Nummatus: A Privacy Preserving Proof of Reserves Protocol for Quisquis

Arijit Dutta[1], Arnab Jana[2], and Saravanan Vijayakumaran[1]

[1] Department of Electrical Engineering
[2] Department of Computer Science and Engineering
Indian Institute of Technology Bombay, Mumbai, India
{arijit.dutta@, arnabjanacse@cse., sarva@ee.}iitb.ac.in

Abstract. Quisquis is a recently proposed design for a privacy-focused cryptocurrency. We present *Nummatus*, a privacy preserving proof of reserves protocol for Quisquis.[3] Nummatus enables exchanges to create a Pedersen commitment to the amount of Quisquis coins they own, without revealing the exact accounts they own. These commitments can be combined with a commitment to the total liabilities of an exchange to generate a proof of solvency. The Nummatus protocol also facilitates detection of account sharing collusion between exchanges. Our simulations show that the cost of using Nummatus instead of a non-private proof of reserves protocol is not prohibitive.

Keywords: Proof of Reserves · Quisquis · Cryptocurrency · Privacy in Blockchain.

1 Introduction

Cryptocurrency exchanges arose to enable cryptocurrency acquisition without mining. They provide custodial wallets and trading services to their customers. Custodial wallets not only free customers from the burden of storing private keys but also allow for more efficient trading (as they do not need trades to be recorded on the blockchain). The downside of custodial wallets is that customer funds are lost when the exchange gets hacked or when the exchange operators execute an exit scam. In both these undesirable scenarios, corrective measures can be more effective if the attacks are detected early.

Proofs of solvency can enable early detection of loss of customer funds from cryptocurrency exchanges [10,11,20]. These proofs show that the cryptocurrency reserves of an exchange exceed its liabilities (the amount of coins the exchange has sold to customers). Exchanges are more likely to provide proofs of solvency if they are privacy preserving, i.e. the proofs do not reveal which outputs or accounts on the blockchain belong to the exchange and they also do not reveal the actual amounts corresponding to the total reserves and total liabilities.

[3] *Quisquis* is Latin for "whoever, whatever" [18]. *Nummatus* is Latin for "moneyed, rich" [17]. We chose this name for our protocol as it enables an exchange to show that it is *rich enough* to meet its liabilities.

A popular method of providing proofs of solvency is using Pedersen commitments [10, 15]. First, a Pedersen commitment p_{liab} to the total amount of coins the exchange has sold to customers (the liabilities) is created. The protocol used to ensure that p_{liab} is in fact a commitment to the exchange's total liabilities is called a *proof of liabilities* protocol. Second, a Pedersen commitment p_{res} to the total amount of coins owned by the exchange (the reserves) is created. The protocol used to ensure that p_{res} is a commitment to the exchange's total reserves is called a *proof of reserves* protocol. Third, a range proof on the quantity $p_{\text{res}} p_{\text{liab}}^{-1}$ is provided to show that the difference between the reserves and liabilities is non-negative and in the correct range.[4] Proof of reserves protocols use the data available on the blockchain. Consequently, reasonable proof of reserves protocols have been proposed for Bitcoin [10], Monero [12], and Mimblewimble [13]. On the other hand, proof of liabilities protocols need to use an exchange's customer database to generate the proofs. As a malicious exchange can conceal some customer records to reduce its liabilities, the proofs of liabilities protocols proposed so far are not robust. In this paper, we limit our attention to a privacy preserving proof of reserves protocol for Quisquis [14].

Quisquis is a recently proposed design for privacy-focused cryptocurrency. It solves the problem of the always growing *unspent transaction output (UTXO)* set which plagues other privacy-focused cryptocurrencies like Monero [3] and Zcash [5]. While no reference implementation of Quisquis exists, the design is novel enough to warrant designing proof of reserves protocol for it. So when an implementation does emerge and the Quisquis cryptocurrency becomes available on exchanges, the Nummatus proof of reserves protocol can be employed in proofs of solvency.

Our contribution. We propose Nummatus, a proof of reserves protocol for Quisquis exchanges. It is, to the best of our knowledge, the first such protocol for Quisquis. Our protocol is privacy preserving in the sense that it only reveals that the exchange-owned accounts belong to a larger anonymity set of accounts, without identifying which ones are exchange-owned. The protocol gives a technique to detect collusion between exchanges who use the same account to generate their respective proofs of reserves. We also describe a non-private proof of reserves protocol for Quisquis exchanges called Simplus, with the intention of quantifying the cost of deploying a privacy preserving protocol. We give simulation results to compare the performance of the Nummatus and Simplus protocols. These simulations show that, while privacy has a cost, deploying Nummatus is a practical proposition.

2 Overview of Quisquis

Privacy-focused cryptocurrencies like Monero and Zcash allow users to conceal the source of coins in a transaction using ring signatures [16] or zk-SNARKs [4]. As the true source of coins is not revealed, a one-time address in Monero

[4] We present elliptic curve group operations in multiplicative notation to be consistent with the presentation in the Quisquis paper [14].

and a commitment in Zcash cannot be considered spent.[5] Consequently, these cryptocurrencies have poor scalability in the long term without the opportunity to prune spent outputs from the blockchain.

Quisquis is a cryptocurrency proposal which offers both privacy and scalability [14]. It is an account-based design (as opposed to a UTXO-based design), where each account consists of a public key and a commitment to the balance in the account. The public key is generated from the secret key and a randomizing scalar. Hence there are many possible public keys corresponding to a secret key (unlike Bitcoin where the public key is a deterministic function of the secret key).

Each Quisquis transaction involves some *input accounts* and an *equal* number of *output accounts*. Each output account is an updated version of exactly one of the input accounts, where the update consists of an update of the input account's public key (the account secret key remains unchanged) *and/or* an update of the input account's balance. Unlike Bitcoin where the input UTXOs in a transaction represent the source of funds and output UTXOs represent destinations, the input accounts in a Quisquis transaction consist of both source accounts and destination accounts. Additionally, some *passive accounts* are added to the list of input accounts in the transaction to obfuscate the sources and destinations of funds. Only the public keys of the passive accounts are updated in a transaction and their balances are unchanged. On the other hand, the balances of source accounts are reduced and the balances of destination accounts are increased. For both source and destination accounts, the public keys are updated. The output accounts are presented in a lexicographical order to prevent linking of specific output accounts with input accounts. Once a Quisquis transaction appears on the blockchain, the input accounts can be pruned. Quisquis has special transactions for creation and deletion of accounts. Regular transactions do not create new accounts and this is the main reason for the scalability of the design. In the following subsections, we present a more precise description of those aspects of Quisquis that are necessary to present Nummatus.

2.1 Quisquis Accounts

Let $\mathbb{G}$ be a group with prime order p and generator g. The Decisional Diffie-Hellman (DDH) problem is assumed to be hard in $\mathbb{G}$. A Quisquis account based on $\mathbb{G}$ is specified by four group elements (a, b, c, d). The first two group elements specify a *public key* $\mathtt{pk} = (a, b) = (g^t, g^{k\cdot t})$ where $t \in \mathbb{F}_p$ is an arbitrary scalar and $k \in \mathbb{F}_p$ is the *secret key*. The last two group elements specify a *commitment* which depends on $\mathtt{pk}$ and is given by $\mathtt{com} = (c, d) = (a^r, g^v b^r)$ where $r \in \mathbb{F}_p$ is an arbitrary scalar. Here $v \in \mathbb{F}_p$ is the value being committed to by $\mathtt{com}$. In summary, a Quisquis account is of the form

$$\mathtt{acct} = (a, b, c, d) = (a, b, a^r, g^v b^r) = \left(g^t, g^{k\cdot t}, g^{t\cdot r}, g^{v+k\cdot t\cdot r}\right) \tag{1}$$

[5] Source addresses in Monero transactions where the sender deliberately chose a ring size of one can be considered spent. But this kind of behavior is seen only in old transactions as the default ring size was set to 11 in October 2018 [2].

where k is the secret key, v is the value in the account, and r, t are arbitrary scalars.

In the Quisquis design, knowledge of the secret key k is sufficient to prove ownership of an account and to create transactions which transfer value out of it, i.e. knowledge of the scalars r and t is not required. This feature allows an entity to perform a *secret key preserving update* of an account, even when the entity has no knowledge of the secret key, value, or scalars used to create the account elements. An update of an account $\texttt{acct} = (a, b, c, d)$ to an account $\texttt{acct}' = (a_1, b_1, c_1, d_1)$ preserves the secret key k and changes the amount from v to $v + \delta$ if the following equations hold.

$$\begin{aligned} b &= a^k, \quad d = g^v c^k, \\ b_1 &= a_1^k, \quad d_1 = g^{v+\delta} c_1^k. \end{aligned} \tag{2}$$

A Quisquis transaction involves account updates of this kind in addition to range proofs on the values $v + \delta$ to ensure that the amount changes are valid.

The account update procedure is as follows:

1. Suppose $\texttt{acct}$ in (1) is the account to be updated.
2. The updater chooses scalars $t_1, r_1, \delta \in \mathbb{F}_p$. While t_1, r_1 are chosen arbitrarily, δ represents the change in the value stored in the account.
3. The updater computes the updated public key as
$$\texttt{pk}' = (a_1, b_1) = (a^{t_1}, b^{t_1}). \tag{3}$$
4. The updater computes the updated commitment as
$$\texttt{com}' = (c_1, d_1) = (ca^{r_1}, dg^\delta b^{r_1}). \tag{4}$$
Note that this update can be interpreted as the coordinate-wise product of the commitment (c, d) with the commitment $(a^{r_1}, g^\delta b^{r_1})$.
5. The updated account is $\texttt{acct}' = (\texttt{pk}', \texttt{com}') = (a_1, b_1, c_1, d_1)$.

It is easy to check that the equations in (2) hold. Since $b = a^k$, we have

$$b_1 = b^{t_1} = \left(a^k\right)^{t_1} = \left(a_1^t\right)^k = a_1^k. \tag{5}$$

Since $d = g^v c^k$ and $c_1 = ca^{r_1}$, we have

$$d_1 = dg^\delta b^{r_1} = g^v c^k g^\delta \left(a^k\right)^{r_1} = g^{v+\delta} \left(ca^{r_1}\right)^k = g^{v+\delta} c_1^k. \tag{6}$$

To see that the updated account has the structure specified in (1), consider the following version of $\texttt{acct}'$.

$$\begin{aligned} \texttt{acct}' = (a_1, b_1, c_1, d_1) &= \left(a^{t_1}, b^{t_1}, ca^{r_1}, dg^\delta b^{r_1}\right) & (7) \\ &= \left(g^{t \cdot t_1}, g^{k \cdot t \cdot t_1}, g^{t(r+r_1)}, g^{v+\delta+k \cdot t \cdot (r+r_1)}\right) & (8) \\ &= \left(g^{t'}, g^{k \cdot t'}, g^{t' \cdot r'}, g^{v+\delta+k \cdot t' \cdot r'}\right), & (9) \end{aligned}$$

where $t' = t \cdot t_1$ and $r' = t_1^{-1} \cdot (r + r_1)$.

In the subsequent discussion, we will find it convenient to denote the above account update procedure by the notation $\texttt{UpdateAcct}(\texttt{acct}, t_1, r_1, \delta)$.

2.2 Quisquis Transactions

Suppose Alice owns account $\texttt{acct}_1$ and wants to transfer δ amount to account $\texttt{acct}_2$. Alice will choose $n-2$ additional accounts from the blockchain which will play the role of passive accounts. Let these passive accounts be denoted by $\texttt{acct}_3$, $\texttt{acct}_4$, …, $\texttt{acct}_n$. Alice will construct a transaction with input accounts given by $\texttt{inputs} = \{\texttt{acct}_1, \texttt{acct}_2, \ldots, \texttt{acct}_n\}$. The input accounts will be listed in a canonical order like lexicographical ordering to conceal the identity of the non-passive accounts. Alice will update each of the input accounts to generate output accounts given by $\texttt{outputs} = \{\texttt{acct}'_1, \texttt{acct}'_2, \ldots, \texttt{acct}'_n\}$ where

$$\begin{aligned}
\texttt{acct}'_1 &= \texttt{UpdateAcct}(\texttt{acct}_1, t_1, r_1, -\delta),\\
\texttt{acct}'_2 &= \texttt{UpdateAcct}(\texttt{acct}_2, t_2, r_2, \delta),\\
\texttt{acct}'_3 &= \texttt{UpdateAcct}(\texttt{acct}_3, t_3, r_3, 0),\\
\vdots &= \qquad \vdots\\
\texttt{acct}'_n &= \texttt{UpdateAcct}(\texttt{acct}_n, t_n, r_n, 0),
\end{aligned}$$

where the t_is and r_is are arbitrarily chosen scalars. Note that the balance in the source account $\texttt{acct}_1$ is reduced by δ and the balance in the destination account $\texttt{acct}_2$ is increased by δ. The balances of the passive accounts remain the same. The output accounts will also be presented in a canonical ordering to conceal the mapping from the inputs to the outputs. Alice then constructs a zero knowledge proof π that convinces the verifier of the following statements.

1. Each account in `outputs` is an update of exactly one of the accounts in `inputs`.
2. The account updates cumulatively satisfy preservation of balances i.e. $\sum \delta_i = 0$, where δ_i is the update of balance of $\texttt{acct}_i$.
3. The balance of the source account does not become negative after the update.
4. The balance of the destination account after the update is in the correct range of values (range proof).
5. The balances of the passive accounts remain unchanged.

The transaction `txn` itself consists of the sets `inputs`, `outputs`, and the zero knowledge proof π, i.e. $\texttt{txn} = (\texttt{inputs}, \texttt{outputs}, \pi)$. While our illustrative example had only one source account and one destination account, transactions with multiple sources and destinations are allowed in Quisquis.

The implication of this transaction model to our context is that exchange-owned accounts may be updated several times before they are used in the proof of reserves protocol. If the exchange is not involved in all the updates of an account, it will not know the discrete logarithm of the group elements forming the public key and commitment with respect to the generator g. This fact has to be taken into consideration in the proof of reserves protocol design.

3 Nummatus Proof of Reserves Protocol

The overall design of the Nummatus protocol is similar to the proof of reserves protocols which have appeared in the literature [10,12,13]. However, unlike these previously proposed protocols, the Nummatus protocol requires a sequence of elements $h_1, h_2, h_3, \ldots$ from $\mathbb{G}$ whose discrete logarithms with respect to g and each other are unknown. The sequence $\{h_j \in \mathbb{G} \mid j = 1, 2, \ldots\}$ can be generated by repeatedly hashing g while ensuring the result falls in the group $\mathbb{G}$. All the exchanges need to agree upon the specific sequence generation procedure. A Nummatus proof which is generated after the jth Quisquis block has appeared and before the $(j+1)$th block has appeared on the blockchain will use the jth element h_j. We will see that this sequence will be used to serve three purposes, namely, (1) to compute the commitment to the total reserves amount of the exchange, (2) to reveal collusion between exchanges sharing account to generate proof of reserves, and (3) to conceal the identity of the exchange's accounts across multiple Nummatus proofs.

Suppose an exchange is generating the Nummatus proof of reserves after the jth Quisquis block. We give a high-level description the procedure followed by the exchange below (a more precise description is given in Section 3.1).

1. Let $\mathcal{A}_{\text{all}}$ be the set of all accounts and let $\mathcal{A}_{\text{own}} \subset \mathcal{A}_{\text{all}}$ be the accounts owned by the exchange.[6] The exchange chooses a set of accounts $\mathcal{A}_{\text{oth}}$ not owned by it, i.e. $\mathcal{A}_{\text{oth}} \subset \mathcal{A}_{\text{all}} \setminus \mathcal{A}_{\text{own}}$. These other accounts are added to the set of exchange-owned accounts to form the *anonymity set* $\mathcal{A}_{\text{anon}} = \mathcal{A}_{\text{own}} \cup \mathcal{A}_{\text{oth}}$.
2. Let $\mathcal{A}_{\text{anon}} = [\texttt{acct}_1, \texttt{acct}_2, \ldots, \texttt{acct}_n]$ where
$$\texttt{acct}_i = (a_i, b_i, c_i, d_i) = \left(a_i, a_i^{k_i}, a_i^{r_i}, g^{v_i} a_i^{k_i \cdot r_i}\right). \tag{10}$$
Here $k_i \in \mathbb{F}_p$ is the secret key, $v_i \in \mathbb{F}_p$ is the account balance, $r_i \in \mathbb{F}_p$ is an arbitrary scalar, and $a_i = g^{t_i}$ for an arbitrary $t_i \in \mathbb{F}_p$.
For each $\texttt{acct}_i$, the exchange creates a Pedersen commitment p_i and a non-interactive zero knowledge proof of knowledge (NIZKPoK) σ_i which proves the *disjunction* of the following statements:
   (i) Account $\texttt{acct}_i$ is owned by the exchange, i.e. the exchange knows the secret key k_i associated with $\texttt{acct}_i$, and p_i is a Pedersen commitment to the balance v_i in $\texttt{acct}_i$.
   (ii) Pedersen commitment p_i is a commitment to the zero amount.
Note that the proof σ_i proves that *one of these two statements* is true without revealing which one.
3. The exchange publishes the anonymity set $\mathcal{A}_{\text{anon}}$, Pedersen commitments $[p_1, p_2, \ldots, p_n]$, and NIZKPoKs $[\sigma_1, \sigma_2, \ldots, \sigma_n]$. It claims that $p_{\text{res}} = \prod_{i=1}^{n} p_i$ is a Pedersen commitment to the total reserves of the exchange.

[6] Sets $\mathcal{A}_{\text{all}}$ and $\mathcal{A}_{\text{own}}$ may change everytime a new block is added to the Quisquis blockchain. Here we consider particular instances of these sets after the jth block is added to the blockchain.

To understand the different parts of the protocol, we need to look at the structure of the individual Pedersen commitments p_i. As discussed above, the discrete logarithms g and h_j are unknown with respect to each other. A Pedersen commitment to an amount $v \in \mathbb{F}_p$ with respect to bases g and h_j is given by $g^v h_j^w$ where $w \in \mathbb{F}_p$ is the blinding factor.

When $\mathtt{acct}_i \in \mathcal{A}_{\text{own}}$, the Nummatus protocol sets $p_i = g^{v_i} h_j^{k_i}$. So p_i is a commitment to the balance v_i of $\mathtt{acct}_i$ and the secret key k_i is the blinding factor in this case. When $\mathtt{acct}_i \notin \mathcal{A}_{\text{own}}$, the Nummatus protocol sets $p_i = h_j^{w_i}$ for a randomly chosen $w_i \in \mathbb{F}_p$, making p_i a commitment to the zero amount. Let $\mathcal{I}_{\text{own}}$ be the indices in $\{1, 2, \ldots, n\}$ corresponding to accounts in $\mathcal{A}_{\text{own}}$, i.e. $\mathtt{acct}_i \in \mathcal{A}_{\text{own}}$ for all $i \in \mathcal{I}_{\text{own}}$. Let $\mathcal{I}_{\text{own}}^c$ denote those indices in $\{1, 2, \ldots, n\}$ which are not in $\mathcal{I}_{\text{own}}$. Then we have

$$p_{\text{res}} = \prod_{i=1}^{n} p_i = \prod_{i \in \mathcal{I}_{\text{own}}} g^{v_i} h_j^{k_i} \prod_{i \in \mathcal{I}_{\text{own}}^c} h_j^{w_i} = g^{v_{\text{res}}} h_j^{w_{\text{res}}}, \tag{11}$$

where

$$v_{\text{res}} = \sum_{i \in \mathcal{I}_{\text{own}}} v_i, \tag{12}$$

$$w_{\text{res}} = \sum_{i \in \mathcal{I}_{\text{own}}} k_i + \sum_{i \in \mathcal{I}_{\text{own}}^c} w_i. \tag{13}$$

Thus p_{res} is a Pedersen commitment to the exchange's total reserves v_{res}. If $p_{\text{liab}} = g^{v_{\text{liab}}} h_j^{w_{\text{liab}}}$ is a Pedersen commitment to the total liabilities of the exchange, then a proof of solvency reduces to showing that

$$p_{\text{res}} p_{\text{liab}}^{-1} = g^{v_{\text{res}} - v_{\text{liab}}} h_j^{w_{\text{res}} - w_{\text{liab}}} \tag{14}$$

is a commitment to a non-negative amount in the correct range.

If two exchanges share an account $\mathtt{acct}_i$ while generating their respective proofs of reserves after the jth Quisquis block, then the account will appear in both the anonymity sets. But this is not enough to prove account sharing collusion between the exchanges. However, the commitment $p_i = g^{v_i} h_j^{k_i}$ corresponding to a shared account will appear in both lists of commitments, revealing the collusion.

The reason for choosing a different base h_j for generating the Pedersen commitments after each block is to prevent leaking the identity of exchange-owned accounts across multiple Nummatus proofs. Suppose the same base h is used to generate commitments in all Nummatus proofs given by an exchange. Then the commitments of exchange-owned accounts will remain same across proofs assuming the balances of the accounts (the v_is) remain same. However, the commitments of unknown accounts will be different in different proofs because of different w_is. Thus an observer will be able identify which accounts belong to the exchange.

A consequence of this design is that exchanges cannot use the same secret key for multiple accounts if they want to use the Nummatus proof of reserves *protocol.* This is not a serious restriction as the convenience afforded by having the same secret key for multiple accounts is negligible compared to the security provided by having different keys for different accounts. This issue is further discussed in Section 4.2.

Note that the proof σ_i proves that p_i is a Pedersen commitment of the form which is either $g^{v_i}h_j^{k_i}$ or $h_j^{w_i}$. If an exchange does not own the account $\texttt{acct}_i$, it will be forced to set p_i to the form $h_j^{w_i}$. When exchange does own the account, it can set p_i to be of the form $g^{v_i}h_j^{k_i}$. In the latter scenario, there is nothing stopping the exchange from setting p_i to be of the form $h_j^{w_i}$. But this will mean that the balance v_i of account $\texttt{acct}_i$ is not counted in the total reserves amount v_{res}. In other words, the exchange is under-reporting the reserves it owns. This is not a problem as long as the reserves exceed the liabilities, since proving solvency is the final goal.

Due to the DDH assumption in the underlying group $\mathbb{G}$, the Nummatus proof of reserves protocol satisfies the following properties:

- *Inflation resistance*: No probabilistic polynomial time (PPT) exchange will be able to generate a commitment to an amount which exceeds the reserves it actually owns.
- *Proof of non-collusion between exchanges*: If two exchanges share an account while generating their respective proofs of reserves (from the same blockchain state), then such collusion can be detected.
- *Privacy of accounts*: No PPT adversary will be able to distinguish whether an account in the anonymity set belongs to the exchange or not.

3.1 Proof Generation

Suppose a Quisquis exchange wants to generate a Nummatus proof corresponding to its reserves after the jth Quisquis block. It performs the following procedure:

1. The exchange chooses the anonymity set of accounts $\mathcal{A}_{\text{anon}}$ from the set of all accounts $\mathcal{A}_{\text{all}}$ present on the blockchain after the jth Quisquis block has appeared and before the $(j+1)$th block appeared. The exchange owns a subset $\mathcal{A}_{\text{own}}$ of $\mathcal{A}_{\text{anon}} = [\texttt{acct}_1, \texttt{acct}_2, \ldots, \texttt{acct}_n]$.
2. For each $\texttt{acct}_i \in \mathcal{A}_{\text{anon}}$ such that $\texttt{acct}_i = \left(a_i, a_i^{k_i}, c_i, g^{v_i}c_i^{k_i}\right)$, the exchange generates a Pedersen commitment p_i of the form
$$p_i = \begin{cases} g^{v_i}h_j^{k_i} & \text{if } \texttt{acct}_i \in \mathcal{A}_{\text{own}}, \\ h_j^{w_i} & \text{if } \texttt{acct}_i \notin \mathcal{A}_{\text{own}}, \end{cases} \tag{15}$$
where the w_is are chosen independently and uniformly from $\mathbb{F}_p$.
3. For each $\texttt{acct}_i \in \mathcal{A}_{\text{anon}}$ given by $\texttt{acct}_i = (a_i, b_i, c_i, d_i) = \left(a_i, a_i^{k_i}, c_i, g^{v_i}c_i^{k_i}\right)$, the exchange generates a NIZKPoK $\sigma_i = (e_{i,1}, e_{i,2}, s_{i,1}, s_{i,2}) \in \mathbb{F}_p^4$ of the form
$$\text{PoK}\left\{(\alpha, \beta) \,\middle|\, \left(b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1}h_j\right)^{\alpha}\right) \vee \left(p_i = h_j^{\beta}\right)\right\}. \tag{16}$$

The NIZKPoK σ_i proves that the exchange knows scalars α, β such that *either* $p_i = h_j^\beta$ *or* $b_i = a_i^\alpha$ and $p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^\alpha$. The algorithm for generating σ_i is given in Appendix A.

4. The exchange publishes the base h_j, the anonymity set $\mathcal{A}_{\text{anon}}$, Pedersen commitments $[p_1, p_2, \ldots, p_n]$, and NIZKPoKs $[\sigma_1, \sigma_2, \ldots, \sigma_n]$. It claims that $p_{\text{res}} = \prod_{i=1}^n p_i$ is a Pedersen commitment to the total reserves of the exchange.

Equation (15) reflects the requirement that p_i is a commitment to the account balance for exchange-owned accounts and a commitment to the zero amount for other accounts. The choice of blinding factor in each case makes p_i a deterministic function of the secret key and the balance for exchange-owned accounts and a random group element for other accounts. The NIZKPoK condition in (16) in fact ensures that an exchange does not deviate from the constructions of p_i given in (15). It states that either p_i is a commitment to the zero amount or the following conditions (in italics) hold:

(i) *The discrete logarithm of b_i with respect a_i is known to the party generating the proof σ_i.*
As $b_i = a_i^{k_i}$, this condition implies that the secret key k_i is known to the exchange.

(ii) *The party generating the proof σ_i knows the discrete logarithm of $p_i d_i^{-1}$ with respect to $c_i^{-1} h_j$. Furthermore, the discrete logarithm is equal to the discrete logarithm of b_i with respect to a_i.*
As $b_i = a_i^{k_i}$, from (16) we have

$$p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{k_i}. \tag{17}$$

Since $d_i = g^{v_i} c_i^{k_i}$, from the above equation we get

$$p_i g^{-v_i} c_i^{-k_i} = c_i^{-k_i} h_j^{k_i} \implies p_i = g^{v_i} h_j^{k_i}. \tag{18}$$

Thus p_i is a commitment to the balance v_i in the account $\texttt{acct}_i$ with blinding factor k_i as given in (15).

3.2 Proof Verification

Given a Nummatus proof of reserves from an exchange referring to the blockchain state after the jth Quisquis block, the verifier checks the following conditions:

1. All the accounts in the anonymity set $\mathcal{A}_{\text{anon}}$ must appear on the blockchain immediately after the jth block. If an account in $\mathcal{A}_{\text{anon}}$ does not appear on the blockchain, the proof is considered invalid.
2. For each i, the NIZKPoK σ_i must pass the verification procedure given in Appendix A.
3. The commitments $p_i, i = 1, 2, \ldots, n$, must not appear in another exchange's Nummatus proof. If the same commitments p_i appears in the Nummatus proofs of two different exchanges, then collusion is declared and the proof of reserves is considered invalid.

4 Nummatus Security Properties

In this section, we discuss the security properties of the Nummatus protocol. We are concerned with inflation resistance, collusion resistance, and account privacy (as defined in Section 3).

4.1 Inflation and Collusion Resistance

Inflation resistance refers to the requirement that a PPT exchange should not be able to use the Nummatus protocol to generate a Pedersen commitment p_{res} to an amount v_{res} which is greater than the total reserves it owns. Suppose p_{res} is a commitment to v_{res} which is greater than $\sum_{i \in \mathcal{I}_{\text{own}}} v_i$. Then since

$$p_{\text{res}} = \prod_{i=1}^{n} p_i = \prod_{i \in \mathcal{I}_{\text{own}}} p_i \prod_{i \in \mathcal{I}_{\text{own}}^c} p_i, \tag{19}$$

it must be that either p_i is not a commitment to zero for some $i \in \mathcal{I}_{\text{own}}^c$ or p_i is a commitment to an amount larger than the account balance v_i for some $i \in \mathcal{I}_{\text{own}}$. For $i \in \mathcal{I}_{\text{own}}^c$, the exchange does not know the secret key k_i and consequently it must generate the NIZKPoK σ_i by setting p_i to be commitment to the zero amount. For $i \in \mathcal{I}_{\text{own}}$, if a PPT exchange sets p_i to be a commitment to a nonzero amount then σ_i forces this amount to be the account balance v_i (see equation (18)). So the inflation resistance property of Nummatus follows from the unforgeability of the NIZKPoKs σ_i.

When p_i is not a commitment to zero, the account $\texttt{acct}_i$ must be owned by the exchange as the private key k_i is needed to create p_i as $g^{v_i} h_j^{k_i}$. This form of p_i is a deterministic function of v_i and k_i. So two exchanges sharing $\texttt{acct}_i$ after the jth Quisquis block will produce same p_i in their proofs. If this happens, then account sharing collusion is immediately detected.

4.2 Account Privacy

Account privacy refers to the requirement that a PPT distinguisher $\mathcal{D}$, which is given a polynomial number of Nummatus proofs as input, cannot identify exchange-owned accounts in the anonymity set $\mathcal{A}_{\text{anon}}$ except with a negligible probability. Our proof that the Nummatus protocol preserves account privacy relies on the DDH problem being hard in the group used to implement Quisquis.

Let $\lambda \in \mathbb{N}$ be a security parameter. Suppose $\texttt{Setup}(1^\lambda)$ is a group generation algorithm which gives $(\mathbb{G}, g, p)$ where $\mathbb{G}$ is a group with generator g and prime order p. We assume that there is no algorithm which can solve the DDH problem in $\mathbb{G}$ with a running time which is polynomial in λ.

To define account privacy security, we will use an experiment with name `AccountPriv`.

– Let us consider an exchange which has generated $f(\lambda)$ Nummatus proofs where f is a polynomial.

– Let $\mathcal{A}_{\text{anon}}^1, \mathcal{A}_{\text{anon}}^2, \ldots, \mathcal{A}_{\text{anon}}^{f(\lambda)}$, be the anonymity sets used in these $f(\lambda)$ proofs having sizes $N_1, N_2, \ldots, N_{f(\lambda)}$, respectively.
– For $l = 1, 2, \ldots, f(\lambda)$, assume that the lth Nummatus proof was created after the j_lth block appeared and before the $(j_l + 1)$th block appeared on the blockchain. So the lth Nummatus proof will use bases g and h_{j_l} to create Pedersen commitments.
– The lth Nummatus proof consists of the base h_{j_l}, the anonymity set $\mathcal{A}_{\text{anon}}^l$, Pedersen commitments $[p_{l,1}, p_{l,2}, \ldots, p_{l,N_l}]$, and NIZKPoKs $[\sigma_{l,1}, \sigma_{l,2}, \ldots, \sigma_{l,N_l}]$.

We make two assumptions:

(i) *The secret keys of all exchange-owned accounts in the lth anonymity set $\mathcal{A}_{anon}^l$ are all distinct.* This is necessary as the Nummatus protocol cannot provide account privacy without this constraint. To see why, suppose two accounts in $\mathcal{A}_{\text{anon}}^l$ share the same secret key k. Let their corresponding Pedersen commitments be $p_{l,i} = g^v h_{j_l}^k$ and $p_{l,i'} = g^{v'} h_{j_l}^k$ where i, i' are the account indices and v, v' are the account balances. An adversary can figure out that these two accounts are exchange-owned accounts by checking if the equality $p_{l,i} g^{-v_1} = p_{l,i'} g^{-v_2}$ holds for some $(v_1, v_2) \in V^2$ where V is the set of all possible amounts. As the size of V is small, this attack is practical. In fact, the receiver of a funds in a regular Quisquis transaction has to search through all possible values in V to figure out the amount received [14, Section 5.2.3].

(ii) *There is an exchange-owned account with secret key k which appears in the anonymity sets of all the $f(\lambda)$ Nummatus proofs.* This assumption serves to simplify the notation. In the `AccountPriv` experiment, we want to consider an adversary which can identify an exchange-owned account based on multiple Nummatus proofs. The existence of such an adversary who can successfully identify an exchange-owned account with a non-negligible probability will lead to a contradiction of the DDH assumption. An adversary who succeeds only if the account appears in all the $f(\lambda)$ Nummatus proofs is weaker than an adversary who can succeed even if the account appears in a subset of the $f(\lambda)$ Nummatus proofs. Thus obtaining a contradiction from the non-negligible success probability of the weaker adversary is sufficient.
Let i_l be the index of the account with secret key k in the lth anonymity set $\mathcal{A}_{\text{anon}}^l = [\texttt{acct}_1^l, \texttt{acct}_2^l, \ldots, \texttt{acct}_{N_l}^l]$. Let v_l be the non-zero balance of this account when the lth Nummatus proof is created. Thus we have

$$p_{l,i_l} = g^{v_l} h_{j_l}^k \text{ or } p_{l,i_l} = h_{j_l}^{w_{l,i_l}}, \tag{20}$$

where $w_{l,i_l} \in \mathbb{F}_p$.

The experiment `AccountPriv` proceeds as follows.

1. The group parameters are generated as $(\mathbb{G}, g, p) \leftarrow \texttt{Setup}(1^\lambda)$. A sequence of group elements $h_1, h_2, h_3, \ldots$ are chosen uniformly and independently from $\mathbb{G}$.

2. From the sequence generated in the previous step, the exchange chooses a subsequence $h_{j_1}, h_{j_2}, \ldots, h_{j_{f(\lambda)}}$ where f is a polynomial.
3. The exchange chooses a bit $\mathfrak{b}$ uniformly from $\{0, 1\}$.
4. If $\mathfrak{b} = 0$, the exchange sets $p_{l,i_l} = h_{j_l}^{w_{l,i_l}}$ for some uniformly chosen w_{l,i_l} from $\mathbb{F}_p$ for all $l = 1, 2, \ldots, f(\lambda)$, i.e. the i_lth account does not contribute to the reserves in all the $f(\lambda)$ Nummatus proofs. The exchange generates the NIZKPoKs σ_{l,i_l} accordingly.
   The other commitments $p_{l,i}$ for all $l = 1, 2, \ldots, f(\lambda)$ and $i = 1, 2, \ldots, N_l, i \neq i_l$ are generated arbitrarily . The corresponding NIZKPoKs $\sigma_{l,i}$ are generated accordingly.
5. If $\mathfrak{b} = 1$, the exchange sets $p_{l,i_l} = g^{v_l} h_{j_l}^{k}$ for all $l = 1, 2, \ldots, f(\lambda)$, i.e. the i_lth account contributes its balance to the reserves in all the $f(\lambda)$ Nummatus proofs. The exchange generates the NIZKPoKs σ_{l,i_l} accordingly.
   The other commitments $p_{l,i}$ for all $l = 1, 2, \ldots, f(\lambda)$ and $i = 1, 2, \ldots, N_l, i \neq i_l$ are generated arbitrarily . The corresponding NIZKPoKs $\sigma_{l,i}$ are generated accordingly.
6. Let $\mathfrak{N}_l = \left(h_{j_l}, \mathcal{A}^l_{\text{anon}}, [p_{l,1}, \ldots, p_{l,N_l}], [\sigma_{l,1}, \ldots, \sigma_{l,N_l}]\right)$ be the lth Nummatus proof. Let the account index vector be $\mathcal{I} = [i_1, i_2, \ldots, i_{f(\lambda)}]$ and the account balance vector be $\mathcal{V} = [v_1, v_2, \ldots, v_{f(\lambda)}]$. The $f(\lambda)$ Nummatus proofs $\{\mathfrak{N}_l\}_{l=1}^{f(\lambda)}$, $\mathcal{I}$, and $\mathcal{V}$ are given as input to a distinguisher $\mathcal{D}$ which then outputs a bit $\mathfrak{b}'$, i.e.
   $$\mathfrak{b}' = \mathcal{D}\left(\mathfrak{N}_1, \mathfrak{N}_2, \ldots, \mathfrak{N}_{f(\lambda)}, \mathcal{I}, \mathcal{V}\right). \tag{21}$$
7. $\mathcal{D}$ succeeds if $\mathfrak{b}' = \mathfrak{b}$. Otherwise it fails.

Definition 1. *The Nummatus proof of reserves protocol provides account privacy if for every PPT distinguisher $\mathcal{D}$ in the* `AccountPriv` *experiment with a probability which is negligibly close to $\frac{1}{2}$.*

This definition captures the requirement that a distinguisher should not be able to tell if an account was used in a sequence of Nummatus proofs even if it knew the account index in the anonymity set and the account balance in all the proofs. Note that distinguisher who knows $\mathcal{I}$ and $\mathcal{V}$ is more likely to succeed than a distinguisher which does not know these vectors, i.e.

$$\Pr\left[\mathcal{D}\left(\mathfrak{N}_1, \mathfrak{N}_2, \ldots, \mathfrak{N}_{f(\lambda)}\right) = \mathfrak{b}\right] \leq \Pr\left[\mathcal{D}\left(\mathfrak{N}_1, \mathfrak{N}_2, \ldots, \mathfrak{N}_{f(\lambda)}, \mathcal{I}, \mathcal{V}\right) = \mathfrak{b}\right]. \tag{22}$$

So if we can show that for every distinguisher $\mathcal{D}$ with knowledge of $\mathcal{I}$ and $\mathcal{V}$, there is a negligible function `negl` such that

$$\Pr\left[\mathcal{D}\left(\mathfrak{N}_1, \mathfrak{N}_2, \ldots, \mathfrak{N}_{f(\lambda)}, \mathcal{I}, \mathcal{V}\right) = \mathfrak{b}\right] \leq \frac{1}{2} + \texttt{negl}(\lambda), \tag{23}$$

then the same upper bound applies holds on the success probability of distinguishers which do not know $\mathcal{I}$ and $\mathcal{V}$. We will use a distinguisher of the form $\mathcal{D}\left(\mathfrak{N}_1, \mathfrak{N}_2, \ldots, \mathfrak{N}_{f(\lambda)}, \mathcal{I}, \mathcal{V}\right)$ to construct an adversary who can solve the generalized DDH problem [6] resulting in the following theorem.

Theorem 1. *The Nummatus proof of reserves protocol provides account privacy in the random oracle model under the DDH assumption provided that the exchange uses distinct secret keys for its accounts in the anonymity set.*

Proof. Suppose $\mathcal{E}$ is an adversary which wants to solve the generalized DDH problem given a tuple $\left(g_1, \ldots, g_{f(\lambda)}, u_1, \ldots, u_{f(\lambda)}\right) \in \mathbb{G}^{2f(\lambda)}$ [6]. Specifically, it want to distinguish between the following two situations:

- In the tuple $\left(g_1, \ldots, g_{f(\lambda)}, u_1, \ldots, u_{f(\lambda)}\right)$, each of the g_is and u_is are uniformly and independently chosen from $\mathbb{G}$.
- In the tuple $\left(g_1, \ldots, g_{f(\lambda)}, u_1, \ldots, u_{f(\lambda)}\right)$, each of the g_is are uniformly and independently chosen from $\mathbb{G}$. Each $u_l = g_l^k$ for all $l = 1, 2, \ldots, f(\lambda)$ where k is chosen uniformly from $\mathbb{F}_p$.

Let $\mathfrak{d} = 0$ denote the former situation and $\mathfrak{d} = 1$ denote the latter situation. If $\mathfrak{d}'$ is the output of a PPT $\mathcal{E}$, then $\Pr\left[\mathfrak{d}' = \mathfrak{d}\right]$ must be negligibly close to $\frac{1}{2}$.

The adversary $\mathcal{E}$ will construct a valid input for the distinguisher $\mathcal{D}$ in the `AccountPriv` experiment in the following manner:

1. For $l = 1, 2, \ldots, f(\lambda)$, $\mathcal{E}$ chooses anonymity set sizes N_l and test account indices i_l where $1 \leq i_l \leq N_l$. It also chooses non-zero values v_l for the test account balances from the allowed set of amount values V.
2. For each l, the accounts in the anonymity set $\mathcal{A}^l_{\text{anon}} = [\texttt{acct}^l_1, \ldots, \texttt{acct}^l_{N_l}]$ are constructed as
$$\texttt{acct}^l_i = (a_{l,i}, b_{l,i}, c_{l,i}, d_{l,i}) = \begin{cases} \left(g^{t_{l,i}}, g^{k_{l,i} \cdot t_{l,i}}, g^{t_{l,i} \cdot r_{l,i}}, g^{v_{l,i} + k_{l,i} \cdot t_{l,i} \cdot r_{l,i}}\right) & \text{if } i \neq i_l \\ \left(g_l^{t_{l,i}}, u_l^{k' \cdot t_{l,i}}, g_l^{t_{l,i} \cdot r_{l,i}}, g^{v_l} u_l^{k' \cdot t_{l,i} \cdot r_{l,i}}\right) & \text{if } i = i_l \end{cases}$$
where $k', k_{l,i}, t_{l,i}, r_{l,i}$ are chosen uniformly and independently from $\mathbb{F}_p$ and $v_{l,i}$ are chosen uniformly and independently from V. From equation (1), it follows that $\texttt{acct}^l_i$ is a valid account for $i \neq i_l$. For $i = i_l$ and $u_l = g_l^k$, i.e. the case of $\mathfrak{d} = 1$, it again follows that $\texttt{acct}^l_l$ is a valid account with secret key $k \cdot k'$. For $\mathfrak{d} = 0$, the u_ls are independent of the g_ls. But since the g_ls are generators of the prime order group $\mathbb{G}$, we have $u_l = g_l^{s_l}$ for some $s_l \in \mathbb{F}_p$. Even though the s_ls are not known to $\mathcal{E}$, $\texttt{acct}^l_i$ can be expressed as $\left(g_l^{t_{l,i}}, g_l^{k' \cdot s_l \cdot t_{l,i}}, g_l^{t_{l,i} \cdot r_{l,i}}, g^{v_l} g_l^{k' \cdot s_l \cdot t_{l,i} \cdot r_{l,i}}\right)$ which is a valid account structure.
3. For each l, $\mathcal{E}$ sets $h_{j_l} = g_l$ and $p_{l,i_l} = g^{v_l} u_l^{k'}$. As $\mathcal{E}$ does not know the discrete logarithm of $u_l^{k'}$ with respect to $h_{j_l} = g_l$, it generates valid NIZKPoKs σ_{l,i_l} using the random oracle assumption on H (see [12] for a similar argument). It involves changing the outputs of H for some inputs such that equation (31) is satisfied. We omit the details due to space constraints.
The other commitments $p_{l,i}$ for all $l = 1, 2, \ldots, f(\lambda)$ and $i = 1, 2, \ldots, N_l, i \neq i_l$ are generated arbitrarily . The corresponding NIZKPoKs $\sigma_{l,i}$ are generated accordingly using knowledge of $k_{l,i}$.
4. $\mathcal{E}$ gives the $f(\lambda)$ Nummatus proofs generated in the previous steps, the index vector $\mathcal{I}$, and the balance vector $\mathcal{V}$ to a distinguisher $\mathcal{D}$ in the `AccountPriv` experiment. Let $\mathfrak{b}' = \mathcal{D}\left(\mathfrak{N}_1, \mathfrak{N}_2, \ldots, \mathfrak{N}_{f(\lambda)}, \mathcal{I}, \mathcal{V}\right)$. $\mathcal{E}$ outputs $\mathfrak{d}' = \mathfrak{b}'$. If $\mathcal{D}$ is a PPT algorithm, then so is $\mathcal{E}$.

$\mathcal{A}_{\text{anon}}$ size	$\mathcal{A}_{\text{own}}$ size	Nummatus Proof Size	Nummatus Generat. Time	Nummatus Verification Time	Simplus Proof Size	Simplus Generat. Time	Simplus Verification Time
100	25	0.02 MB	1.15 s	1.15 s	0.006 MB	0.29 s	0.28 s
100	50	0.02 MB	1.16 s	1.16 s	0.011 MB	0.58 s	0.57 s
100	75	0.02 MB	1.19 s	1.19 s	0.017 MB	0.91 s	0.91 s
1000	250	0.29 MB	11.94 s	11.76 s	0.057 MB	3.00 s	2.98 s
1000	500	0.29 MB	11.92 s	11.77 s	0.114 MB	5.97 s	5.95 s
1000	750	0.29 MB	11.83 s	11.74 s	0.171 MB	8.92 s	8.74 s
10000	2500	2.93 MB	112.65 s	113.36 s	0.572 MB	28.99 s	28.06 s
10000	5000	2.93 MB	112.08 s	113.23 s	1.145 MB	56.40 s	56.63 s
10000	7500	2.93 MB	111.71 s	112.87 s	1.717 MB	85.07 s	85.72 s

Table 1. Proof Generation and Verification Performance of Nummatus and Simplus

Suppose the $\mathfrak{d} = 0$ situation occurs, i.e. $u_l = g_l^{s_l}$ for uniform $s_l \in \mathbb{F}_p$. Then irrespective of the values of v_l, the terms $p_{l,i_l} = g^{v_l} u_l^{k'}$ are uniformly distributed over $\mathbb{G}$. This corresponds to the situation of $\mathfrak{b} = 0$ in the `AccountPriv` experiment. On the other hand if the $\mathfrak{d} = 1$ situation occurs, then $u_l = g_l^k$ for a fixed $k \in \mathbb{F}_p$ for all l. Then for $h_{j_l} = g_l$, we have $p_{l,i} = g^{v_l} u_l^{k'} = g^{v_l} g_l^{k \cdot k'} = g^{v_l} h_{j_l}^{k \cdot k'}$. This corresponds to the situation of $\mathfrak{b} = 1$ in the `AccountPriv` experiment. Thus we have $\Pr[\mathfrak{d}' = \mathfrak{d}] = \Pr[\mathfrak{b}' = \mathfrak{b}]$.

If there exists a PPT distinguisher $\mathcal{D}$ whose success probability $\Pr[\mathfrak{b}' = \mathfrak{b}]$ is larger than $\frac{1}{2} + \frac{1}{q(\lambda)}$ for some polynomial q, then this will imply that the success probability $\Pr[\mathfrak{d}' = \mathfrak{d}]$ of $\mathcal{E}$ is also larger than $\frac{1}{2} + \frac{1}{q(\lambda)}$. As a PPT adversary who can solve the generalized DDH problem is equivalent to a PPT adversary who can solve classical DDH problem [6], we get a contradiction. Thus any PPT distinguisher $\mathcal{D}$ in the `AccountPriv` experiment can only succeed with a probability which is negligibly close to $\frac{1}{2}$. $\square$

5 Performance

To the best of our knowledge, Nummatus is the first proof of reserves protocol for Quisquis exchanges which keeps the identities of the exchange accounts private. For benchmarking purposes, we compare Nummatus to a simple non-private protocol which we will call *Simplus*.[7] In the Simplus protocol, the exchange reveals the accounts it owns, i.e. the set $\mathcal{A}_{\text{own}}$ is revealed. Like Nummatus, this protocol outputs a Pedersen commitment to the total reserves of the exchange. While the Simplus protocol does not provide account privacy, it provides reserve amount privacy. The proof generation in Simplus proceeds as follows:

1. The exchange chooses a set $\mathcal{A}_{\text{own}} = \{\texttt{acct}_1, \texttt{acct}_2, \ldots, \texttt{acct}_m\}$ of its own accounts which are sufficient to meet its liabilities. These accounts need to

[7] Simplus is Latin for “simple” [19].

be present on the blockchain after the jth Quisquis block has appeared and before the $(j+1)$th block appeared.

2. For each $\texttt{acct}_i \in \mathcal{A}_{\text{own}}$ given by $\texttt{acct}_i = (a_i, b_i, c_i, d_i) = \left(a_i, a_i^{k_i}, c_i, g^{v_i} c_i^{k_i}\right)$, the exchange generates a Pedersen commitment $p_i := g^{v_i} h^{k_i}$ and a NIZKPoK $\psi_i = (e_{i,1}, s_{i,1}) \in \mathbb{F}_p^2$ of the form
$$\text{PoK}\left\{\alpha \;\middle|\; \left(b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h\right)^{\alpha}\right)\right\}. \tag{24}$$
Note that in case of Nummatus, we need to use h_j as base of p_i to make exchange's accounts indistinguishable from accounts not owned by the exchange across multiple Nummatus proofs. But in case of Simplus, the exchange has already revealed the accounts owned by it. Therefore we can simply use h instead of h_j as a base of p_i. The algorithm for generating ψ_i is given in Appendix B.
3. The exchange publishes the set $\mathcal{A}_{\text{own}}$, Pedersen commitments $[p_1, p_2, \ldots, p_m]$, and NIZKPoKs $[\psi_1, \psi_2, \ldots, \psi_m]$. It claims that $p_{\text{res}} = \prod_{i=1}^{m} p_i$ is a Pedersen commitment to the total reserves of the exchange.

The NIZKPoK in (24) ensures that the exchange knows the private key k_i for each account $\texttt{acct}_i$. Furthermore, by the argument presented in the Nummatus protocol discussion, the NIZKPoK ensures that p_i is a commitment to the account balance v_i of $\texttt{acct}_i$. As the exchange's accounts are revealed in the Simplus protocol, collusion between exchanges can be detected if the same account appears in the own account lists of two different exchanges.

The Simplus proof verification proceeds as follows:

1. All the accounts in the set $\mathcal{A}_{\text{own}}$ must appear on the blockchain immediately after the jth block. If not, the proof is considered invalid.
2. For each i, the NIZKPoK ψ_i must pass the verification procedure given in Appendix B.

The simulation code was implemented in Rust using the rust-secp256k1-zkp library [1] which has also been used for Revelio [13]. The performance of the Nummatus proof generation and verification algorithms is given in Table 1 for anonymity list $\mathcal{A}_{\text{anon}}$ having sizes 100, 1000, and 10000. For each case, the percentage of accounts belonging to the exchange is either 25%, 50%, or 75%. Table 1 also shows the performance of the Simplus protocol as a function of $\mathcal{A}_{\text{own}}$ size (the $\mathcal{A}_{\text{anon}}$ parameter is irrelevant here). The execution times were measured on single core of an Intel i7-7700 3.6 GHz CPU. The Nummatus protocol is at most 3 to 4 times slower and its proof size is at most 4 to 5 times larger compared to the Simplus protocol. The proof size and execution time of Nummatus protocol are practical and can be reduced further by parallel signature generations and verifications for different accounts in $\mathcal{A}_{\text{anon}}$. The higher values of performance parameters for Nummatus than that of Simplus can be considered as the price we are paying for privacy.

6 Conclusion

We give Nummatus, the first privacy preserving proof of reserves protocol for Quisquis [14] exchanges. Using Nummatus, a Quisquis exchange can prove that it holds more reserves than what it owes to its customers without revealing the reserves amount or the identity of owned accounts. Nummatus also detects the account sharing collusion between exchanges provided all exchanges generate their proofs after a particular block is added to the Quisquis blockchain. We give the performance comparison of Nummatus and a non-private proof of reserves protocol which we call Simplus. Our simulation shows that deployment of Nummatus is practical and feasible.

A Nummatus NIZKPoK Generation and Verification Algorithms

In this appendix, we present algorithms for generating and verifying the NIZKPoK σ_i that is used in Nummatus. In the notation proposed by Camenisch and Stadler [7], [8], the NIZKPoK is of the form

$$\text{PoK}\left\{(\alpha,\beta) \,\middle|\, \left(b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{\alpha}\right) \vee \left(p_i = h_j^{\beta}\right)\right\}.$$

The above proof is for a disjunction of two statements. We motivate the structure of σ_i by first describing methods to prove these two statements individually. Then the method first proposed by Cramer *et al.* [9] is used to generate a proof for the disjunction of the two statements.

Let $H : \{0,1\}^* \mapsto \mathbb{F}_p$ be a cryptographic hash function which is modeled as a random oracle. Let $\|$ denote the bitstring concatenation operator. For notational convenience, we write $H(x, y, z)$ to denote $H(x\|y\|z)$ where x, y, z are group elements represented as bitstrings.

Definition 2. *An ordered pair $(e, s) \in \mathbb{F}_p^2$ is a NIZKPoK of the discrete logarithm of a group element p_i with respect to a base h_j if*

$$e = H(h_j, p_i, h_j^s p_i^e). \tag{25}$$

The pair (e, s) is said to be of the form PoK$\{\beta \mid p_i = h_j^{\beta}\}$.

The proof (e, s) is generated by first choosing a scalar r uniformly from $\mathbb{F}_p$ and calculating $e = H(h_j, p_i, h_j^r)$. The second element of the pair is calculated as $s = r - e\beta$ where β is the discrete logarithm of p_i with respect to h_j, which is known to the prover. It now follows that

$$e = H(h_j, p_i, h_j^r) = H(h_j, p_i, h_j^{s+e\beta}) = H(h_j, p_i, h_j^s p_i^e). \tag{26}$$

The verification of the proof (e, s) simply consists of checking the equality in equation (25).

Definition 3. *An ordered pair $(e, s) \in \mathbb{F}_p^2$ is a NIZKPoK of*

(i) the knowledge of the discrete logarithms of the group elements b_i with respect to base a_i, ***and***
(ii) the knowledge of discrete logarithms of the group element $p_i d_i^{-1}$ with respect to base $c_i^{-1} h_j$, ***and***
(iii) the equality of the discrete logarithm of b_i with respect to a_i and of $p_i d_i^{-1}$ with respect to $c_i^{-1} h_j$,

if it satisfies

$$e = H\left(\mathtt{stmt}_i, a_i^s b_i^e, \left(c_i^{-1} h_j\right)^s \left(p_i d_i^{-1}\right)^e\right). \tag{27}$$

where $\mathtt{stmt}_i = (h_j, a_i, b_i, c_i, d_i, p_i)$ is the tuple of group elements appearing in the statement being proved. The ordered pair (e, s) is said to be of the form

$$PoK\left\{\alpha \;\middle|\; b_i = a_i^\alpha \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^\alpha\right\}.$$

A prover who knows α can generate the proof (e, s) as follows:

– The prover chooses scalars r uniformly from $\mathbb{F}_p$ and calculates

$$e = H\left(\mathtt{stmt}_i, a_i^r, \left(c_i^{-1} h_j\right)^r\right). \tag{28}$$

– The second scalar in the proof is calculated as

$$s = r - e\alpha \tag{29}$$

It follows that

$$\begin{aligned} e &= H\left(\mathtt{stmt}_i, a_i^r, \left(c_i^{-1} h_j\right)^r\right) \\ &= H\left(\mathtt{stmt}_i, a_i^{s+e\alpha}, \left(c_i^{-1} h_j\right)^{s+e\alpha}\right) \\ &= H\left(\mathtt{stmt}_i, a_i^s b_i^e, \left(c_i^{-1} h_j\right)^s \left(p_i d_i^{-1}\right)^e\right). \end{aligned} \tag{30}$$

The verification of the proof (e, s) simply consists of checking the equality in equation (27).

The NIZKPoK σ_i in Nummatus is a proof of the disjunction of the two statements proved above.

Definition 4. *The tuple $\sigma_i = (e_1, e_2, s_1, s_2) \in \mathbb{F}_p^4$ is a NIZKPoK of the knowledge of the discrete logarithm of a group element p_i with respect to base h_j* ***or***

– *the knowledge of the discrete logarithm of the group element b_i with respect to base a_i,* ***and***
– *the knowledge of discrete logarithm of the group element $p_i d_i^{-1}$ with respect to base $c_i^{-1} h_j$,* ***and***
– *the equality of the discrete logarithm of b_i with respect to a_i and of $p_i d_i^{-1}$ with respect to $c_i^{-1} h_j$,*

if it satisfies

$$e_1 + e_2 = H\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2} p_i^{e_2}\right). \tag{31}$$

where $\mathtt{stmt}_i = (h_j, a_i, b_i, c_i, d_i, p_i)$ *is the tuple of group elements appearing in the statement being proved. The tuple* (e_1, e_2, s_1, s_2) *is said to be of the form*

$$PoK\left\{(\alpha, \beta) \,\middle|\, \left(b_i = a_i^{\alpha} \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{\alpha}\right) \vee \left(p_i = h_j^{\beta}\right)\right\}.$$

Suppose the prover know the discrete logarithm β of p_i with respect to base h_j. Then she can create the proof σ_i as follows:

(i) She chooses scalars r_2, e_1, s_1 uniformly and independently from $\mathbb{F}_p$. She calculates e_2 as

$$e_2 = H\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{r_2}\right) - e_1. \tag{32}$$

(ii) Using her knowledge of β, she calculates s_2 as

$$s_2 = r_2 - e_2 \beta. \tag{33}$$

It follows that

$$\begin{aligned} e_1 + e_2 &= H\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2 + e_2 \beta}\right) \\ &= H\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2} p_i^{e_2}\right). \end{aligned} \tag{34}$$

On the other hand, if the prover knows α such that $b_i = a_i^{\alpha}$, and $p_i d_i^{-1} = \left(c_i^{-1} h_j\right)^{\alpha}$, then she can create the proof σ_i as follows:

(i) She chooses scalars r_1, e_2, s_2 uniformly and independently from $\mathbb{F}_p$. She calculates e_1 as

$$e_1 = H\left(\mathtt{stmt}_i, a_i^{r_1}, \left(c_i^{-1} h_j\right)^{r_1}, h_j^{s_2} p_i^{e_2}\right) - e_2. \tag{35}$$

(ii) Using her knowledge of α, she calculates s_1 as

$$s_1 = r_1 - e_1 \alpha. \tag{36}$$

It follows that

$$\begin{aligned} e_1 + e_2 &= H\left(\mathtt{stmt}_i, a_i^{r_1}, \left(c_i^{-1} h_j\right)^{r_1}, h_j^{s_2} p_i^{e_2}\right) \\ &= H\left(\mathtt{stmt}_i, a_i^{s_1 + e_1 \alpha}, \left(c_i^{-1} h_j\right)^{s_1 + e_1 \alpha}, h_j^{s_2} p_i^{e_2}\right) \\ &= H\left(\mathtt{stmt}_i, a_i^{s_1} b_i^{e_1}, \left(c_i^{-1} h_j\right)^{s_1} \left(p_i d_i^{-1}\right)^{e_1}, h_j^{s_2} p_i^{e_2}\right). \end{aligned} \tag{37}$$

In both cases, the verification of the proof (e_1, e_2, s_1, s_2) consists of checking the equality in equation (31). In the proof of disjunction of statements, the prover has one degree of freedom as only the sum $e_1 + e_2$ has to be equal to the hash function output (whose argument contains the scalars). This freedom is exploited to choose which knowledge is used to prove the disjunction.

B Simplus NIZKPoK Generation and Verification Algorithms

Compared to the NIZKPoK in the Nummatus protocol, the NIZKPoK ψ_i in the Simplus protocol is simpler to compute.

Definition 5. *An ordered pair $\psi_i = (e, s) \in \mathbb{F}_p^2$ is a NIZKPoK of*

(i) the knowledge of the discrete logarithm of the group element b_i with respect to base a_i, ***and***
(ii) the knowledge of discrete logarithm of the group element $p_i d_i^{-1}$ with respect to base $c_i^{-1}h$, ***and***
(iii) the equality of the discrete logarithm of b_i with respect to a_i and of $p_i d_i^{-1}$ with respect to $c_i^{-1}h$,

if it satisfies

$$e = H\left(\textit{stmt}_i, a_i^s b_i^e, \left(c_i^{-1}h\right)^s \left(p_i d_i^{-1}\right)^e\right). \tag{38}$$

where $\textit{stmt}_i = (h, a_i, b_i, c_i, d_i, p_i)$ is the tuple of group elements appearing in the statement being proved. The ordered pair (e, s) is said to be of the form

$$PoK\left\{\alpha \;\middle|\; b_i = a_i^\alpha \;\wedge\; p_i d_i^{-1} = \left(c_i^{-1}h\right)^\alpha\right\}.$$

A prover who knows α can generate the proof (e, s) as follows:

– The prover chooses scalars r uniformly from $\mathbb{F}_p$ and calculates

$$e = H\left(\mathtt{stmt}_i, a_i^r, \left(c_i^{-1}h\right)^r\right). \tag{39}$$

– The second scalar in the proof is calculated as

$$s = r - e\alpha \tag{40}$$

It follows that

$$\begin{aligned} e &= H\left(\mathtt{stmt}_i, a_i^r, \left(c_i^{-1}h\right)^r\right) \\ &= H\left(\mathtt{stmt}_i, a_i^{s+e\alpha}, \left(c_i^{-1}h\right)^{s+e\alpha}\right) \\ &= H\left(\mathtt{stmt}_i, a_i^s b_i^e, \left(c_i^{-1}h\right)^s \left(p_i d_i^{-1}\right)^e\right). \end{aligned} \tag{41}$$

The verification of the proof (e, s) simply consists of checking the equality in equation (38).

References

1. Grin rust-secp256k1-zkp github repository, `https://github.com/mimblewimble/secp256k1-zkp/`
2. Monero 0.13.0 Beryllium Bullet Release Notes, `https://src.getmonero.org/2018/10/11/monero-0.13.0-released.html`, [Accessed 02-Aug-2019]
3. Monero website, `https://getmonero.org/`
4. What are zk-SNARKs?, `https://z.cash/technology/zksnarks/`, [Accessed 02-Aug-2019]
5. Zcash website, `https://z.cash/`
6. Bao, F., Deng, R.H., Zhu, H.: Variations of Diffie-Hellman problem. In: Information and Communications Security. pp. 301–312 (2003)
7. Camenisch, J.: Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. dissertation, ETH Zürich (1998)
8. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Tech. rep. (1997)
9. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Advances in Cryptology — CRYPTO '94. pp. 174–187. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
10. Dagher, G.G., Bünz, B., Bonneau, J., Clark, J., Boneh, D.: Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (ACM CCS). pp. 720–731. New York, NY, USA (2015)
11. Decker, C., Guthrie, J., Seidel, J., Wattenhofer, R.: Making Bitcoin exchanges transparent. In: 20th European Symposium on Research in Computer Security (ESORICS). pp. 561–576 (2015)
12. Dutta, A., Vijayakumaran, S.: MProve: A proof of reserves protocol for Monero exchanges. In: 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 330–339 (June 2019). https://doi.org/10.1109/EuroSPW.2019.00043
13. Dutta, A., Vijayakumaran, S.: Revelio: A MimbleWimble proof of reserves protocol. In: 2019 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 7–11 (June 2019). https://doi.org/10.1109/CVCBT.2019.000-5
14. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A new design for anonymous cryptocurrencies. Cryptology ePrint Archive, Report 2018/990 (2018), `https://eprint.iacr.org/2018/990`
15. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Advances in Cryptology — CRYPTO '91. pp. 129–140. Springer (1992)
16. Saberhagen, N.v.: CryptoNote v 2.0. White paper (2013), `https://cryptonote.org/whitepaper.pdf`
17. Wiktionary contributors: nummatus — Wiktionary, the free dictionary, `https://en.wiktionary.org/wiki/nummatus`, [Accessed 02-Aug-2019]
18. Wiktionary contributors: quisquis — Wiktionary, the free dictionary, `https://en.wiktionary.org/wiki/quisquis`, [Accessed 02-Aug-2019]
19. Wiktionary contributors: simplus — Wiktionary, the free dictionary, `https://en.wiktionary.org/wiki/simplus`, [Accessed 02-Aug-2019]
20. Wilcox, Z.: Proving your Bitcoin reserves. Bitcoin Talk Forum Post (May 2014), `https://bitcointalk.org/index.php?topic=595180.0`