of $\mathtt{A}_1 \cup \{\mathsf{acct}_{\mathsf{D}}, \mathsf{acct}_{\mathsf{C}}\}$, $\mathtt{A}_2 \cup \{\mathsf{userInfo}\}$ in some canonical order. The algorithm re-randomizes and decreases by one the commitment to the number of accounts the user owns in $\mathsf{userInfo}$, adds to the corresponding fields of $\mathsf{acct}_{\mathsf{C}}$ the total amount sent and received of $\mathsf{acct}_{\mathsf{D}}$ and re-randomizes $\mathsf{acct}_{\mathsf{C}}$, removes $\mathsf{acct}_{\mathsf{D}}$, and re-randomizes all other accounts and user information. The set $\mathtt{outputs}$ consists of the resulting accounts and user information in some canonical order, and π consists of a NIZK argument of correct construction. The detailed description of the $\mathsf{DelAcct}$ algorithm can be found in Appendix B.3.

Transaction Verification. The $0/1 \leftarrow \mathsf{VerifyTrans}(\mathsf{tx}, \mathsf{state})$ algorithm guarantees the validity of transaction tx, which can be either transfer, create or delete account transaction. The algorithm checks that all accounts in $\mathsf{tx}.\mathtt{inputs}$ are present in state and executes the verification algorithm for π.

Apply Transaction. The $\mathsf{state}' \leftarrow \mathsf{ApplyTrans}(\mathsf{tx}, \mathsf{state})$ algorithm is executed after the verification of the tx to update the state by adding $\mathsf{tx}.\mathtt{outputs}$ and removing $\mathsf{tx}.\mathtt{inputs}$. It returns the new state state'.

Similarly to [17], upon receiving a new state, users whose accounts are included in a transaction's $\mathtt{inputs}$ should identify their updated accounts in $\mathtt{outputs}$. This can be accomplished by iterating through every $\mathsf{acct} \in \mathtt{outputs}$ 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 $\mathtt{bl}$. 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 requires reasonable time.

5.4 Audit

During auditing, the $\mathtt{AA}$ selects a user by their initial public key $\mathtt{pk}_0$, and two state snapshots $\mathsf{state}_1, \mathsf{state}_2$. For the policies applied to transactions (namely $f_{\mathsf{txlimit}}, f_{\mathsf{open}}$), state_2 should be the state that results from applying them to state_1. For the policies applied to a time period (namely $f_{\mathsf{slimit}}, f_{\mathsf{rlimit}}, f_{\mathsf{np}}$), the snapshots $\mathsf{state}_1, \mathsf{state}_2$ represent the beginning and end of the time period the auditor is interested in.

The user first opens for each of the two snapshots the committed value of the number of accounts they own ($\boxed{\mathtt{\#accs}}$ field of $\mathsf{userInfo}$). To protect their privacy, the user picks two anonymity sets, one for each snapshot, and re-randomizes and shuffles the union of the set of their accounts in each snapshot and the corresponding anonymity set. Then, they reveal their re-randomized accounts in each of the resulting set. For each set, they should reveal a number of accounts equal to the corresponding commitment opening. Finally, they create a NIZK argument that proves the correct re-randomization of accounts, the ownership of the accounts, and that the sets of re-randomized accounts satisfy the required policy predicate (cf. section 4).

As a first example, consider a user that is audited for f_{slimit} with $\mathtt{aux} = a_{\max}$, and let $\{\mathsf{acct}_{1i}\}_{i=1}^{\mathtt{\#accs}_1}, \{\mathsf{acct}_{2i}\}_{i=1}^{\mathtt{\#accs}_2}$ be the sets of their accounts in each re-randomized snapshot. The user calculates $\boxed{\mathtt{out}_j^*} = \prod_{i=1}^{\mathtt{\#accs}_j} \mathsf{acct}_{ji}.\boxed{\mathtt{out}}$ for $j = 1, 2$, and $\boxed{\mathtt{out}^*} = \boxed{\mathtt{out}_2^*}\left(\boxed{\mathtt{out}_1^*}\right)^{-1}$. Notice that the value $\mathtt{out}^*$ corresponds to the total amount sent in transactions of the user in the time period defined by state_1 and state_2. Finally, they prove in zero-knowledge that $\mathtt{out}^* \leq a_{\max}$ [6]. An example of an audit for the slimit policy is depicted in Figure 3.

As a second example, consider a user that is audited for f_{open} for a transaction. Using the same notation as in the previous example, the user calculates $\boxed{\mathtt{bl}_j^*} = \prod_{i=1}^{\mathtt{\#accs}_j} \mathsf{acct}_{ji}.\boxed{\mathtt{bl}}$ for $j = 1, 2$, then $\boxed{\mathtt{bl}^*} = \boxed{\mathtt{bl}_2^*}\left(\boxed{\mathtt{bl}_1^*}\right)^{-1}$. Then, calculates $\mathtt{bl}^* = \mathsf{OpenCom}(\mathsf{sk}, \boxed{\mathtt{bl}^*})$, which, since state_2 is the state resulting from applying the transaction to state_1, it is the value sent or received in the transaction. Then, they prove in zero-knowledge that $\mathtt{bl}^*$ is the opening of $\boxed{\mathtt{bl}^*}$.

All these actions are performed in the $\mathsf{auditInfo} \leftarrow \mathsf{PrepareAudit}(\mathsf{sk}, \mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}), \mathtt{A}_1, \mathtt{A}_2)$ algorithm which is invoked by the user. The algorithm returns $\mathsf{auditInfo} = (\mathtt{outputs}_1, \mathtt{outputs}_2, \mathtt{\#accs}_1, \{\mathsf{acct}_{1i}\}_{i=1}^{\mathtt{\#accs}_1}, \mathtt{\#accs}_2, \{\mathsf{acct}_{2i}\}_{i=1}^{\mathtt{\#accs}_2}, \pi)$, where $\mathtt{outputs}_j$ is the re-randomized $\mathsf{state}_j.\mathsf{UTXOSet}$, for $j = 1, 2$. We detail its operations in Appendix B.4.

We note that since the user re-randomizes their accounts together with an anonymity set and then reveals the re-randomized accounts, the $\mathtt{AA}$ is unable to link the revealed accounts with the user's original accounts in the initial snapshots. This guarantees that the user's privacy is preserved during individual audits. The size of the anonymity sets is picked by the user, with respect to the level of privacy they wish to achieve.

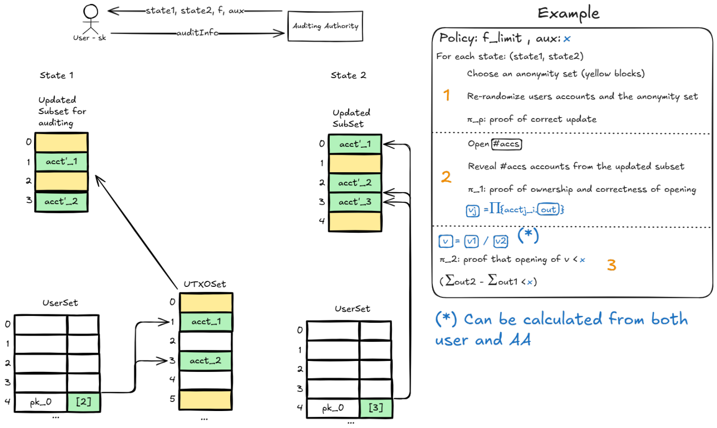
state1, state2, f, aux
auditInfo
User - sk
Auditing Authority
State 1
State 2
Updated Subset for auditing
Updated SubSet
acct'_1
acct'_2
acct'_3
UTXOSet
acct_1
acct_2
UserSet
pk_0
[2]
[3]
Example
Policy: f_limit , aux: x
For each state: (state1, state2)
Choose an anonymity set (yellow blocks)
1
Re-randomize users accounts and the anonymity set
π_p: proof of correct update
Open #accs
2
Reveal #accs accounts from the updated subset
π_1: proof of ownership and correctness of opening
vj =∏{acctj_i.out}
v = v1 / v2 (*)
π_2: proof that opening of v < x
3
(∑out2 - ∑out1 < x)
(*) Can be calculated from both user and AA

Fig. 3. Audit Example: Limit policy. The user is audited for the slimit policy with $\mathtt{aux} = x$. She opens to $\mathtt{AA}$ the commitment to the number of her accounts and then she re-randomizes a part of the $\mathsf{UTXOSet}$ in the snapshots $\mathsf{state}_1, \mathsf{state}_2$ and reveals her (rerandomized) accounts to $\mathtt{AA}$. Then she proves in zero-knowledge that the total amount sent in transactions in the time period defined by $\mathsf{state}_1, \mathsf{state}_2$ is less than x.

To check the compliance of a user with a policy, the $\mathtt{AA}$ executes the $0/1 \leftarrow \mathsf{VerifyAudit}(\mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}), \mathsf{auditInfo}, \mathtt{A}_1, \mathtt{A}_2)$ algorithm, which verifies π of $\mathsf{auditInfo}$ and returns its result.

It is important to note that if the $\mathtt{AA}$ colludes with the $\mathtt{RA}$, the only information gained is the ability to associate a user's real-world identity with their initial public key ($\mathtt{pk}_0$). This association could potentially be used to penalize non-compliant users outside the system's scope. However, such collusion does not provide the $\mathtt{AA}$ or $\mathtt{RA}$ with any additional knowledge about the user's accounts or transactions within the system, as the $\mathtt{RA}$ has no access to this information.

6 Security analysis

A private and auditable payment system should possess *anonymity*, *theft prevention*, *audit soundness* and *audit privacy* in order to be secure. We formally define these properties using security games. Our adversary can corrupt users of the system, create, delete and register new accounts, issue transactions, and request and receive audit proofs, through access to the following oracles:

- $\mathsf{sk} \leftarrow \mathsf{OCorrupt}(\mathsf{pk}, \mathsf{state})$: Returns the secret key for a public key of the provided state.
- $\mathsf{state} \leftarrow \mathsf{ORegister}()$: Used by the adversary to register a new user. The oracle 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.
- $(\mathsf{tx}_{\mathsf{DA}}, \mathsf{state}) \leftarrow \mathsf{ODelAcct}(\mathsf{userInfo}, \mathsf{acct}_{\mathsf{C}}, \mathsf{acct}_{\mathsf{D}}, \mathtt{A}_1, \mathtt{A}_2)$: Creates and applies a transaction to delete an account. 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 to send or receive funds, returns the transaction and the new state.
- $\mathsf{state} \leftarrow \mathsf{OApplyTrans}(\mathsf{tx})$: Can be used by the adversary to apply maliciously generated transactions to the state. The oracle checks if the 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 security games make use of bookkeeping functionalities which 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 $\mathsf{entries}$ 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 $\mathsf{corrupt}$, 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.
- $s \leftarrow \mathsf{totalWealth}(\mathsf{set}, \mathsf{state})$: 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 $\mathsf{set} = \mathsf{corrupt}$).
- $0/1 \leftarrow \mathsf{verifyPolicy}(\mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}))$: Checks whether $\mathtt{pk}_0$ is compliant with policy f for the time period represented by $\mathsf{state}_1, \mathsf{state}_2$.

We provide formal descriptions in Appendix C.1 and Appendix C.2.

6.1 Anonymity

Anonymity requires that an observer of the system cannot find the identities of senders and 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. Anonymity is defined in Game 1 and Definition 1, where the following rules must be enforced or else the adversary could trivially guess b.

- Both senders must be honest. If one of the senders is corrupt, the adversary would be able to see whose account's balance decreases.
- Both receivers are honest, or both are corrupt and $\mathsf{acct}'_0 = \mathsf{acct}'_1$ and $v_0 = v_1$. If only one is corrupt, the adversary would be able to see which account's balance increased. If both are corrupt but accounts or amounts are different, the adversary would be able to see which account's balance increased or by how much.

Game 1: Anonymity game $\mathsf{Exp}^{\text{anon}}_{\mathcal{A}}(\lambda)$

```
Input  : $\lambda$
Output: $\{0,1\}$
$b \leftarrow\!\!\$ \{0,1\}$
$(\mathsf{state}_0, \mathsf{pp}) \leftarrow \mathsf{Setup}(\lambda)$
$(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1, \mathtt{A}, v_0, v_1) \leftarrow$
  $\mathcal{A}^{\mathsf{OCorrupt},\mathsf{ORegister},\mathsf{OCreateAcct},\mathsf{ODelAcct},\mathsf{OTrans},\mathsf{OApplyTrans}}(\mathsf{state}_0)$
$\mathsf{state} \leftarrow \mathsf{states}[-1]$                    // 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 $(\mathsf{sk}_0 \in \mathsf{corrupt} \lor \mathsf{sk}_1 \in \mathsf{corrupt}) \lor ((\mathsf{sk}'_0 \in \mathsf{corrupt} \lor \mathsf{sk}'_1 \in \mathsf{corrupt}) \land ((\mathsf{acct}'_0 \neq$
  $\mathsf{acct}'_1) \lor (\mathsf{acct}'_0 = \mathsf{acct}'_1 \land v_0 \neq v_1))) \lor (\mathsf{acct}_0.\mathtt{bl} < v_0 \lor \mathsf{acct}_1.\mathtt{bl} < v_1)$ 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:* $\mathsf{Adv}^{\text{anon}}_{\mathcal{A}}(\lambda) =\mid \Pr[\mathsf{Exp}^{\text{anon}}_{\mathcal{A}}(\lambda) = 1] - \frac{1}{2}$. *A DPS satisfies* anonymity *if for every PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\text{anon}}_{\mathcal{A}}(\lambda)$ is negligible in λ.*

Theorem 1. *AQQUA satisfies anonymity.*

We formally prove Theorem 1, through a sequence of hybrid arguments (cf. Appendix C.3). 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 the UPK scheme, the hiding property of the commitment scheme, or the zero-knowledge property of the NIZK arguments. Transactions consist of $\mathtt{inputs}, \mathtt{outputs}$, and NIZK argument π (and if the transaction is the result of $\mathsf{CreateAcct}$ or $\mathsf{DelAcct}$ a newly created account acct). One way $\mathcal{A}$ could determine b is based on π, but that violates the zero-knowledge property of the NIZK arguments. Another way that $\mathcal{A}$ could determine b is to distinguish between $\mathsf{tx}_0, \mathsf{tx}_1$ through the $\mathtt{outputs}$ sets of each tx. The only differences in the two $\mathtt{outputs}$ sets $\mathsf{tx}_0.\mathtt{outputs}, \mathsf{tx}_1.\mathtt{outputs}$ are the accounts which are used in $\mathtt{P} = \mathtt{S} \cup \mathtt{R}$ ($\mathtt{P}_0, \mathtt{P}_1$ respectively) and in $\mathtt{A}$ ($\mathtt{A}_0, \mathtt{A}_1$ respectively) as well as the amount v used to increase/decrease the variables in the accounts of $\mathtt{P}$. However, since both the accounts' amounts and transferred value v are presented in a committed form, if $\mathcal{A}$ could determine b based on the different values v_0, v_1 then the hiding property of the commitment scheme would be 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 the UPK scheme.

6.2 Theft prevention

Theft prevention means that users can only move funds from accounts they own. It is formally defined in Game 2 and Definition 2. 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).

Game 2: Theft prevention game $\mathsf{Exp}_{\mathcal{A}}^{\text{theft}}(\lambda)$

```
Input  : $\lambda$
Output: $\{0, 1\}$
$(\mathsf{state}_0, \mathsf{pp}) \leftarrow \mathsf{Setup}(\lambda)$
$\mathsf{tx} \leftarrow \mathcal{A}^{\mathsf{OCorrupt}, \mathsf{ORegister}, \mathsf{OCreateAcct}, \mathsf{ODelAcct}, \mathsf{OTrans}, \mathsf{OApplyTrans}}(\mathsf{state}_0)$
$\mathsf{state} \leftarrow \mathsf{states}[-1]$                // 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 $(s'_h < s_h) \vee (s'_c > s_c) \vee (s'_c + s'_h \neq s_c + s_h)$
```

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

Theorem 2. *AQQUA satisfies theft prevention.*

We formally prove Theorem 2 in Appendix C.4. 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 the UPK scheme, the binding property of the commitment scheme, or the soundness property of the NIZK arguments. 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: Firstly, 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 NIZK argument 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 NIZK argument or break the binding property of the commitment scheme.

6.3 Audit soundness

Audit soundness means that there cannot be a successfully verified audit generated by a user who is non-compliant. Our definition in Game 3 is inspired from verifiability in electronic voting [12]. 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.

Game 3: Audit soundness game $\mathsf{Exp}^{\text{ausound}}_{\mathcal{A},f}(\lambda)$

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

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.*

We formally prove that AQQUA possesses audit soundness in Theorem 3. 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 arguments.

Theorem 3. *AQQUA satisfies audit soundness.*

Proof. Assume that there exist a PPT $\mathcal{A}$ that wins the audit soundness game of Game 3 with non-negligible probability. Thus, $\mathcal{A}$ outputted a proof $\pi = (\pi_p, \pi_1, \pi_2)$ that verifies, but $\mathcal{A}$ is not compliant with the specified policy.

$\mathcal{A}$ chooses a policy f with its auxiliary parameters $\mathtt{aux}$, an initial public key pk_0 and two snapshots from the blockchain $\mathsf{state}_1, \mathsf{state}_2$. Then $\mathcal{A}$ constructs $\pi = (\pi_p, \pi_1, \pi_2)$ which as defined in Figure 7 is a ZK-proof for the correct shuffle of the accounts and the relations $R_1(x, w)$, with $x = (\mathsf{pk}_0, \{\#\mathsf{accs}_j, \boxed{\#\mathsf{accs}_j}, \{\mathsf{acct}_{ji}\}_{i=1}^{\#\mathsf{accs}_j}\}_{j=1}^{2})$ and $w = (\mathsf{sk})$ and $R_2(x, w)$, with $x = (\{\mathsf{acct}_{1i}\}_{i=1}^{\#\mathsf{accs}_j}, \{\mathsf{acct}_{2i}\}_{i=1}^{\#\mathsf{accs}_j}, \boxed{\mathtt{v}}, \mathtt{aux})$ and $w = (\mathsf{sk}, \mathtt{v})$, where $\mathtt{v}, \mathtt{aux}$ are values that depend on the policy.

The soundness property of the correctness of the shuffle [3] prevents the adversary from altering the values of the initial accounts.

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 $\mathsf{pk} \in \{\mathsf{pk}_0\} \cup \{\mathsf{acct}_{ji}.\mathsf{pk}\}_{i=1}^{\#\mathsf{accs}_j}$, $\mathsf{VerifyKP}(\mathsf{sk}^*, \mathsf{pk})$. Therefore similarly to the theft-prevention proof (cf. Appendix C.4) 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 $\boxed{\#\mathtt{accs}}$ 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 $w^* = \mathtt{v}^*$ such that $R_1(x, w^*) = 1$. Again since $\mathcal{A}$ wince the game the sum of the openings of the committed 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. □

6.4 Audit Privacy

Audit privacy prevents an adversarial auditor from using the knowledge gained through the audit process to break the privacy of the users *after* one or more audits. We require that an auditor cannot find the identities of senders and receivers of a transaction that takes place after the time period of the audits if they don't own the sender's private key. Audit privacy is defined in Definition 4 and Game 4-a slight modification of Game 1.

Definition 4. *The* advantage *of the adversary in winning the audit privacy game is defined as:* $\mathsf{Adv}_{\mathcal{A}}^{\text{audpriv}}(\lambda) =\mid \Pr[\mathsf{Exp}_{\mathcal{A}}^{\text{audpriv}}(\lambda) = 1] - \dfrac{1}{2}$. *A DPS satisfies* audit privacy *if for every PPT adversary $\mathcal{A}$,* $\mathsf{Adv}_{\mathcal{A}}^{\text{audpriv}}(\lambda)$ *is negligible in λ.*

We prove that AQQUA satisfies audit privacy in Theorem 4. Intuitively, we argue that the audit privacy property is satisfied if the anonymity property holds and the audit process does not reveal the secret keys of the senders or receivers involved in a transaction.

Theorem 4. *AQQUA satisfies audit privacy.*

Proof. Assume a PPT adversary $\mathcal{A}$ that wins the audit privacy game with non-negligible probability. We will create an algorithm $\mathcal{B}$ that breaks the anonymity of AQQUA with the same non-negligible probability using $\mathcal{A}$ as a subroutine.

Game 4: Audit Privacy game $\mathsf{Exp}_{\mathcal{A}}^{\text{audpriv}}(\lambda)$

```
Input  : $\lambda$
Output: $\{0,1\}$
$b \leftarrow\!\$ \{0,1\}$
$(\mathsf{state}_0, \mathtt{pp}) \leftarrow \mathsf{Setup}(\lambda)$
$\{(\mathsf{state}_1, \mathsf{state}_2, \mathtt{pk}_0, f, \mathtt{aux}, \mathtt{A}_1, \mathtt{A}_2)\}^{poly(\lambda)} \leftarrow$
  $\mathcal{A}^{\mathsf{OCorrupt},\mathsf{ORegister},\mathsf{OCreateAcct},\mathsf{ODelAcct},\mathsf{OTrans},\mathsf{OApplyTrans}}(\mathsf{state}_0)$
$\mathsf{state} \leftarrow \mathsf{states}[-1]$                    // most recent state of bookkeeping
for $i = 1, \ldots, poly(\lambda)$ do
    $\mathsf{sk}_i \leftarrow \mathsf{findSecretKey}(\mathtt{pk}_{0i}, \mathsf{state})$
    $\mathsf{auditInfo}_i \leftarrow \mathsf{PrepareAudit}(\mathsf{sk}_i, \mathtt{pk}_{0i}, \mathsf{state}_1, \mathsf{state}_2, f_i, \mathtt{aux}_i, \mathtt{A}_{1i}, \mathtt{A}_{2i})$
$(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1, \mathtt{A}, v_0, v_1) \leftarrow$
  $\mathcal{A}^{\mathsf{OCorrupt},\mathsf{ORegister},\mathsf{OCreateAcct},\mathsf{ODelAcct},\mathsf{OTrans},\mathsf{OApplyTrans}}(\mathsf{states}, \{\mathsf{auditInfo}\}^{poly(\lambda)})$
$\mathsf{state} \leftarrow \mathsf{states}[-1]$
$\mathsf{sk}_0 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}_0.\mathsf{pk}, \mathsf{state})$; $\mathsf{sk}_1 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}_1.\mathsf{pk}, \mathsf{state})$
$\mathsf{sk}'_0 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}'_0.\mathsf{pk}, \mathsf{state})$; $\mathsf{sk}'_1 \leftarrow \mathsf{findSecretKey}(\mathsf{acct}'_1.\mathsf{pk}, \mathsf{state})$
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 $(b = b')$
```

The reduction works as follows: $\mathcal{B}$ takes as input the initial state state_0 and gives it to $\mathcal{A}$. Whenever $\mathcal{A}$ queries one of the oracles, $\mathcal{B}$ will forward the query to the corresponding oracle, update its state and return the answer to $\mathcal{A}$. When $\mathcal{A}$ asks for an audit proof by giving $(\mathsf{state}_1, \mathsf{state}_2, \mathtt{pk}_0, f, \mathtt{aux}, \mathtt{A}_1, \mathtt{A}_2)$, $\mathcal{B}$ uses the ZK simulator to construct it and sends it to $\mathcal{A}$. By the zero-knowledge property of the audit procedure, with overwhelming probability $\mathcal{A}$ cannot distinguish between real and simulated proofs. When $\mathcal{A}$ outputs $(\mathsf{acct}_0, \mathsf{acct}_1, \mathsf{acct}'_0, \mathsf{acct}'_1, \mathtt{A}, v_0, v_1)$, $\mathcal{B}$ will send these to the challenger of the anonymity game, which will return state'. $\mathcal{B}$ will give this state to $\mathcal{A}$, which will return the answer to b. $\mathcal{B}$ will return b to the challenger of the anonymity game. □

7 Performance

We follow the approach of Quisquis [17] to reason about the performance of AQQUA. Each AQQUA account consists of 8 group elements, twice the 4 used in [17], resulting in doubling the number of group elements required per transaction.

Thus, AQQUA transactions contain $48n$ group elements, compared to $24n$ in [17], where n represents the number of accounts in the `inputs` or `outputs` list. Using the same elliptic curve as [17], with each group element requiring 33 bytes, we project that an AQQUA transaction requires $1584n$ bytes.

In AQQUA, the inclusion of the new variables `out`, `in` results in each proof requiring additional Σ-protocols to verify both the correct shuffling of the commitments to these values and the proper updating of these values, as described in the transaction algorithm. Specifically, to implement this, each proof uses $4n+2|\mathtt{S}|+2\log_2(|\mathtt{S}|)+2\log_2(log_2(V))+8$ more group elements and $8n+4|\mathtt{S}|+9$ more field elements compared to Quisquis. Recall that the total size of the proof in Quisquis is $6n+22\sqrt{n}+52+2(\log_2(|\mathtt{S}|+|\mathtt{R}|)+\log_2(\log_2(V)))$ group elements and $6n+10\sqrt{n}+39$ field elements. Thus, asymptotically, the transaction proofs in AQQUA and Quisquis require the same size.

Regarding the audit, AQQUA requires $O(\sqrt{|\mathtt{A} \cup S|})$ group and field elements for shuffling, where $\mathtt{A}$ is the anonymity set and S the set of accounts owned by the audited user. Additionally, the audit involves a constant number of group and field elements equal to the number of accounts owned by the audited user, alongside $2\log_2(\log_2(V))+4$ group elements and 5 field elements.

8 Conclusion and Future Work

In this work we presented AQQUA, a decentralized private and auditable payment system. AQQUA accounts allow the aggregation of the total influx/outflux for an updatable public key while maintaining privacy. AQQUA authorities allow checking compliance to specific policies without intervening in the normal flow of transactions. Auditing may be voluntary and follows minimal information disclosure practices. External mechanisms can be used to prevent $\mathtt{AA}$ from abusing its power and requesting unnecessary audits. We also formally defined and proved security for AQQUA.

This work is a first step towards a general framework for non-invasive but auditable and private cryptocurrencies. In this regard, we plan to explore applying the AQQUA architecture to other cryptocurrencies beyond Quisquis (i.e. in [1]). We also plan to explore more policies that can be supported by AQQUA and to address in a game-theoretic manner the motivation of users to participate in audits. Additionally, we aim to provide ways to strengthen user privacy in relation to the size of the anonymity set and its sampling, without disregarding their effect on performance. Finally, another direction we are considering is to convert audit proofs to be designated-verifier [19]. As a result, the $\mathtt{AA}$ 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, a thorough consideration and formal modelling of the motives and actions of the $\mathtt{AA}$ is required.

Acknowledgement

This work has been partially supported by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the European Union under the NextGenerationEU Program.

References

[1] Jayamine Alupotha, Mathieu Gestin, and Christian Cachin. *Nopenena Untraceable Payments: Defeating Graph Analysis with Small Decoy Sets.* Cryptology ePrint Archive, Paper 2024/903. 2024. URL: https://eprint.iacr.org/2024/903.

[2] Elli Androulaki, Jan Camenisch, Angelo De Caro, Maria Dubovitskaya, Kaoutar Elkhiyaoui, and Björn Tackmann. "Privacy-preserving auditable token payments in a permissioned blockchain system". In: *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies.* AFT '20. New York, NY, USA: Association for Computing Machinery, 2020, 255–267. ISBN: 9781450381390. DOI: 10.1145/3419614.3423259. URL: https://doi.org/10.1145/3419614.3423259.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Vitalik Buterin, Jacob Illum, Matthias Nadler, Fabian Schär, and Ameen Soleimani. "Blockchain privacy and regulatory compliance: Towards a practical equilibrium". In: *Blockchain: Research and Applications* 5.1 (2024), p. 100176. ISSN: 2096-7209. DOI: https://doi.org/10.1016/j.bcra.2023.100176. URL: https://www.sciencedirect.com/science/article/pii/S2096720923000519.

[8] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. *Zether: Towards Privacy in a Smart Contract World.* Cryptology ePrint Archive, Paper 2019/191. 2019. URL: https://eprint.iacr.org/2019/191.

[9] 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.* Vol. 12727. Lecture Notes in Computer Science. 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.

[10] 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.

[11] 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.

[12] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. "Election Verifiability for Helios under Weaker Trust Assumptions". In: *ESORICS 2014.* Cham, 2014, pp. 327–344.

[13] 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.

[14] 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.

[15] Muhammed F. Esgin, Raymond K. Zhao, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. "MatRiCT: Efficient, Scalable and Post-Quantum Blockchain Confidential Transactions Protocol". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, 567–584. ISBN: 9781450367479. DOI: `10.1145/3319535.3354200`. URL: `https://doi.org/10.1145/3319535.3354200`.

[16] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. "On the Non-malleability of the Fiat-Shamir Transform". In: *Progress in Cryptology - INDOCRYPT 2012*. Ed. by Steven Galbraith and Mridul Nandi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 60–79. ISBN: 978-3-642-34931-7.

[17] 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.

[18] 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`.

[19] 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.

[20] 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`.

[21] 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`.

[22] 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`.

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

[24] 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.

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

[26] 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.

[27] 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.

[28] Karl Wüst, Kari Kostiainen, Noah Delius, and Srdjan Capkun. "Platypus: A Central Bank Digital Currency with Unlinkable Transactions and Privacy-Preserving Regulation". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. CCS '22. Los Angeles, CA, USA: Association for Computing Machinery, 2022, 2947–2960. ISBN: 9781450394505. DOI: 10.1145/3548606.3560617. URL: https://doi.org/10.1145/3548606.3560617.

[29] Karl Wüst, Kari Kostiainen, Vedran vCapkun, and Srdjan vCapkun. "PRCash: Fast, Private and Regulated Transactions for Digital Currencies". In: *Financial Cryptography and Data Security: 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers*. St. Kitts, Saint Kitts and Nevis: Springer-Verlag, 2019, 158–178. ISBN: 978-3-030-32100-0. DOI: 10.1007/978-3-030-32101-7_11. URL: https://doi.org/10.1007/978-3-030-32101-7_11.

A Building Blocks

A.1 Updatable Public Keys

AQQUA accounts are built on the Updatable Public Key (UPK) primitive from [17] In a UPK scheme public keys can be updated while remaining indistinguishable from freshly generated keys.

More specifically, a UPK scheme is a tuple $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Update}, \mathsf{VerifyKP}, \mathsf{VerifyUpdate})$.

- 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 $\{\mathsf{pk}_i\}_{i=1}^n$ and generates a new set $\{\mathsf{pk}'_i\}_{i=1}^n$ 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 properties next, formally defined in [17]:

- **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 the construction of section 3 satisfies correctness, indistinguishability and unforgeability [17].

A.2 *Σ*-protocols

Let R be a binary relation for instances x and witnesses w, and let $\mathcal{L}$ be its corresponding language, i.e. $\mathcal{L} = \{x | \exists w : (x, w) \in R\}$. A Σ-protocol for 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 $(x, w) \in R$.
- **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.

Additionally we utilize the following Σ-protocols defined in [17] and repeated below for convenience:

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

Prover$(\mathsf{pk}, \mathsf{pk}', w)$		**Verifier**$(\mathsf{pk}, \mathsf{pk}')$
$s \leftarrow\!\!\$\ \mathbb{F}_p$		
$\alpha \leftarrow \mathsf{pk}^s = (g^s, h^s)$	$\xrightarrow{\alpha}$	
	$\xleftarrow{c}$	$c \leftarrow\!\!\$\ \{0,1\}^\kappa$
$z \leftarrow cw + s$	$\xrightarrow{z}$	
		Check $\mathsf{pk}^z = (\mathsf{pk}')^c \cdot \alpha$

- 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 $\mathtt{com}_1 = \mathsf{Commit}(\mathsf{pk}_1, v; r_1), \mathtt{com}_2 = \mathsf{Commit}(\mathsf{pk}_2, v; r_2)$.

	$\mathsf{pk}_1 = (g_1, h_1), \mathtt{com}_1 = (c_1, d_1)$	
	$\mathsf{pk}_2 = (g_2, h_2), \mathtt{com}_2 = (c_2, d_2)$	
Prover(v, r_1, r_2)		**Verifier**
$v', r_1', r_2' \leftarrow\!\!\$\ \mathbb{F}_p$		
$(e_1, f_1) \leftarrow (g_1^{r_1'}, g^{v'} h_1^{r_1'})$		
$(e_2, f_2) \leftarrow (g_2^{r_2'}, g^{v'} h_2^{r_2'})$	$\xrightarrow{e_1, f_1, e_2, f_2}$	
	$\xleftarrow{x}$	$x \leftarrow\!\!\$\ \{0,1\}^\kappa$
$(z_v, z_{r_1}, z_{r_2}) \leftarrow x(v, r_1, r_2) + (v', r_1', r_2')$	$\xrightarrow{z_v, z_{r_1}, z_{r_2}}$	
		Check for $i = 1, 2$:
		$g_i^{z_{r_i}} = c_i^x \cdot e_i$
		$g^{z_v} h_i^{z_{r_i}} = d_i^x \cdot f_i$

B AQQUA Components

B.1 Transaction Algorithm

The detailed description of the Trans algorithm appears in Figure 4. The algorithm 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 $\mathtt{tx} = (\mathtt{inputs}, \mathtt{outputs}, \pi)$, where π is a NIZK argument that $\mathtt{outputs}$ is created correctly.

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 $\mathcal{V}$

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 $\mathtt{I_S} = \{1, \ldots, |\mathtt{S}|\}$. 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 $\mathtt{I_R} = \{|\mathtt{S}| + 1, \ldots, |\mathtt{S}| + |\mathtt{R}|\}$. 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) $\forall i \in \mathtt{I_S} : -\mathtt{v_{bl}}_i \in \mathcal{V} \wedge \mathtt{bl}_i + \mathtt{v_{bl}}_i \in \mathcal{V}$
4. Construct $\overrightarrow{\mathtt{v_{out}}}, \overrightarrow{\mathtt{v_{in}}}$ as follows:
   (a) $\overrightarrow{\mathtt{v_{out}}} = \overrightarrow{\mathtt{v_S}} || \underbrace{(0, \ldots, 0)}_{\text{length } |\mathtt{R}|} || \underbrace{(0, \ldots, 0)}_{\text{length } |\mathtt{A}|}$
   (b) $\overrightarrow{\mathtt{v_{in}}} = \underbrace{(0, \ldots, 0)}_{\text{length } |\mathtt{S}|} || \overrightarrow{\mathtt{v_R}} || \underbrace{(0, \ldots, 0)}_{\text{length } |\mathtt{A}|}$.
   Furthermore, expand $\overrightarrow{\mathtt{v_{bl}}}$ too with zero values for each $\mathsf{acct} \in \mathtt{A}$.
5. Sort $\mathtt{S} \cup \mathtt{R} \cup \mathtt{A}$ in some canonical order and store the result in $\mathtt{inputs}$. 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 $r_1, r_2, r_3, r_4 \leftarrow\$ \mathbb{F}^*_p$ and let $\vec{r} = (r_1, r_2, r_3, r_4)$.
   Perform $\mathsf{UpdateAcct}(\mathtt{inputs}, \overrightarrow{\mathtt{v_{bl}}}', \overrightarrow{\mathtt{v_{out}}}', \overrightarrow{\mathtt{v_{in}}}'; \vec{r})$ and sort the result in some canonical order. The results are assigned to $\mathtt{outputs}$.
7. Let $\psi : \{1, ..., \mathtt{n}\} \rightarrow \{1, ..., \mathtt{n}\}$ 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 NIZK argument π of the relation $R(x, w)$, where $x = (\mathtt{inputs}, \mathtt{outputs}), w = (\mathsf{sk}, \{\mathtt{bl}_i, \mathtt{out}_i, \mathtt{in}_i\}_{i \in \mathtt{I}^*_\mathtt{S}}, \overrightarrow{\mathtt{v_{bl}}}', \overrightarrow{\mathtt{v_{out}}}', \overrightarrow{\mathtt{v_{in}}}', \vec{r}, \psi, \mathtt{I}^*_\mathtt{S}, \mathtt{I}^*_\mathtt{R}, \mathtt{I}^*_\mathtt{A})$, and $R(x, w) = 1$ if

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

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

Fig. 4. The Trans algorithm.

5. the vectors $-\overrightarrow{\mathtt{v}_{\mathtt{bl}}}', \overrightarrow{\mathtt{v}_{\mathtt{out}}}'$ have the same values for the sender accounts and $\overrightarrow{\mathtt{v}_{\mathtt{bl}}}', \overrightarrow{\mathtt{v}_{\mathtt{in}}}'$ for the receivers accounts and $(\overrightarrow{\mathtt{v}_{\mathtt{out}}}', \overrightarrow{\mathtt{v}_{\mathtt{in}}}')$ have zero value for the rest.

The properties 3,4 can be proved by range proofs - i.e. with Bulletproofs [6]. For the properties 1,2,5 similarly to Quisquis[17] it holds that:

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 $\vec{\mathsf{acct}}_\delta$, where $\mathsf{acct}_{\delta,i} = (\mathsf{pk}_i, \boxed{\mathtt{v}_{\mathtt{bl}i}}, \boxed{\mathtt{v}_{\mathtt{out}i}}, \boxed{\mathtt{v}_{\mathtt{in}i}})$.

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 $\mathsf{pk}_\epsilon = (g,h)$, where $h = g^{\mathsf{sk}_\epsilon}$. So the prover creates $\vec{\mathsf{acct}}_\epsilon$ where $\mathsf{acct}_{\epsilon i} = ((g,h), \boxed{\mathtt{v}_{\mathtt{bl}i}}_\epsilon, \boxed{\mathtt{v}_{\mathtt{out}i}}_\epsilon, \boxed{\mathtt{v}_{\mathtt{in}i}}_\epsilon)$. Then they use the homomorphic property of the commitment in order to prove the preservation of value, since $\sum_i \mathtt{v}_{\mathtt{bl}i} = 0 \iff \prod_i \boxed{\mathtt{v}_{\mathtt{bl}i}}_\epsilon$ 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 $\mathtt{v}_{\mathtt{bl}\epsilon,s+1}, \ldots, \mathtt{v}_{\mathtt{bl}\epsilon,n} \in \mathcal{V}$. In addition, in order to prove property 5, the prover shows that for $\mathsf{acct}_{\epsilon,1}, \ldots, \mathsf{acct}_{\epsilon,s}$ the values under the $\boxed{\mathtt{v}_{\mathtt{bl}i}}_\epsilon$ and $\boxed{\mathtt{v}_{\mathtt{out}i}}_\epsilon$ are the opposite. Respectively for the recipients, for $\mathsf{acct}_{\epsilon,s+1}, \ldots, \mathsf{acct}_{\epsilon,t}$ the values under the $\boxed{\mathtt{v}_{\mathtt{bl}i}}_\epsilon$ and $\boxed{\mathtt{v}_{\mathtt{in}i}}_\epsilon$ are the same.

Now in order to hide the sender's and the receiver's position in `inputs` and `outputs` we first shuffle `inputs` list to `inputs′` before the updates, then we execute the updates to produce `outputs′`, and finally we shuffle again after the updates to get `outputs` in arbitrary order. The first shuffle uses the aforementioned permutation 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 $\mathtt{bl}, \mathtt{out}, \mathtt{in}$ 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 $r_{(\mathtt{bl}|\mathtt{out}|\mathtt{in}),1}, \ldots, r_{(\mathtt{bl}|\mathtt{out}|\mathtt{in}),n-1} \xleftarrow{\$} \mathbb{F}_p$ 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 $\mathsf{acct}_{\delta,i} = (\mathsf{pk}_i, \boxed{\mathtt{v}_{\mathtt{bl}i}}, \boxed{\mathtt{v}_{\mathtt{out}i}}, \boxed{\mathtt{v}_{\mathtt{in}i}})$.
  2. If $\prod_{i=1}^n \mathsf{acct}_{\epsilon,i}.\boxed{\mathtt{v}_{\mathtt{bl}}} = (1,1)$ and $\forall i \; \boxed{\mathtt{v}_{\mathtt{bl}i}} = \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{bl}i}; r_{\mathtt{bl},i}) \;\wedge\; \boxed{\mathtt{v}_{\mathtt{out}i}} = \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{out}i}; r_{\mathtt{out},i}) \;\wedge\; \boxed{\mathtt{v}_{\mathtt{in}i}} = \mathsf{Commit}(\mathsf{pk}_i, \mathtt{v}_{\mathtt{in}i}; r_{\mathtt{in},i})$ $\wedge\; \mathsf{VerifyAcct}(\mathsf{acct}_{\epsilon,i}, \mathsf{sk}_\epsilon, \mathtt{v}_{\mathtt{bl}i}, \mathtt{v}_{\mathtt{out}i}, \mathtt{v}_{\mathtt{in}i})$ output 1. Else output 0.

- $\mathsf{VerifyNonNegative}(\mathsf{acct}_\epsilon, v, r)$: Verifies that an account contains a balances in $\mathcal{V}$. More specifically, if $\mathsf{acct}_\epsilon.\mathsf{pk} = (g, h) \wedge \mathsf{acct}_\epsilon.\boxed{\mathtt{v}_{\mathtt{bl}}} = (g^r, g^v h^r) \wedge v \in \mathcal{V}$ 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 $\mathsf{acct}_i = (\mathsf{pk}_i, \mathtt{com}_{\mathtt{bl},i}, \mathtt{com}_{\mathtt{out},i}, \mathtt{com}_{\mathtt{in},i})$ 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 $\{(\mathsf{pk}_i, \mathtt{com}_{\mathtt{bl},i} \cdot \boxed{\mathtt{v}_{\mathtt{bl}i}}, \mathtt{com}_{\mathtt{out},i} \cdot \boxed{\mathtt{v}_{\mathtt{out}i}}, \mathtt{com}_{\mathtt{in},i} \cdot \boxed{\mathtt{v}_{\mathtt{in}i}})\}$, else output $\perp$.

- $\mathsf{VerifyUD}(\mathsf{acct}, \mathsf{acct}', \mathsf{acct}_\delta)$: Verifies that $\mathsf{UpdateDelta}$ was performed correctly:
  1. Parse $\mathsf{acct} = (\mathsf{pk}, \mathtt{com}_{\mathtt{bl}}, \mathtt{com}_{\mathtt{out}}, \mathtt{com}_{\mathtt{in}}), \mathsf{acct}' = (\mathsf{pk}, \mathtt{com}'_{\mathtt{bl}}, \mathtt{com}'_{\mathtt{out}}, \mathtt{com}'_{\mathtt{in}})$ 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 $\mathsf{pk} = \mathsf{pk}' = \mathsf{pk}_\delta \wedge \mathtt{com}'_{\mathtt{bl}} = \mathtt{com}_{\mathtt{bl}} \cdot \boxed{\mathtt{v}_{\mathtt{bl}}} \wedge \mathtt{com}'_{\mathtt{out}} = \mathtt{com}_{\mathtt{out}} \cdot \boxed{\mathtt{v}_{\mathtt{out}}} \wedge \mathtt{com}'_{\mathtt{in}} = \mathtt{com}_{\mathtt{in}} \cdot \boxed{\mathtt{v}_{\mathtt{in}}}$.
- $\mathsf{VerifyDeltaSender}(\mathsf{acct}_\epsilon, v, r_{\mathtt{bl}}, r_{\mathtt{out}})$: Verifies that sender's value $\mathtt{out}$ is correct.
  1. Parse $\mathsf{acct}_\epsilon = ((g, h), \boxed{\mathtt{v}_{\mathtt{bl}}}_\epsilon, \boxed{\mathtt{v}_{\mathtt{out}}}_\epsilon, \boxed{\mathtt{v}_{\mathtt{in}}}_\epsilon)$.
  2. If $\boxed{\mathtt{v}_{\mathtt{bl}}}_\epsilon = \mathsf{Commit}((g, h), -v; r_{\mathtt{bl}}) \wedge \boxed{\mathtt{v}_{\mathtt{out}}}_\epsilon = \mathsf{Commit}((g, h), v; r_{\mathtt{out}})$ then return 1. Else return 0.
- $\mathsf{VerifyDeltaReceiver}(\mathsf{acct}_\epsilon, v, r_{\mathtt{bl}}, r_{\mathtt{in}})$: Verifies that receiver's value $\mathtt{in}$ is correct.
  1. Parse $\mathsf{acct}_\epsilon = ((g, h), \boxed{\mathtt{v}_{\mathtt{bl}}}_\epsilon, \boxed{\mathtt{v}_{\mathtt{out}}}_\epsilon, \boxed{\mathtt{v}_{\mathtt{in}}}_\epsilon)$.
  2. If $\boxed{\mathtt{v}_{\mathtt{bl}}}_\epsilon = \mathsf{Commit}((g, h), v; r_{\mathtt{bl}}) \wedge \boxed{\mathtt{v}_{\mathtt{in}}}_\epsilon = \mathsf{Commit}((g, h), v; r_{\mathtt{in}})$ 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 $(\{\mathsf{acct}_{\delta,i}\}, \{\mathsf{acct}_{\epsilon,i}\}, r_{\vec{\mathtt{bl}}}, r_{\vec{\mathtt{out}}}, r_{\vec{\mathtt{in}}}) \leftarrow\$\ \mathsf{CreateDelta}(\mathtt{inputs}', \overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}})$;
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}))| \\
&\quad \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}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}, \vec{r_{\mathtt{bl}}}, \vec{r_{\mathtt{out}}}, \vec{r_{\mathtt{in}}}))| \\
&\quad \mathsf{VerifyUD}(\mathtt{inputs}'_i, \mathtt{outputs}'_i, \mathsf{acct}_{\delta,i}) = 1 \ \forall i \\
&\quad \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] \\
&\quad \wedge \mathsf{VerifyNonNegative}(\mathsf{acct}_{\epsilon,i}, \mathtt{v}_{\mathtt{bl}\,i}, r_{\mathtt{bl},i}) = 1 \ \forall i \in [s+1, t] \\
&\quad \wedge \mathsf{VerifyAcct}(\mathtt{outputs}'_i, (\mathsf{sk}, \mathtt{bl}_i + \mathtt{v}_{\mathtt{bl}\,i})) = 1 \ \forall i \in [1, s] \\
&\quad \wedge \mathsf{VerifyDelta}(\{\mathsf{acct}_{\delta,i}\}, \{\mathsf{acct}_{\epsilon,i}\}, \overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}, \vec{r_{\mathtt{bl}}}, \vec{r_{\mathtt{out}}}, \vec{r_{\mathtt{in}}}) = 1 \\
&\quad \wedge \mathsf{VerifyDeltaSender}(\mathsf{acct}_{\epsilon,i}, \mathtt{v}_{\mathtt{out}\,i}, r_{\mathtt{bl},i}, r_{\mathtt{out},i}) = 1 \ \forall i \in [1, s] \\
&\quad \wedge \mathsf{VerifyDeltaReceiver}(\mathsf{acct}_{\epsilon,i}, \mathtt{v}_{\mathtt{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'}))| \\
&\quad \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 [3]. 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 $\overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}, \vec{r_{\mathtt{bl}}}, \vec{r_{\mathtt{out}}}, \vec{r_{\mathtt{in}}}$ such that $\mathsf{VerifyDelta}(\{\mathsf{acct}_{\delta,i}\}_{i=1}^{n}, \{\mathsf{acct}_{\epsilon,i}\}_{i=1}^{n}, \overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}, \vec{r_{\mathtt{bl}}}, \vec{r_{\mathtt{out}}}, \vec{r_{\mathtt{in}}}) = 1$.
   Σ_δ can be implemented by using Σ_{com}:
   $\Sigma_\delta = \wedge_{i=1}^{n} \Sigma_{com}((\mathsf{pk}_{\delta,i}, \boxed{\mathtt{bl}}_{\delta,i}), (\mathsf{pk}_{\epsilon,i}, \boxed{\mathtt{bl}}_{\epsilon,i}); (\mathtt{v}_{\mathtt{bl}}, r_{\mathtt{bl},i}, r_{\mathtt{bl},i}))$
   $\wedge_{i=1}^{n} \Sigma_{com}((\mathsf{pk}_{\delta,i}, \boxed{\mathtt{out}}_{\delta,i}), (\mathsf{pk}_{\epsilon,i}, \boxed{\mathtt{out}}_{\epsilon,i}); (\mathtt{v}_{\mathtt{out}}, r_{\mathtt{out},i}, r_{\mathtt{out},i}))$
   $\wedge_{i=1}^{n} \Sigma_{com}((\mathsf{pk}_{\delta,i}, \boxed{\mathtt{in}}_{\delta,i}), (\mathsf{pk}_{\epsilon,i}, \boxed{\mathtt{in}}_{\epsilon,i}); (\mathtt{v}_{\mathtt{in}}, r_{\mathtt{in},i}, r_{\mathtt{in},i}))$, but the verifier additionally checks that $\forall i \ \mathsf{pk}_{\epsilon,i} = (g, h)$ and that $\prod_{i=1}^{n} \boxed{\mathtt{v}_{\mathtt{bl}\,i}}_{\epsilon} = (1, 1)$.
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 $\mathsf{acct}_1 = (\mathsf{pk}, \boxed{\mathtt{v}_{\mathtt{bl}}}_1, \boxed{\mathtt{v}_{\mathtt{out}}}_1, \boxed{\mathtt{v}_{\mathtt{in}}}_1), \mathsf{acct}_2 = (\mathsf{pk}, \boxed{\mathtt{v}_{\mathtt{bl}}}_2, \boxed{\mathtt{v}_{\mathtt{out}}}_2, \boxed{\mathtt{v}_{\mathtt{in}}}_2)$, the prover knows $r_{\mathtt{bl}}, r_{\mathtt{out}}, r_{\mathtt{in}}$ such that $\boxed{\mathtt{v}_{\mathtt{bl}}}_1 = \boxed{\mathtt{v}_{\mathtt{bl}}}_2 \cdot \mathsf{pk}^{r_{\mathtt{bl}}}, \boxed{\mathtt{v}_{\mathtt{out}}}_1 = \boxed{\mathtt{v}_{\mathtt{out}}}_2 \cdot \mathsf{pk}^{r_{\mathtt{out}}}, \boxed{\mathtt{v}_{\mathtt{in}}}_1 = \boxed{\mathtt{v}_{\mathtt{in}}}_2 \cdot \mathsf{pk}^{r_{\mathtt{in}}}$. 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 $\mathsf{acct}_{\epsilon,i}$ has the opposite value under commitments $\boxed{\mathtt{v}_{\mathtt{bl}}}, \boxed{\mathtt{v}_{\mathtt{out}}}$, denoted as $\mathtt{com}_1, \mathtt{com}_2$ respectively. This is equivalent to $\mathsf{VerifyUpdate}((g, h), \mathtt{com}_1 \cdot \mathtt{com}_2, r_{\mathtt{bl},i} + r_{\mathtt{out},i})$. Σ_{vds}^{i} can be implemented by using Σ_{vu}.
5. Σ_{vdr}^{i}: prover shows knowledge of $v, r_{\mathtt{bl},i}, r_{\mathtt{in},i}$ such that $\mathsf{acct}_{\epsilon,i}$ has the same value under commitments $\boxed{\mathtt{v}_{\mathtt{bl}}}, \boxed{\mathtt{v}_{\mathtt{in}}}$, denoted as $\mathtt{com}_1, \mathtt{com}_2$ respectively.

This is equivalent to $\mathsf{VerifyUpdate}((g,h), \frac{\mathtt{com}_1}{\mathtt{com}_2}, r_{\mathtt{bl},i} - r_{\mathtt{out},i})$. Σ^i_{vdr} can be implemented by using Σ_{vu}.

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 [6].
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 $\Sigma_2 = \Sigma_{vud} \wedge \Sigma_\delta \wedge \left(\wedge^t_{i=s+1} \Sigma_{range}(\mathsf{acct}_{\delta,i}, \mathtt{v}'_{\mathtt{bl}i}, r_{\mathtt{bl},i})\right) \wedge \left(\wedge^n_{i=t+1} \Sigma^i_{zero}\right) \wedge (\wedge^s_{i=1} \Sigma_{range,\mathsf{sk}}(\mathtt{outputs}'_i, \mathtt{bl}_i + \mathtt{v}_{\mathtt{bl}i}, \mathsf{sk})) \;\wedge\; \left(\wedge^s_{i=1} \Sigma^i_{vds}\right) \;\wedge \left(\wedge^t_{i=s+1} \Sigma^i_{vdr}\right)$. 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 $\Sigma := \Sigma_1 \wedge \Sigma_2 \wedge \Sigma_3$.

B.2 Algorithm to create accounts

The algorithm $\mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A})$ in Figure 5 performs the following steps:

1. Pick $r_1, r_2, r_3, r_4 \leftarrow\$ \mathbb{F}^*_p$ 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 $\mathsf{NewAcct}(\mathsf{userInfo}.\mathtt{pk}_0; \vec{r})$.
2. Let $\mathtt{inputs} = \{\mathsf{userInfo}\} \cup \mathtt{A}$ in some canonical order. Let $\mathtt{c}, \mathtt{I_A}$ 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: $\mathtt{v}_i = 0 \; \forall i \in \mathtt{I_A}$ and $\mathtt{v_c} = 1$.
4. Pick $r_5 \leftarrow\$ \mathbb{F}^*_p$ and let $\mathtt{outputs}$ be the output of $\mathsf{UpdateUser}(\mathtt{inputs}, \vec{\mathtt{v}}; r_5)$.
5. Form a NIZK argument 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 $\forall i \in \{\mathtt{c}\} \cup \mathtt{I_A}$, $\mathsf{userInfo}_i \in \mathtt{inputs}$, $\mathsf{userInfo}'_i \in \mathtt{outputs}$ we have that:

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

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

Fig. 5. The CreateAcct algorithm.

B.3 Delete Account Algorithm

The algorithm $\mathsf{DelAcct}(\mathsf{sk}, \mathsf{userInfo}, \mathsf{acct}_{\mathsf{D}}, \mathsf{acct}_{\mathsf{C}}, \mathtt{A}_1, \mathtt{A}_2)$ in Figure 6 performs the following steps:

1. Ensure that $\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{userInfo.pk}) = 1 \wedge \mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct}_{\mathsf{C}}.\mathsf{pk}) = 1 \wedge \mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct}_{\mathsf{D}}.\mathsf{pk}) = 1$.
2. For the account $\mathsf{acct}_{\mathsf{D}}$, calculate the opening of the commitments $\mathsf{acct}_{\mathsf{D}}.\mathtt{com}_{\mathtt{out}}, \mathsf{acct}_{\mathsf{D}}.\mathtt{com}_{\mathtt{in}}$, denoted $\mathtt{out}_{\mathsf{D}}, \mathtt{in}_{\mathsf{D}}$, using the secret key sk.
3. 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.
4. Construct $\overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}$ as follows:
   - $\overrightarrow{\mathtt{v}_{\mathtt{bl}}} = 0 \ \forall i \in \{c^*\} \cup \mathtt{I}_{\mathtt{A}1}$
   - $\overrightarrow{\mathtt{v}_{\mathtt{out}}} = 0 \ \forall i \in \mathtt{I}_{\mathtt{A}1}$ and $\mathtt{v}_{\mathtt{out}\,c^*} = \mathtt{out}_{\mathsf{D}}$
   - $\overrightarrow{\mathtt{v}_{\mathtt{in}}} = 0 \ \forall i \in \mathtt{I}_{\mathtt{A}1}$ and $\mathtt{v}_{\mathtt{in}\,c^*} = \mathtt{in}_{\mathsf{D}}$
5. Pick $r_1, r_2, r_3, r_4 \leftarrow\!\$\ \mathbb{F}_p^*$. and let $\vec{r} = (r_1, r_2, r_3, r_4)$. Let $\mathtt{outputs}_{\mathsf{UTXOSet}}$ be the output of $\mathsf{UpdateAcct}(\mathtt{inputs}_{\mathsf{UTXOSet}}, \overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}; \vec{r})$ in some canonical order.
6. Let $\psi : \{1, ..., \mathtt{n}\} \to \{1, ..., \mathtt{n}\}$ 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.
7. Form a NIZK argument 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
$$\begin{aligned}&\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct}_{\mathsf{D}}.\mathsf{pk}) = 1 \wedge \mathsf{VerifyKP}(\mathsf{sk}, \mathsf{acct}_{c^*}.\mathsf{pk}) = 1\\ &\wedge \mathsf{VerifyUpdateAcct}({\mathsf{acct}'}_{\psi(i)}, \mathsf{acct}_i, 0, 0, 0; \vec{r}) = 1 \ \forall i \in \mathtt{I}_{\mathtt{A}1}\\ &\wedge \mathsf{VerifyUpdateAcct}({\mathsf{acct}'}_{\psi(c^*)}, \mathsf{acct}_{c^*}, 0, \mathtt{out}_{\mathsf{D}}, \mathtt{in}_{\mathsf{D}}; \vec{r}) = 1\\ &\wedge \mathsf{VerifyCom}(\mathsf{acct}_{\mathsf{D}}.\mathsf{pk}, \mathsf{acct}_{\mathsf{D}}.\mathtt{com}_{\mathtt{bl}}, (\mathsf{sk}, 0)) = 1\end{aligned}$$
8. 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.
9. Construct $\vec{\mathtt{v}}$ as follows: $\mathtt{v}_i = 0 \ \forall i \in \mathtt{I}_{\mathtt{A}2}$ and $\mathtt{v}_{s^*} = -1$.
10. Pick $r \leftarrow\!\$\ \mathbb{F}_p^*$ and let $\mathtt{outputs}_{\mathsf{UserSet}}$ be the output of $\mathsf{UpdateUser}(\mathtt{inputs}_{\mathsf{UserSet}}, \vec{\mathtt{v}}; r)$.
11. Form a NIZK argument 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:
$$\begin{aligned}&\mathsf{VerifyKP}(\mathsf{sk}, \mathsf{userInfo}_{s^*}.\mathtt{pk}_0) = 1\\ &\wedge \mathsf{VerifyUpdateUser}(\mathsf{userInfo}'_i, \mathsf{userInfo}_i, 0; r) = 1 \ \forall i \in \mathtt{I}_{\mathtt{A}2}\\ &\wedge \mathsf{VerifyUpdateUser}(\mathsf{userInfo}'_{s^*}, \mathsf{userInfo}_{s^*}, -1; r) = 1\end{aligned}$$

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. 6. The $\mathsf{DelAcct}$ algorithm.

B.4 Auditing

The $\mathsf{PrepareAudit}$ algorithm in Figure 7 takes as input the user's secret key sk, the two blockchain snapshots $(\mathsf{state}_1, \mathsf{state}_2)$, the policy f along with the necessary information $\mathtt{aux}$ and two anonymity set for each snapshot.

Both the $\mathsf{Register}$ and $\mathsf{PrepareAudit}$ functionalities need a NIZK argument 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:

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

  This can be proven through $\varSigma_{dlog}$ 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 $\mathtt{com}$ to value v. The corresponding language can be written as:

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

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

The argument needed for $\mathsf{Register}$ results from the composition of these $\varSigma$-protocols and a range proof for showing that $bl \in \mathcal{V}$. The $\mathsf{PrepareAudit}$ proof uses the same combination of these $\varSigma$-protocols and appropriate range proofs for each policy $f_{\mathsf{slimit}}, f_{\mathsf{rlimit}}, f_{\mathsf{open}}, f_{\mathsf{txlimit}}, f_{\mathsf{np}}$.

C Security Proofs

C.1 Details of bookkeeping functionalities

The bookkeeping functionalities which are used in Game 1, Game 2 Game 3 are analyzed in Figure 5:

C.2 Details of oracles

The oracles where the adversary has access in Game 1, Game 2 Game 3 are analyzed in Figure 6:

Definition 5 ([17], [16]). *(Unbounded NIZK) Let L be an NP language, and let H be the hash function modeled as a random oracle RO. Then* $(\mathsf{P}^H, \mathsf{V}^H)$ *is an unbounded NIZK proof for language L if for all PPT adversary $\mathcal{A}$ there exists a PPT simulator* Sim *such that for all $x \in L$*

$$\left|\Pr\left[\mathcal{A}^{H(\cdot),\mathsf{P}(\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{RO(\cdot),\mathsf{Sim}(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

Definition 6 ([17], [16]). *(Weak simulation-extractable zero-knowledge). Informally, even if an adversary $\mathcal{A}$ has seen $q = poly(\lambda)$ simulated proofs, whenever $\mathcal{A}$ makes a fresh accepting proof, one can use this to extract a witness.*

It is proven in [16] that NIZK created from SHVZK arguments using the Fiat-Shamir transform are unbounded and weak simulation-extractable. We use these properties to prove the following theorems.

The algorithm $\mathsf{auditInfo} \leftarrow \mathsf{PrepareAudit}(\mathsf{sk}, \mathtt{pk}_0, \mathsf{state}_1, \mathsf{state}_2, (f, \mathtt{aux}), \mathtt{A}_1, \mathtt{A}_2)$ 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. Ensure that $\mathtt{A}_1, \mathtt{A}_2$ are subsets of the accounts in $\mathsf{state}_1.\mathsf{UTXOSet}, \mathsf{state}_2.\mathsf{UTXOSet}$ respectively.
3. For $j = 1, 2$, find the set of accounts $S_j = \{\mathsf{acct}_i\}_{i=1}^{\mathsf{\#accs}_j}$ from $\mathsf{state}_j.\mathsf{UTXOSet}$ that belong to the user. That is, $\forall \mathsf{acct} \in \mathsf{state}_j.\mathsf{UTXOSet}$, if $\mathsf{VerifyKP}(\mathsf{acct.pk}, \mathsf{sk}) = 1$, then add acct to S_j.
4. For $j = 1, 2$ re-randomize the accounts in $S_j \cup \mathtt{A}_j$. In particular:
   (a) Let $\mathtt{inputs}_j$ be all the accounts of $S_j \cup \mathtt{A}_j$.
   (b) Sample $r_1, r_2, r_3, r_4 \leftarrow\$ \mathbb{F}_p$ and let $\vec{r} = (r_1, r_2, r_3, r_4)$.
   (c) Compute $\mathtt{outputs}_j = \mathsf{UpdateAcct}(\mathtt{inputs}_j, \vec{0}, \vec{0}, \vec{0}; r)$.
   (d) Let $\psi : \{1, ..., |\mathtt{inputs}_j|\} \rightarrow \{1, ..., |\mathtt{inputs}_j|\}$ be the implicit permutation mapping $\mathtt{inputs}_j$ into $\mathtt{outputs}_j$; such that accounts $\mathsf{acct}_k \in \mathtt{inputs}_j$ and $\mathsf{acct}'_{\psi(k)} \in \mathtt{outputs}_j$ share the same secret key.
   (e) Form a NIZK argument π_p of the relation $R(x, w)$, where $x = (\mathtt{inputs}_j, \mathtt{outputs}_j), w = (\psi, r)$, and $R(x, w) = 1$ if $\mathsf{VerifyUpdateAcct}(\mathsf{acct}'_{\psi(k)}, \mathsf{acct}_k, 0, 0, 0; \vec{r}) = 1\ \forall k \in \{1, ..., |\mathtt{inputs}_j|\}$

   The re-randomized sets of accounts are $\mathtt{outputs}_1, \mathtt{outputs}_2$.
5. For $j = 1, 2$, find the set of accounts $S'_j = \{\mathsf{acct}'_i\}_{i=1}^{\mathsf{\#accs}_j}$ from $\mathtt{outputs}_j$ that belong to the user. That is, $\forall \mathsf{acct} \in \mathtt{outputs}_j$, if $\mathsf{VerifyKP}(\mathsf{acct.pk}, \mathsf{sk}) = 1$, then add acct to S'_j.
6. Form a NIZK argument π_1 of the relation $R(x, w)$, where $x = (\mathtt{pk}_0, \{\mathtt{\#accs}_j, \boxed{\mathtt{\#accs}_j}, \{\mathsf{acct}'_{ji}\}_{i=1}^{\mathsf{\#accs}_j}\}_{j=1}^{2}), w = (\mathsf{sk})$ and $R(x, w) = 1$ if:

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

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

4. For $j = 1, 2$ calculate $\boxed{\mathtt{out}^*_j} = \prod_{i=1}^{\mathsf{\#accs}_j} \mathsf{acct}_{ji}.\boxed{\mathtt{out}}, \boxed{\mathtt{in}^*_j} = \prod_{i=1}^{\mathsf{\#accs}_j} \mathsf{acct}'_{ji}.\boxed{\mathtt{in}}$. 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 NIZK argument π_2 of the relation $R(x, w)$ where $x = (\{\mathsf{acct}'_{1i}\}_{i=1}^{\mathsf{\#accs}_j}, \{\mathsf{acct}'_{2i}\}_{i=1}^{\mathsf{\#accs}_j}, \boxed{\mathtt{out}^*}, \boxed{\mathtt{in}^*}, \mathtt{aux}), w = (\mathtt{out}^*, \mathtt{in}^*)$ 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 $j = 1, 2$ calculate $\boxed{\mathtt{bl}^*_j} = \prod_{i=1}^{\mathsf{\#accs}_j} \mathsf{acct}'_{ji}.\boxed{\mathtt{bl}}$. 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 NIZK argument π_2 of the relation $R(x, w)$ where $x = (\{\mathsf{acct}'_{1i}\}_{i=1}^{\mathsf{\#accs}_j}, \{\mathsf{acct}'_{2i}\}_{i=1}^{\mathsf{\#accs}_j}, \boxed{\mathtt{bl}^*}, \mathtt{aux}), w = (\mathtt{bl}^*)$ and $R(x, w) = 1$ if:
$$f(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), \mathtt{aux}) = 1$$

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

Fig. 7. The $\mathsf{PrepareAudit}$ algorithm.

Algorithm 5: bookkeeping functionalities

```
entries $\leftarrow \emptyset$                                  // set of all secret keys
corrupt $\leftarrow \emptyset$                              // set of corrupt secret keys
honest $\leftarrow \emptyset$                                // set of honest secret keys
states $\leftarrow []$                   // list of states, updated through oracles
Function findSecretKey(pk, state)
    if state $\notin$ states then
        return $\perp$
    for sk $\in$ entries do
        for acct $\in$ state.UTXOSet do
            if acct.pk = pk $\wedge$ VerifyKP(sk, acct.pk) $= 1$ then
                return sk
        for userInfo $\in$ state.UserSet do
            if userInfo.$\mathtt{pk}_0$ = pk $\wedge$ VerifyKP(sk, userInfo.pk) $= 1$ then
                return sk
    return $\perp$
Function totalWealth(set, state)
    $s \leftarrow 0$
    for sk $\in$ set do
        for acct $\in$ state.UTXOSet do
            if VerifyKP(sk, acct.pk) then
                $s \leftarrow s +$ OpenCom(sk, acct.$\mathtt{com}_{\mathtt{bl}}$)
    return $s$
Function verifyPolicy($\mathtt{pk}_0$, $\mathsf{state}_1$, $\mathsf{state}_2$, $f$, aux)
    if $\mathsf{state}_1$, $\mathsf{state}_2$ $\notin$ states $\vee$ $\mathsf{state}_1$ is not older than $\mathsf{state}_2$ then
        return $\perp$
    $A_1, A_2 \leftarrow \emptyset, \emptyset$
    sk $\leftarrow$ findSecretKey($\mathtt{pk}_0$, $\mathsf{state}_1$)
                    // Find accounts owned by sk in $\mathsf{state}_1$.UTXOSet and
     $\mathsf{state}_2$.UTXOSet resp.
    for acct $\in$ $\mathsf{state}_1$.UTXOSet do
        if VerifyKP(sk, acct.pk) then
            $A_1 \leftarrow A_1 \cup \{$acct$\}$
    for acct $\in$ $\mathsf{state}_2$.UTXOSet do
        if VerifyKP(sk, acct.pk) then
            $A_2 \leftarrow A_2 \cup \{$acct$\}$
    if $f(\mathtt{pk}_0, (\mathsf{state}_1, \mathsf{state}_2), \mathtt{aux}) = 1$ then
                                        // Check if $f$ holds using $A_1, A_2$, sk
        return 1
    return 0
```

Algorithm 6: Oracles for security definitions

```
Oracle OCorrupt(pk, state)
    // pk should be a key of an account or user information in state, aborts otherwise
    sk $\leftarrow$ findSecretKey(pk, state)
    honest $\leftarrow$ honest $\setminus \{$sk$\}$
    corrupt $\leftarrow$ corrupt $\cup \{$sk$\}$
    return sk
Oracle ORegister()
    state $\leftarrow$ bookkeeping.states$[-1]$    // most recent state of bookkeeping
    (sk, userInfo, acct, $\pi$) $\leftarrow$ Register()
    if VerifyRegister(userInfo, acct, $\pi$, state) $= 0$ then
        return $\perp$          // cannot be registered given current state
    entries $\leftarrow$ entries $\cup \{$sk$\}$
    honest $\leftarrow$ honest $\cup \{$sk$\}$
    state$'$ $\leftarrow$ ApplyRegister(userInfo, acct, state); states $\leftarrow$ states $\cup$ [state$'$]
    return state$'$
Oracle OCreateAcct(userInfo, A)
    state $\leftarrow$ bookkeeping.states$[-1]$    // most recent state of bookkeeping
    tx$_{\mathsf{CA}}$ $\leftarrow$ CreateAcct(userInfo, A)
    if VerifyTrans(tx$_{CA}$, state) $= 0$ then
        return $\perp$            // transaction cannot be applied to state
    state$'$ $\leftarrow$ ApplyTrans(tx$_{\mathsf{CA}}$, state); states $\leftarrow$ states $\cup$ [state$'$]
    return tx$_{CA}$, state$'$
Oracle ODelAcct(userInfo, acct$_C$, acct$_D$, A$_1$, A$_2$)
    state $\leftarrow$ bookkeeping.states$[-1]$
    sk $\leftarrow$ findSecretKey(acct$_{\mathsf{C}}$)
    tx$_{\mathsf{DA}}$ $\leftarrow$ DelAcct(sk, userInfo, acct$_{\mathsf{C}}$, acct$_{\mathsf{D}}$, A$_1$, A$_2$)
    if VerifyTrans(tx$_{DA}$, state) $= 0$ then
        return $\perp$            // transaction cannot be applied to state
    state$'$ $\leftarrow$ ApplyTrans(tx$_{\mathsf{DA}}$, state); states $\leftarrow$ states $\cup$ [state$'$]
    return tx$_{DA}$, state$'$
Oracle OTrans(S, R, $\vec{v_{\mathtt{S}}}$, $\vec{v_{\mathtt{R}}}$, A)
    state $\leftarrow$ bookkeeping.states$[-1]$    // most recent state of bookkeeping
    for sk $\in$ entries do
        Take an arbitrary acct $\in$ S
        if VerifyKP(sk, acct.pk) $= 1$ then
            tx $\leftarrow$ Trans(S, R, $\vec{\mathtt{v_S}}$, $\vec{\mathtt{v_R}}$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 $\perp$     // transaction cannot be applied to state
            state$'$ $\leftarrow$ ApplyTrans(tx, state); states $\leftarrow$ states $\cup$ [state$'$]
            return tx, state$'$
    return $\perp$
Oracle OApplyTrans(tx)
    if VerifyTrans(tx, state) $= 0$ then
        return $\perp$
    state$'$ $\leftarrow$ ApplyTrans(tx, state)
    states $\leftarrow$ states $\cup$ [state$'$]; return state$'$
Oracle OPrepareAudit(pk$_0$, state$_1$, state$_2$, $f$, aux)
    sk $\leftarrow$ findSecretKey(pk$_0$, state$_1$)
    if state$_1$, state$_2$ $\in$ states $\wedge$ state$_1$ is older than state$_2$ then
        auditInfo $\leftarrow$ 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 $\perp$  // pk$_0$ was invalid for the snapshots, state$_1$, state$_2$ were not valid or $f$ was not satisfied
```

C.3 Full proof of anonymity

Before we give the proof of anonymity, we first recall a definition for indistinguishability of UPK scheme [17].

Definition 7. *The* advantage *of the adversary in winning the indistinguishability game is defined as:*

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

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

Game 7: Indistinguishability game $\mathsf{Exp}^{ind}_{\mathcal{A}}(\lambda)$

```
Input  : $\lambda$
Output: $\{0, 1\}$
$b \leftarrow \{0, 1\}$
$(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KGen}()$
$r \leftarrow\$ \mathbb{F}_p^*$
$\mathsf{pk}_0 \leftarrow \mathsf{Update}(\mathsf{pk}^*; r)$
$(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{KGen}()$
$b' \leftarrow \mathcal{A}(\mathsf{pk}^*, \mathsf{pk}_b)$
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 7 if the DDH assumption holds in $(\mathbb{G}, g, p)$.*

Proof of this lemma can be found in [17]. We are now ready to prove the anonymity property of AQQUA.

Proof (Theorem 1). We prove Theorem 1 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 $\mathsf{OTrans}(\mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v}_\mathtt{S}}, \overrightarrow{\mathtt{v}_\mathtt{R}}, \mathtt{A})$ Oracle, the Oracle runs $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v}_\mathtt{S}}, \overrightarrow{\mathtt{v}_\mathtt{R}}, \mathtt{A})$, 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 $\mathsf{tx}.\mathtt{outputs}$), the Oracle runs $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v}_\mathtt{S}}, \overrightarrow{\mathtt{v}_\mathtt{R}}, \mathtt{A})$, $(\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{KGen}$ 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 $\mathsf{acct}'_i = (\mathsf{pk}'_i, \mathsf{com}_{\mathtt{bl}i}, \mathsf{com}_{\mathtt{out}i}, \mathsf{com}_{\mathtt{in}i})$.

- 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 $\mathsf{tx}.\mathtt{outputs}$), the Oracle runs $\mathsf{tx} \leftarrow \mathsf{Trans}(\mathsf{sk}, \mathtt{S}, \mathtt{R}, \overrightarrow{\mathtt{v}_\mathtt{S}}, \overrightarrow{\mathtt{v}_\mathtt{R}}, \mathtt{A})$, $(r_1, r_2, r_3) \leftarrow\$ \mathbb{F}_p^*$, $\mathtt{bl}_i \leftarrow \mathsf{OpenCom}(\mathsf{sk}, \mathsf{acct}_i.\mathsf{com}_{\mathtt{bl}})$, $\mathtt{out}_i \leftarrow \mathsf{OpenCom}(\mathsf{sk}, \mathsf{acct}_i.\mathsf{com}_{\mathtt{out}})$, $\mathtt{in}_i \leftarrow \mathsf{OpenCom}(\mathsf{sk}, \mathsf{acct}_i.\mathsf{com}_{\mathtt{in}})$, $\mathsf{com}'_{\mathtt{bl}} \leftarrow \mathsf{Commit}(\mathsf{pk}', \mathtt{bl}_i; r_1)$, $\mathsf{com}'_{\mathtt{out}} \leftarrow \mathsf{Commit}(\mathsf{pk}', \mathtt{out}_i; r_2)$, $\mathsf{com}'_{\mathtt{in}} \leftarrow \mathsf{Commit}(\mathsf{pk}', \mathtt{in}_i; r_3)$ 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 $\mathsf{acct}' = (\mathsf{pk}, \mathsf{com}'_{\mathtt{bl}}, \mathsf{com}'_{\mathtt{out}}, \mathsf{com}'_{\mathtt{in}})$. ($\mathsf{pk} = \mathsf{pk}'$ as in the Hybrid 3).

- when $\mathcal{A}$ creates one of these accounts acct_i through the $\mathsf{OCreateAcct}$ Oracle, the Oracle runs $\mathsf{tx} \leftarrow \mathsf{CreateAcct}(\mathsf{userInfo}, \mathtt{A}), (r_1, r_2, r_3) \leftarrow\!\!\$\ \mathbb{F}_p^*, \mathsf{com}'_{\mathtt{bl}} \leftarrow \mathsf{Commit}(\mathsf{pk}'0; r_1), \mathsf{com}'_{\mathtt{out}} \leftarrow \mathsf{Commit}(\mathsf{pk}', 0; r_2), \mathsf{com}'_{\mathtt{in}} \leftarrow \mathsf{Commit}(\mathsf{pk}', 0; r_3)$ 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 $\mathsf{acct}' = (\mathsf{pk}, \mathsf{com}'_{\mathtt{bl}}, \mathsf{com}'_{\mathtt{out}}, \mathsf{com}'_{\mathtt{in}})$. ($\mathsf{pk} = \mathsf{pk}'$ 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 `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 `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 `inputs` and `outputs`) are fresh accounts, where in `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 $\mathsf{O}_{\mathsf{zk}}(\cdot)$ be an oracle that on input $(\mathsf{tx}.\mathtt{inputs}, \mathsf{tx}.\mathtt{outputs})$ creates a valid NIZK argument for the transaction. Then $\mathcal{B}$ wins if they can decide wether $\mathsf{O}_{\mathsf{zk}}(\cdot)$ is a prover or simulator oracle.

$\mathcal{B}$ takes as input the $\mathsf{O}_{\mathsf{zk}}(\cdot)$ 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 $\mathsf{O}_{\mathsf{zk}}(\mathsf{tx}[\mathtt{inputs}], \mathsf{tx}[\mathtt{outputs}])$
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 $\mathsf{O}_{\mathsf{zk}}(\mathsf{tx}[\mathtt{inputs}], \mathsf{tx}[\mathtt{outputs}])$

4. $\mathcal{B}$ runs $b \leftarrow \mathcal{A}(state)$;

If $\mathcal{A}$ answers Hybrid 0 then $\mathsf{O}_{\mathsf{zk}}(\cdot)$ is a prover oracle. If $\mathcal{A}$ answers Hybrid 1 then $\mathsf{O}_{\mathsf{zk}}(\cdot)$ 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 $\mathtt{inputs}$ 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.

C.4 Full proof of theft prevention

We now prove Theorem 2.

Proof (Theorem 2). Assume that there exists a PPT $\mathcal{A}$ that wins the theft prevention game of Game 2 with non-negligible probability. Using the notation of the game, we have that $\mathcal{A}$ outputted a valid transaction $\mathtt{tx}$ that verifies and that results in one of the three winning conditions of the game being satisfied.

We have that $\mathtt{tx} = (\mathtt{inputs}, \mathtt{outputs}, \pi)$, where π is a ZK-proof for the relation $R(x, w)$ as defined in Figure 4, with $x = (\mathtt{inputs}, \mathtt{outputs})$ and $w = (\mathsf{sk}, \mathtt{bl}, \mathtt{out}, \mathtt{in}, \overrightarrow{\mathtt{v}_{\mathtt{bl}}}, \overrightarrow{\mathtt{v}_{\mathtt{out}}}, \overrightarrow{\mathtt{v}_{\mathtt{in}}}, \overrightarrow{r}, \psi, \mathtt{I}^*_{\mathtt{S}}, \mathtt{I}^*_{\mathtt{R}}, \mathtt{I}^*_{\mathtt{A}})$.

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 $\mathtt{tx}$ 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 $\mathtt{tx}$ 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.