Privacy-Preserving, Taxable Bank Accounts

Elli Androulaki, Binh Vo and Steven Bellovin
{elli, binh, smb}@cs.columbia.edu

Columbia University

Abstract. Current banking systems do not aim to protect user privacy. Purchases made from a single bank account can be linked to each other by many parties. This could be addressed in a straightforward way by generating unlinkable credentials from a single master credential using Camenisch and Lysyanskaya's algorithm; however, if bank accounts are taxable, some report must be made to the tax authority about each account. Using unlinkable credentials, digital cash, and zero knowledge proofs of knowledge, we present a solution that prevents anyone, even the tax authority, from knowing which accounts belong to which users, or from being able to link any account to another or to purchases or deposits.

1 Introduction

One of the hardest realms in which to achieve privacy is finance. Apart from the obvious — few transactions are made via cash or other anonymous payment mechanisms — society often requires that other information about bank accounts be disclosed. In the U.S., for example, banks and other financial institutions are required to report interest or dividend payments, since they are generally considered to be taxable income. Some jurisdictions require that a portion of the interest be paid directly to the government; other jurisdictions impose taxes on actual balances. These requirements conflict with a desire for privacy and suggesting a way to combine the two is the topic of this paper.
Pseudonymity as Privacy Mechanism. One particular aspect of the conflict concerns a very common technique for achieving transactional privacy: pseudonymity. In pseudonymous systems, an individual has a multitude of separate, unlinkable identities that can be used as desired. A separate pseudonym can be used for each peer, thus preventing linkage between different sorts of activities.

We claim that pseudonymity may be adopted in the banking system to achieve privacy, as, at least for tax purposes, neither banks nor the government need to know who owns a particular bank account. In fact, there are both security and privacy benefits to having multiple pseudonymous accounts. Often, knowledge of a "routing number" (effectively, the bank's identity) and an account number are sufficient to *withdraw* money from an account as well as deposit money into it. Having multiple pseudonymous accounts — and closing those created for a special purpose when they are no longer needed — could prevent such incidents.
The Challenge. Although the identity of the account owners is not a functional requirement of the banking system, pseudonymity may harden authorization and encourage fairness attacks as it lacks accountability. Banks need to know that only authorized parties withdraw money from accounts; governments need ensure that balances and income are properly reported, and taxes paid. An ideal system would be one where an individual could open a bank account without disclosing his or her real identity; nevertheless the relevant tax authorities would receive accurate reports of relevant information.
Our Contribution. We present a solution that accomplishes these goals. Individuals need present strong identification credentials only to obtain a single membership to the bank, after which he

may open an arbitrary number of anonymous or nominal accounts, without anyone being able to link those accounts to their owner or one to the other. Periodically, appropriate information on taxable items is supplied; the tax authority can verify that all accounts are properly reported. Our protocols are secure under the strong RSA assumption.

Organization. In Section 2, we give a more precise statement of the architecture and requirements for the system. The protocols are described in Section 3. We explain why we believe this system to be secure in Section 4. Section 5 discusses related work; Section 6 has concluding thoughts.

2 System Architecture

In this section, we present an overview of our system requirements, threat model and other underlying assumptions for our protocols. In addition, we enumerate the types operations considered in the system, followed by the security definition. We note that some of these definitions were inspired by previous work on other primitives, such as [CL01,CHL05].

System Entities It constituting a complete part of our identity management system, the entities involved are exactly the ones mentioned in chapter **??** and which we mention here for convenience:

- *Users*, who open bank accounts and must pay taxes.
- *Banks*, who allow users to open accounts for the purpose of storing cash and handling financial transactions. They are responsible for reporting interest for income tax purposes. In accordance to the two types of accounts they support, each bank B maintains two databases:
  1. the $\mathsf{D}_{\mathsf{reg}}$, which contains the contact and credential information of its clients, as well as the nominal accounts' history information, and
  2. the D_{α}, which contains all the anonymous accounts' information, i.e., authorization information, account balance, etc.
- *Tax Authority* (TA), which is responsible for ensuring that correct income taxes are paid by all users. Tax Authority corresponds to the U.S.'s Internal Revenue Service (IRS), the Canada Revenue Agency, the U.K.'s HM Revenue & Customs, etc.

We also assume that each each individual owns an IDC (see chapter **??** for more details), which is ultimately bound to a specific person. Banks use this ID card to gain strong assurance of the identity of the person who registers to them.

Operations To define the operations supported by a reputation system strictly we will use the following notation: when an operation is an interactive procedure (or a protocol consisting of multiple procedures) between two entities A and B, we denote it by $\langle O_A, O_B \rangle \leftarrow \mathsf{Pro}(I_C)[A(I_A), B(I_B)]$, where Pro is the name of the procedure (or protocol). O_A (resp. O_B) is the private output of A (resp. B), I_C is the common input of both entities, and I_A (resp. I_B) is the private input of A (resp. B). We also note that depending on the setup, some operations may require additional global parameters (e.g., some common parameters for efficient zero-knowledge proofs, a modulus p, etc). Our system will need these additional parameters only when using underlying schemes that use such parameters, e.g., e-cash systems or anonymous credential systems. To simplify notation, we omit these potential global parameters from the inputs to all the operations.

- $(pk_{\mathsf{B}}, sk_{\mathsf{B}}) \leftarrow \mathsf{Bkeygen}(1^k)$ is the key generation algorithm for Bank.

- $(\mathsf{pk_U}, \mathsf{sk_U}) \leftarrow \mathsf{Ukeygen}(1^k)$ is the key generation algorithm for the users. We call $\mathsf{pk_U}$ the (master) public key of U, and $\mathsf{sk_U}$ the master secret key of U.
- $(\mathsf{P}, sec_\mathsf{P}) \leftarrow \mathsf{Pnymgen}(1^k)$ is the pseudonym generation algorithm for users. The sec_P is the secret information used to generate the pseudonym P.
- $\langle sec_\alpha, \mathsf{D}_\alpha{}' \rangle / \langle \perp, \perp \rangle \leftarrow \mathsf{AccountOpen}\ (pk_\mathsf{B})\ [\mathsf{U}(\mathsf{sk_U}), \mathsf{B}(sk_\mathsf{B}, \mathsf{D}_\alpha, \mathsf{D_{reg}})]$. A user U requests and open an anonymous bank account from bank B. Bank, using $\mathsf{D_{reg}}$, checks if U is eligible for it and If so, the account is opened. Both $\mathsf{D}_\alpha$and $\mathsf{D_{reg}}$ are changed to reflect the updated user account information.
- $\langle T^i, \mathsf{D}_\alpha{}' \rangle / \langle \perp, \perp \rangle \leftarrow \mathsf{TaxReportIssue}\ (\mathsf{P}^\mathsf{i}, pk_\mathsf{B})\ [\mathsf{U}(sec^i_\alpha, \mathsf{sk_U}), \mathsf{B}(sk_\mathsf{B})]$. A user U owning account α_i (secret information sec^i_α) and B collaborate for the former to issue a tax report for α_i. U demonstrates knowledge of the account ownership and contributes his secret information to issue tax report T^i. B notes in D_α that a tax report for α_i has been issued.
- $TT^i \ / \ \perp \leftarrow \mathsf{TaxReportTransform}(T^i, \mathsf{sk_U})$. A user U having issued a tax report T^i, transforms it in the unlinkable form TT^i, for which he can still demonstrate ownership of.
- $\langle \top, \mathsf{D}_\alpha{}', \mathsf{D_{reg}}{}' \rangle / \langle \perp, \perp \rangle \leftarrow \mathsf{TaxReportDeposit}\ (pk_\mathsf{B}, \mathsf{pk_U}, TT^1, \ldots, TT^N)\ [\mathsf{U}(\mathsf{sk_U}), \mathsf{B}(sk_\mathsf{B}, \mathsf{D_{reg}})]$. User U who has already issued the tax reports $T^1, \ldots, T^N$ and transformed them to $TT^1, \ldots, TT^N$ respectively, deposits the latters to the bank B.
- $TotalTax/\perp \leftarrow \mathsf{TotalTaxCalculation}\ [\mathsf{U}(\mathsf{sk_U}, T^1, \ldots, T^N), \mathsf{B}(sk_\mathsf{B}, \mathsf{D_{reg}})]$. The user U and the bank B collaborate for the latter to calculate the overall tax amount withheld by U's accounts. The secret input of the user is his secret information and the initial tax reports $T^1, \ldots, T^N$.

Threat Model We make the following assumptions:

- *Users may try to cheat.* A user trying to avoid paying taxes may attempt to lie regarding the tax he has been withheld and is motivated enough to attempt any type of forgery. We also assume that malicious users may collaborate in order to change their reported balance to their benefit, as long as through this collaboration they do not endanger their funds.

- *Banks are "honest but curious".* Aiming to maintain their clientele, banks are trusted to perform all their functional operations correctly, i.e., they issue credentials, open and update accounts as instructed by their customers. However, they may use the information they possess for other reasons, i.e., to sell credit card based profiles to advertising companies, while they may collaborate with tax authority to reveal the identity behind an anonymous account.

- *Tax Authority is considered to be "honest"*, as we assume that it is operated by the government who wants to protect honest users. However, they are not assumed to protect privacy; indeed, there have been a number of incidents in the U.S. of privacy violations by tax authorities or by unscrupulous individuals employed by the tax authorities.

Requirements *Correctness*, user *privacy* and *security* of the operations of the system are our core requirements. We will strictly define each of them.

Correctness requires that if an honest user U, who is eligible for opening anonymous accounts with an honest bank B, runs $\mathsf{AccountOpen}$ with B, then none will output an error message. Also, if honest U, has opened accounts $\alpha_1, \ldots, \alpha_N$ with honest B, and runs $\mathsf{TaxReportIssue}$, then no one will output an error message, while when the user tries to deposit them and thus runs with TA

TaxReportDeposit and TotalTaxCalculation no entity will output error message and they will output the aggregated tax withheld by honest U's accounts.

Privacy — generally equivalent to honest users' activity untraceability — in the context of our bank system is interpreted to the following:

1. *Account-Account-owner Unlinkability.* There should be no way for any entity or collaboration of entities, including the bank and tax authority, to link accounts to a particular user identity.
2. *Account-Account Unlinkability.*There should be no way for any entity or collaboration of entities, including the bank and tax authority, to link different accounts of the same user.

In both cases we consider an adversary who, having corrupted some parties including bank B, is participating in the system for some arbitrary sequence of operations executed by honest and by corrupted parties.

Taking in consideration that each anonymous account is taxable and managed through a pseudonym, we should consider both unlinkability properties w.r.t. pseudonyms and w.r.t. tax reports. More specifically, *account – account-owner unlinkability* can be inducted to account-pseudonym – account-owner unlinkability and account – account tax-report (final form) unlinkability. The first requires that given a pseudonym P_U that does not belong to a corrupted party, the adversary can learn which user owns P_U no better than guessing at random among all non-corrupted peers that appear consistent with P_U. Account – account tax-report unlinkability requires that given a first version of a tax report T that does not belong to a corrupted party, the adversary can learn which user owns T no better than guessing it at random among all non-corrupted users in D_α. In addition, given a transformed (final form) tax report TT, that does not belong to a corrupted party, the adversary can learn which pseudonym (and thus which account) it corresponds to no better than guessing at random among all pseudonyms (accounts) of non-corrupted users in D_α.

In a similar way, *account – account unlinkability* can be interpreted in account-pseudonym – account-pseudonym unlinkability and account-tax report – account tax-report unlinkabilty. *Account-pseudonym – account-pseudonym unlinkability* requires that given two pseudonyms $\mathsf{P}^1_\mathsf{U}, \mathsf{P}^2_\mathsf{U}$ that do not belong to corrupted parties, the adversary has no advantage in telling whether $\mathsf{P}^1_\mathsf{U}, \mathsf{P}^2_\mathsf{U}$ belong to the same user or not. Next, consider an adversary who corrupted some users and the bank as well. Tax report related unlinkability requires that, given two tax reports T^1, T^2 that do not belong to corrupted parties, the adversary has no advantage in telling whether they belong to the same user or not.

Security consists of the following properties:

1. *Fairness.* An accurate statement of the contents or tax liability of all accounts belonging to a given individual is reported to the tax authority per normal practice (i.e., quarterly or annually). More strictly: suppose that n users $\mathsf{U}_1, \ldots, \mathsf{U}_n$ collude together. Let the sum of the tax withheld by all of them together is
$$\mathsf{SumTax} = \sum_{i=1\ldots n} TotalTax^i,$$
where $TotalTax^i$ is U_i's tax amount withheld. Then, fairness requires that the group of users may report in total a minimum of SumTax amount withheld. By fairness, we also require that the following hold:

(a) *Tax Report non-Transferability.* No user should be able to exchange tax report(s) with any other user or use the tax report of another. Assuming two corrupted users U_1 and U_2, where U_1 has issued T^1. Tax Report non-Transferability requires that there is no valid transformation TT^1 of T^1 (through TaxReportTransform) for which the following happens with non negligible probability: if U_2 attempts to deposit TT^1 in honest B through TaxReportDeposit, B accepts.

(b) *Tax Report Unforgeability.* No user or coalition of users should be able to construct a valid tax report for his accounts, i.e., a tax report for which TaxReportDeposit is accepted by the tax authority or the bank.

2. *Privacy preserving Tax Reporting* (covered by privacy property).Tax reports, whether viewed individually or in aggregate, should not reveal any information about the owner or their activities beyond what is necessary for accurate taxation.
3. *Accountability.* Users who attempt to avoid paying taxes for their accounts are traced and punished. We may tentatively assume that this requirement falls in the margin of fairness property, as users who try to avoid paying taxes or attempt to report a higher amount than the amount withheld are caught.

3 Taxation Protocol

In accordance to the two types of accounts it supports, each bank B maintains two databases:

1. the $\mathsf{D}_{\mathsf{reg}}$, which contains the contact and credential information of its clients, as well as the nominal accounts' history information, and
2. the D_{α}, which contains all the anonymous accounts' information, i.e., authorization information, account balance, etc.

As accountability imposes a "privacy-preserving" *centralization* critical, inside the bank, each user can be privately authenticated by demonstrating knowledge of a single master secret, ms_{U}, which he generates at the registration procedure. Users are highly motivated not to share their secret, which they use to open and manage their anonymous accounts. More, specifically the user utilizes his ms_{U} to issue single use, bank (blindly) authorized permissions perm_{α}, which he later deposits anonymously. To manage his accounts, the anonymous user generate account pseudonyms, which are secretly, but provably, connected to their owner's ms_{U}. As privacy requires, pseudonyms of the same user are totally unlinkable one with the other (account-account unlinkability), while pseudonyms, reveal nothing regarding the owner of the ms_{U}(account owner anonymity) without their owner collaboration. However, when a user misbehaves, he risks that his ms_{U} is revealed and his identity and activity completely traced. Users are annually required to deposit to the tax authority an equal amount of tax reports to the number of accounts they own. For tax reporting, we may identify the following user-bank phases:

1. *Tax Report Generation.* It involves three stages:
   (a) *Tax-Report-number Acquisition*, where account owners obtain one valid tax-report-number (TRN) per account. It is important to note that TRNs are not linked to the accounts they are used for.
   (b) *Actual Report Generation*, where the account owner, contacts the bank through his account pseudonym, proves that he is the owner of the account — by demonstrating knowledge of the ms_{U} connected to the account pseudonym — and provides a verifiable commitment to

both his ms_U and TRN. The bank then produces the prime version of the account's tax report:

$$(T^\sigma, T^M) \;=\; (\mathsf{Sig}^x_\mathsf{B}(\text{TaxInfo}),\ \text{TaxInfo}),$$

where

$$\text{TaxInfo} \;= Tax \;||\; \text{Commitment}(\mathsf{TRN},\ \text{Master-Secret}),$$

Tax is the tax withheld from the user's account and by $\mathsf{Sig}^x_\mathsf{B}(M)$, we denote a complicated procedure which involves bank's (x-multiple) signature on M. The exact number of bank signatures applied on M is not revealed to the user. However, the bank provides the user with a randomized token $\mathsf{SigInfo}$ which contains that information, in a form only readable by the taxation authority, along with re-randomization information $\mathsf{SITranform}$ for the user to make $\mathsf{SigInfo}$ unlinkable to its initial form.

(c) *Tax Report Transformation*, where the account owner, applies a transformation function F to both, the bank-signed tax report T^σ, and the corresponding unsigned message, ending up to the depositables

$$TT^\sigma = F(T^\sigma),\ \text{and}\ TT^M = F(T^M).$$

The account owner also transforms $\mathsf{SigInfo}$ through $\mathsf{SITranform}$.

The tax report consists of the pair: (TT^σ, TT^M).

2. *Tax Report Deposit.* Each user deposits all of his tax reports to the bank. The deposit of tax reports includes three stages:
   (a) *Deposit of all the unused* perm_α. In this way, the bank can accurately compute the number of anonymous accounts of each user.
   (b) *Deposit of the depositable tax report pairs*, $(T^{\sigma,i}, T^{M,i})$ corresponding to each account of the accounts owned by the user.The user proves that each tax report pair is valid, i.e., that it corresponds to bank signature(s) (according to the transformed version of $\mathsf{SigInfo}$), that was constructed using the same user's master secret and that is fresh.
   (c) *Tax Amount Calculation procedure*, where, the bank processes the individual tax reports to generate an aggregate tax report for the user, which is revealed with user's collaboration.

3.1 Building Blocks — Primitives for the Suggested Solution

Ecash An E-Cash [CHL05] system consists of three types of players: the *bank*, *users*, and *merchants*. The input and output specifications of the basic operations are as follows. For convenience, we will assume that the operations take place between a merchant M, a user U and the Bank B.

- $(\mathsf{pk}_\mathsf{B}, \mathsf{sk}_\mathsf{B}) \leftarrow \mathsf{EC.BKeyGen}(1^k, params)$ and $(\mathsf{pk}_\mathsf{U}, \mathsf{sk}_\mathsf{U}) \leftarrow \mathsf{EC.UKeyGen}(1^k, params)$, which are the key generation algorithm for the bank and the users respectively.
- $\langle W, \top\rangle \leftarrow \mathsf{EC.Withdraw}(\mathsf{pk}_\mathsf{B}, \mathsf{pk}_\mathsf{U}, n)\ [\mathsf{U}(\mathsf{sk}_\mathsf{U}), \mathsf{B}(\mathsf{sk}_\mathsf{B})]$. U withdraws a wallet W of n coins from B.
- $\langle W', (S, \pi)\rangle \leftarrow \mathsf{EC.Spend}(\mathsf{pk}_\mathsf{M}, \mathsf{pk}_\mathsf{B}, n)\ [\mathsf{U}(W), \mathsf{M}(\mathsf{sk}_\mathsf{M})]$. U spends a coin with serial S from his wallet W to M. W is then reduced to W', M obtains a coin (S, π), where π is a proof of a valid spent coin with a serial S.
- $\langle \top/\bot, L'\rangle \leftarrow \mathsf{EC.Deposit}(\mathsf{pk}_\mathsf{M}, \mathsf{pk}_\mathsf{B})\ [\mathsf{M}(\mathsf{sk}_\mathsf{M}, S, \pi),\ \mathsf{B}(\mathsf{sk}_\mathsf{B}, L)]$. M deposits a coin (S, π) into its account in B. If successful, M's output will be $\top$ and the B's list L of the spent coins will be updated to L'.
- $(\mathsf{pk}_\mathsf{U}, \Pi_G) \leftarrow \mathsf{EC.Identify}(params, S, \pi_1, \pi_2)$. It outputs the public key of the violator U, who spent a coin with serial S twice, producing validity proofs π_1 and π_2, and a proof of guilt Π_G.
- $\top/\bot \leftarrow \mathsf{EC.VerifyGuilt}(params, S, \mathsf{pk}_\mathsf{U}, \Pi_G)$. This algorithm, given Π_G publicly verifies the violation of pk_U.
- $\{(S_i, \Pi_i)\}_i \leftarrow \mathsf{EC.Trace}(params, S, \mathsf{pk}_\mathsf{U}, \Pi_G, D, n)$. This algorithm provides the list of serials S_i of the coins a violator pk_U has issued, with the corresponding ownership proofs Π_i.
- $\top/\bot \leftarrow \mathsf{EC.VerifyOwnership}(params, S, \Pi, \mathsf{pk}_\mathsf{U}, n)$. Given a ownership proof Π it verifies that a coin with serial number S belongs to a user with public key pk_U.

Security Properties: (a) Correctness. (b) Balance. No collection of users and merchants can ever spend more coins than they withdrew. *(c) Identification of Violators.* Given a violation and the corresponding proofs of guilt, the violator's public $\mathsf{pk_U}$ key is revealed such that $\mathsf{EC.VerifyGuilt}$ accepts. *(d) Anonymity of users.* The bank, even when cooperating with any collection of malicious users and merchants, cannot learn anything about a user's spendings. *(e) Exculpability.* An honest user U cannot be accused for conducting a violation such that $\mathsf{EC.VerifyGuilt}$ accepts. *(f) Violators' Traceability.* Given a violator U with a proof of violation Π_G, this property guarantees that $\mathsf{EC.Trace}$ will output the serial numbers of all coins that belong to U with the corresponding ownership proofs.

Pseudonym Systems Pseudonym systems have three types of players: *users, organizations, and verifiers.* Users are entities that receive credentials. Organizations are entities that grant and verify the credentials of users. Finally, verifiers are entities that verify credentials of the users. See [LRSW99][CL01] for more details. The standard operations supported are the following:

- $(\mathsf{pk_O}, \mathsf{sk_O}) \leftarrow \mathsf{PS.OKeyGen}(1^k)$. This procedure generates a public/secret key pair for an organization. We denote a key pair for an organization O by $(\mathsf{pk_O}, \mathsf{sk_O})$.
- $(\mathsf{pk_U}, \mathsf{sk_U}) \leftarrow \mathsf{PS.UKeyGen}(1^k)$. This procedure generates a public/secret key pair for a user. We denote a key pair for a user U by $(\mathsf{pk_U}, \mathsf{sk_U})$. Sometimes we refer to the secret key of a user as a master secret key for the user.
- $\langle (N, \mathsf{NSecr}_N), (N, \mathsf{NLog}_N) \rangle \leftarrow \mathsf{PS.FromNym}(\mathsf{pk_O})\ [\mathsf{U}(\mathsf{pk_U}, \mathsf{sk_U}),\ \mathsf{O}(\mathsf{sk_O})]$. This interactive procedure between a user and an organization generate a pseudonym (or simply nym). The common input is the public key of the organization O. The output for the user is a nym N and some secret information NSecr_N, and for the organization the nym N and some secret information NLog_N.
- $\langle \mathsf{cred}_N, \mathsf{CLog}_{\mathsf{cred}_N} \rangle \leftarrow \mathsf{PS.GrantCred}(N, \mathsf{pk_O})\ [\mathsf{U}(\mathsf{pk_U}, \mathsf{sk_U}, \mathsf{NSecr}_N),\ \mathsf{O}(\mathsf{sk_O}, \mathsf{NLog}_N)]$. This interactive procedure between a user and an organization generate a credential for a nym N. The common input is N and $\mathsf{pk_O}$. The output for the user is the credential cred_N for the nym N. The output for the organization is some secret information $\mathsf{CLog}_{\mathsf{cred}_N}$ for the credential.
- $\langle \top, \top \rangle / \langle \bot, \bot \rangle \leftarrow \mathsf{PS.VerifyCred}(\mathsf{pk_O})\ [\mathsf{U}(N, \mathsf{cred}_N), V]$. In this interactive procedure between a user and a verifier, the user proves that he has a credential on the nym N issued by the organization O.
- $\langle \top, \top \rangle / \langle \bot, \bot \rangle \leftarrow \mathsf{PS.VerifyCredOnNym}\ (N, \mathsf{pk_O}, \mathsf{pk}_{O_1})\ [\mathsf{U}(N_1, \mathsf{cred}_{N_1}), \mathsf{O}(\mathsf{NLog}_N)]$. In this interactive procedure between a user and the organization O, the user proves that N is his valid nym of the organization O and that he has a credential cred_{N_1} on the nym N_1 issued by the organization O_1.

Security Properties. (a) Unique User for Each Pseudonym. Even though the identity of a user who owns a nym must remain unknown, the owner should be unique. *(b) Unlinkability of Pseudonyms.* Nyms of a user are not linkable at any time better than by random guessing. *(c) Unforgeability of Credentials.* A credential may not be issued to a user without the organization's cooperation. *(d) Non-Transferability.* Whenever a user U_1 discloses some information that allows a user U_2 to use her credentials or nyms, U_1 is effectively disclosing her master secret key to him.

Commitment Schemes In a typical commitment scheme, there are provers (let each be P) who are required to commit to a particular value towards verifiers (let each be V), who may be able to see the committed value when provers decide to. The procedures supported are the following:

- $(params) \leftarrow \mathsf{CS.Setup}(1^k)$, which outputs the parameters of a commitment scheme.
- $(\mathsf{C}/false) \leftarrow \mathsf{CS.Commit}(params)[\mathsf{P}(r, m)]$. It outputs either the commitment itself to a value m or *not-completed*. P's input is the message m and randomness r.
- $\langle \top/\bot, m/\bot \rangle \leftarrow \mathsf{CS.Open}(\mathsf{C})[\mathsf{P}(m), \mathsf{V}]$. In this operation the P shows the committed value m to V. V accepts it if m is the value matching C.

Security Properties: (a) Binding. It should be computationally impossible for P, after having committed to m, to generate another message m' that has the same commitment value C. *(b) Hiding.* It should be computationally impossible for a verifier who knows C to get any information regarding m.

Blind Signatures In a typical blind signature scheme, there are signers (let each be S) who produce blind signatures on messages of users (let each be U). The following procedures are supported:

- $(\mathsf{pk_S}, \mathsf{sk_S}) \leftarrow \mathsf{BS.KeyGen}(1^k)$. This is a key-generation algorithm that outputs a public/secret key-pair $(\mathsf{pk_S}, \mathsf{sk_S})$.
- $\langle \top/\bot,\ \sigma/\bot \rangle \leftarrow \mathsf{BS.Sign}(\mathsf{pk_S})[\mathsf{S}(\mathsf{sk_S}),\ \mathsf{C}(m)]$. At the end of this interactive procedure, the output of the S is either *completed* or *not-completed* and the output of U is either the signature (σ) or a failure sign $(\bot)$.
- $\langle \top/\bot \rangle \leftarrow \mathsf{BS.Verify}(m, \sigma, \mathsf{pk_S})$ is a verification algorithm.

Security Properties: (a) Unforgeability. No one but the signer should be able to produce a valid signature σ on a blinded message m. *(b) Blindness* S does not learn any information about the message m on which it generates a signature σ.

Zero Knowledge Proof of Knowledge Protocols In a typical zero knowledge proof of knowledge(ZKPOK) scheme there are two types of players, the provers who need to prove possession of one or more secret number(s), that satisfy a particular property to one or more verifiers and the verifiers. In what follows, we will use the notation introduced by Camenisch and Stadler in [CS97] for the various proofs of knowledge of discrete logarithms and proofs of the validity of statements about discrete logarithms. In particular, $PK\{(\alpha, \beta, \gamma) : y_1 = g_1^\alpha h_1^\beta \wedge y_2 = g_2^\alpha h_2^\gamma \wedge (u \leq \alpha \leq u)\}$ denotes a "zero-knowledge-proof-of-knowledge" of integers α, β and γ such that $y_1 = g_1^\alpha h_1^\beta$ and $y_2 = g_2^\alpha h_2^\beta$, where $u \leq \alpha \leq u$ and $y_1, g_1, h_1, y_2, g_2, h_2$ are all elements of two groups G_1 and G_2 respectively. We make use of the following ZKPoK schemes:
A proof of knowledge of a representation of an element $y \in G$ with respect to bases $z_1, \ldots, z_v \in G$ [CEVDG88], i.e.,
$PK\{(\alpha_1, \ldots, \alpha_v) : y = z_1^{\alpha_1} \cdot \ldots \cdot z_v^{\alpha_v}\}$.
A proof of equality of discrete logarithms of $y_1, y_2 \in G$ to the bases $g, h \in G$ respectively, [C91,CP93] i.e., $PK\{(\alpha) : y_1 = g^\alpha \ \wedge \ y_2 = h^\alpha\}$.
A proof of knowledge of a discrete logarithm of $y \in G$ with base $g \in G$ such that $log_g y$ lies in the interval [a,b], [B00],i.e.,
$PK\{(\alpha) : y = g^\alpha \wedge \alpha \in [a, b]\}$.
Proof of knowledge that the discrete logarithms of two group elements $y_1 \in G_1,\ y_2 \in G_1$ to the bases $g_1 \in G_1$ and $g_2 \in G_2$ in the different groups G_1 and G_2 are equal [BCDG88,CM99], i.e.,
$PK\{(\alpha, \beta) : y_1 =^{G_1} g_1^\alpha \ \wedge \ y_2 =^{G_2} g_2^\alpha \wedge C =^G g^\alpha h^\beta \ \wedge \ \alpha \in [0, min(q_1, q_2)]\}$, where q_1, q_2 are the order of the groups G_1, G_2 respectively, $G =< g >=< h >$ is a group to which the commitment C of α, β is computed.
Security Properties.(a) Correctness. (b) Zero-Knowledge. The verifier learns nothing other than that the prover knows the relevant values. *(c) Proof of Knowledge.* The protocol accepts iff the prover knows the secret value he claims to know.
Pailier Cryptosystem The Paillier cryptosystem is s a probabilistic asymmetric algorithm for public key cryptography and bases its security on the decisional composite residuosity assumption (see [PP99] for details). Assuming the system is meant for a user U to be able to receive messages confidentially, the operations supported are as in every cryptosystem the following:

- $(\mathsf{pk_U}, \mathsf{sk_U}) \leftarrow \mathsf{Pail.KeyGen}(1^k)$, where U generates his encryption key pair. In particular, U chooses two safe large prime numbers p and q, such that $gcd(p-1, q-1) = 2$, computes $n = pq$ and chooses $g \in \mathbb{Z}_{n^2}^*$, such that n divides the order of g. $\mathsf{pk_U} = (n, g), \mathsf{sk_U} = (p, q)$.
- $\langle\ \mathsf{C}/\bot\ \rangle \leftarrow \mathsf{Pail.Encrypt}(\mathsf{pk_U}, m)$, where anyone may use $\mathsf{pk_U}$ to generate ciphertext C on a mesage m: $\mathsf{C} = g^m \cdot r^n (mod n^2)$, where r is randomly chosen.
- $\langle\ m/\bot\ \rangle \leftarrow \mathsf{Pail.Decrypt}(\mathsf{sk_U}, \mathsf{C})$, where U uses his secret key to receive the plaintext.

It is apparent that a particular plaintext may have many ciphertexts, depending on r. We make use of this property in the encryption of x in two ways: (a) two users will not be able to distin-

guish whether they have the same x or not, and are thus unable to know whether they are able to exploit RSA homomorphism; (b) for re-randomization of SigInfo: users who know n can simply compute $\mathsf{C} \cdot (r')^n (mod n^2)$ and generate another ciphertext of x unlinkable to C. Thus in this case of encryption algorithm, SITranform is n.

Security Properties: Semantic security against chosen plaintext attacks (IND-CPA), i.e. given pk, two messages m_1, m_2 and a ciphertext corresponding c to one of them, it is impossible to guess which of the messages corresponds to c with a better probability than 1/2.

RSA Signature Scheme As in a typical signature scheme, there are signers (let each be S) who produce signatures on messages of users (let each be U). The details of the procedures supported are the following:

- $(\mathsf{pk_S}, \mathsf{sk_S}) \leftarrow \mathsf{RSA.KeyGen}(1^k)$, where S chooses two safe large prime numbers $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also large primes, and computes $n = pq$. S chooses also (e, d) such that $1 = ed(mod(\phi(pd)))$, publishes $\mathsf{pk_S} = (e, n)$ and stores $\mathsf{sk_S} = (d, p, q)$.
-
- $\langle\ \sigma/\perp\ \rangle \leftarrow \mathsf{RSA.Sign}(m)[\mathsf{S}(\mathsf{sk_S})]$, where S signs message m using the formula: $\mathsf{RSA.Sign}(m) = m^d(\mod n)$.
-
- $\langle\ \top/\perp\ \rangle \leftarrow \mathsf{RSA.Verify}(\sigma, m, \mathsf{pk}_{signer})[\mathsf{V}]$, where V checks whether $\sigma^e(\mod n)$ equals m.

Security Properties: The most important security property of a signature scheme is *Unforgeability*. Given a message m and assuming that factoring is hard, it is impossible to generate a valid signature that RSA.Verify accepts. In reality the way we defined the signature algorithm, it supports homomorphism, which means that given the signatures $\sigma_{1/2}$ of two messages $m_{1/2}$, one could construct a valid signature σ for $m = m_1 m_2$. We restrict that attack by putting limitations on valid message space such that multiplication of two messages falls outside the accepted message space.

3.2 Detailed Protocol Description

As mentioned before, the bank manages two different registries: one handling users' non-anonymous information (reg-setting) and accounts and another one handling anonymous accounts (α-setting). As each setting is realized as organizations in pseudonymous systems (see section **??** or [CL01] for more details), the bank runs PS.OKeyGen twice, once for the reg-setting and once for the α setting. In particular, the bank:

- generates all the common system parameters (see [CL01]): the length of the RSA moduli ℓ_n, the integer intervals $\Gamma =]-2^{\ell_\Gamma}, 2^{\ell_\Gamma}[$, which is basically the interval master-secrets belong to, $\Delta =]-2^{\ell_\Delta}, 2^{\ell_\Delta}[$, $\Lambda =]-2^{\ell_\Lambda}, 2^{\ell_\Lambda+\ell_\Sigma}[$, such that $\ell_\Delta = \epsilon(\ell_n + \ell_\Lambda) + 1$, where ϵ is a security parameter, and $\ell_\Lambda > \ell_\Sigma + \ell_\Delta + 4$.
- chooses two pairs (one for each setting) of random $\ell_n/2$-bit primes: $p'_{\mathsf{reg}}, q'_{\mathsf{reg}}$ and p'_α, q'_α, such that $p_x = 2p'_x + 1$ and $q_x = 2q'_x + 1$ are prime and sets modulus $n_x = p_x \cdot q_x$, where $x = \mathsf{reg}, \alpha$.
- chooses random elements $a_x, b_x, d_x, g_x, h_x \in QR_{n_x}$, where $x = \mathsf{reg}, \alpha$. In addition to the standard organization setup procedure of [CL01], the bank also chooses random $k_\alpha, l_\alpha, m_\alpha, s_\alpha, z_\alpha \in QR_{n_\alpha}$.

Thus, the Bank's public-secret information for the two settings are

- $\{(n_{\mathsf{reg}}, a_{\mathsf{reg}}, b_{\mathsf{reg}}, d_{\mathsf{reg}}, g_{\mathsf{reg}}, h_{\mathsf{reg}}), (p_{\mathsf{reg}}, q_{\mathsf{reg}})\}$, for the reg-setting, and

– $\{(n_\alpha, a_\alpha, b_\alpha, d_\alpha, g_\alpha, h_\alpha, k_\alpha, l_\alpha, m_\alpha, s_\alpha, z_\alpha), (p_\alpha, q_\alpha)\}$, for the α-setting.

In addition to the aforementioned parameters, the bank generates a blind signature key pair $(\mathsf{pk}_\mathsf{B}^b, \mathsf{sk}_\mathsf{B}^b)$ and an RSA signature key pair,

$$\{sk_\mathsf{B}, pk_\mathsf{B}\} = \{(d, p_\alpha, q_\alpha), (e, n_\alpha)\},$$

based on the α RSA-parameters and $1 < e < \phi(p_\alpha q_\alpha)$ and $de = 1(mod(\phi(p_\alpha q_\alpha)))$. e is given to the taxation authority (TA).

On the other hand, TA generates an encryption key pair (pk_TA, sk_TA) of a known randomized homomorphic encryption scheme (Paillier etc) and provides the bank with the encryption key (see Appendix **??** or [PP99] for more details).

In what follows, we will assume that a user U collaborates with a bank B to open anonymous accounts, handle them and be taxed for.

Registration. In this phase, the user U contacts the bank *in person* to create an entry in the latter's D_reg registry. This is a prerequisite for users to open (anonymous) accounts with B.

1. U provides identification credentials to B(i.e. passport, etc.)
2. U runs PS.UKeyGen to obtains a bank-oriented master secret ms_U and a public/secret key pair $\{\mathsf{pk}_\mathsf{U}\mathsf{B}, \mathsf{sk}_\mathsf{U}\mathsf{B}\}$ connected to his ms_U.
3. U runs PS.FormNym using the reg-parameters to generate a bank pseudonym $\mathsf{P}^{\mathsf{reg}}$ ([CL01]), connected to ms_Uin zero knowledge fashion.
4. U and B U $\leftrightarrow$ B: execute EC.Withdraw procedure for U (see **??** or or [CHL05] for more details) to withdraw a wallet $\mathsf{WAcc}_\mathsf{U}^\mathsf{B}$ of perm_α (ecoins). $\mathsf{WAcc}_\mathsf{U}^\mathsf{B}$ will later authorize U to open anonymous accounts in B. Consequently, the size of the wallet withdrawn depends on the maximum number of anonymous accounts U is eligible for.
5. U $\leftrightarrow$ B: execute PS.GrantCred procedure so that U obtains a registration credential $\mathsf{cred}_\mathsf{U}^\mathsf{B}$ for having registered in D_reg, which is provably connected to ms_U.
6. U stores in his database his secret key ($\mathsf{sk}_\mathsf{U}\mathsf{B}$), the information related to his pseudonym($\mathsf{pubP}^{\mathsf{reg}}$, $\mathsf{secP}^{\mathsf{reg}}$) and credentials ($\mathsf{pubcred}_\mathsf{U}^\mathsf{B}$, $\mathsf{seccred}_\mathsf{U}^\mathsf{B}$), while B stores only the public information.

Account Opening. To open an anonymous account, user U contacts B initially anonymously. Both, B and U make use of the α-parameter group. The following interactions take place:

1. U(anonymous) $\leftrightarrow$ B: run EC.Spend for U to spend an ecoin (S, π) (perm_α) from his $\mathsf{WAcc}_\mathsf{U}^\mathsf{B}$ wallet. If the ecoin used has been spent before, B runs the EC.Identify and EC.Trace procedures to recover U's identity($\mathsf{pk}_\mathsf{U}\mathsf{B}$) and activity ($\mathsf{sk}_\mathsf{U}\mathsf{B}$).
2. user: runs PS.FormNym, to generate a pseudonym P^i for managing his new account α^i. The pseudonym created has the form of
$$\mathsf{P} = a_\alpha^{\mathsf{ms}_\mathsf{U}} b_\alpha^s,$$
where s is a U-B-generated value known only to U (see [CL01]).
3. U(anonymous) $\leftrightarrow$ B: run PS.VerifyCredOnNym, where U demonstrates ownership of $\mathsf{cred}_\mathsf{U}^\mathsf{B}$ and B verifies both, that $\mathsf{cred}_\mathsf{U}^\mathsf{B}$ and P^i are bound to the same ms_U (user) and that their owner has registered to the bank with a reg-pseudonym which is bound to the same ms_U as P^i.
4. U stores in his database the public/secret information related to his account-pseudonym (pubP^i, secP^i). B stores (pubP^i, S, π).

Tax Report Issue. This is a procedure taking place between the owner U of an account α^i, who participates through his pseudonym P^{i} and the bank B. It consists of three stages:

1. *Tax Report Number Acquisition.* The account pseudonym P^{i} collaborates with B in a $\mathsf{BS.Sign}$ procedure, for the former to obtain a (blind towards B) TRN related ticket trtick^i. U deposits *in person* to B the trtick^i to receive a tax-report-number TRN^i. B sends to TA the tuple (U, TRN^i) and stores it in its $\mathsf{D_{reg}}$. Tax report numbers are chosen from a range $\mathsf{Range_T}$, such that the sum of any two tax report numbers will result in a number out of of the corresponding valid range, i.e.,:
$$\forall \mathsf{TRN}^i, \mathsf{TRN}^j \in \mathsf{Range_T} : \mathsf{TRN}^i + \mathsf{TRN}^j \notin \mathsf{Range_T}.$$
2. *Tax Report Generation.* The following take place:
   (a) P^{i}: using $\mathsf{secP}^{\mathsf{i}}$ proves that he is the owner of P^{i}, by engaging in the ZKPoK:
   $$PK\{(\beta,\gamma) : (\mathsf{P}^{\mathsf{i}})^2 = (a_\alpha^2)^\beta \cdot (b_\alpha^2)^\gamma\}.$$
   (b) $\mathsf{P}^{\mathsf{i}} \rightarrow \mathsf{B}$: $C = \mathsf{Com}(\mathsf{ms_U}, \mathsf{TRN}^i, r^i) = k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{\mathsf{TRN}^i} \cdot m_\alpha^{r^i}$,
   where Com is a tax report related commitment scheme, $\mathsf{ms_U}$ U's master-secret, TRN^i, the single-use tax-report-number, which U acquired anonymously, and r^i is a U-generated randomness.
   (c) $\mathsf{P}^{\mathsf{i}} \leftrightarrow \mathsf{B}$: execute the following ZKPoK protocol for P^{i} to show in zero knowledge fashion that C was computed correctly, i.e., that the committed master secret matches the master secret used in the construction of P^{i} ($\mathsf{ms_U}$) and that the exponent of l_α (TRN^i) is among the specified range:
   $$PK\{(\gamma,\delta,\varepsilon,\eta) : (\mathsf{P}^{\mathsf{i}})^2 = (a_\alpha^2)^\gamma (b_\alpha^2)^\delta \ \wedge \ C^2 = (k_\alpha^2)^\gamma (l_\alpha^2)^\eta (m_\alpha^2)^\varepsilon \ \wedge$$
   $$\gamma \in \Gamma \ \wedge \ \delta \in \Delta \ \wedge \ \eta \in \mathsf{Range_T}\}.$$
   (d) P^{i} *ra* B: a random r_x; if B has received r_x before, the procedure is repeated.
   (e) B: decides x based on r_x. It then computes $h_\alpha^{\mathsf{tax}_i}$ and uses his RSA signature key to *sign* $T^{M,i} = h_\alpha^{\mathsf{tax}_i} \cdot C$ x times into $T^{\sigma,i}$. B provides U with an x-related the secret piece of information $\mathsf{SigInfo} = \mathsf{Enc_{TA}}(x)$, where $x \in Range_x$. $T^{\sigma,i}$ is then:
   $$T^{\sigma,i} = h_\alpha^{d^x \mathsf{tax}_i} \cdot k_\alpha^{d^x \mathsf{ms_U}} \cdot l_\alpha^{d^x \mathsf{TRN}^i} \cdot m_\alpha^{d^x r^i} (mod n_\alpha).$$
   (f) $\mathsf{B} \rightarrow \mathsf{U}$: $T^{\sigma,i}$, $\mathsf{SigInfo}$ and $\mathsf{SigInfo}$ re-randomization information $\mathsf{SITranform}$.
3. *Tax Report Transformation.*n this case, after having obtained his signed tax reports, U applies the transformation function F, so that — although provably valid — the modified tax reports are unlinkable to their initial form. In our scheme $F(M)$ is instantiated by adding an extra factor to M. In particular, U
   (a) transforms both $T^{\sigma,i}$ and $T^{M,i}$ using $F(M,r) = M \cdot s_\alpha^{r_1} \cdot z_\alpha^{r_2}$, where M is the message to be transformed and $r = r_1 || r_2$ is a U-specified randomness. Thus, we get the following for the signed tax report and the corresponding message, respectively,
   $TT^{\sigma,i} \leftarrow F(T^{\sigma,i}, r^{\sigma,i}) \leftarrow h_\alpha^{d^x \mathsf{tax}_i} \cdot k_\alpha^{d^x \mathsf{ms_U}} \cdot l_\alpha^{d^x \mathsf{TRN}^i} \cdot m_\alpha^{d^x r^i} \cdot s_\alpha^{r_1^{\sigma,i}} \cdot z_\alpha^{r_2^{\sigma,i}}$
   $TT^{M,i} \leftarrow F(T^{M,i}, r^{M,i}) \leftarrow h_\alpha^{\mathsf{tax}_i} \cdot k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{\mathsf{TRN}^i} \cdot m_\alpha^{r^i} \cdot s_\alpha^{r_1^{M,i}} \cdot z_\alpha^{r_2^{M,i}}$.
   (b) re-randomizes the encryption of $\mathsf{SigInfo}$ according to $\mathsf{SITranform}$

Tax Report Deposit. Each user U(using a real identity) sends to the TA all the tax reports he has acquired, $(TT^{\sigma,1}, TT^{M,1}), \ldots, (TT^{\sigma,N}, TT^{M,N})$, where N is the number of U's accounts. U then proves that each one of these pairs were constructed in a correct way and that they correspond to *his* accounts. The tax report validation consists of two steps:

1. *Signature Validation*, where U shows that $(TT^{\sigma,i}, TT^{M,i})$, for all $i = 1 \ldots N$, correspond to transformations of bank-signatures:
   (a) TA: decrypts $\mathsf{SigInfo}$, reads x and raises all $TT^{\sigma,i}$s to B's signature verification key e, x times using $(modn_\alpha)$:
   $TT^{M',i} \leftarrow (TT^{\sigma,i})^{e^x} \leftarrow h_\alpha^{\mathsf{tax}_i} \cdot k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{\mathsf{TRN}^i} \cdot m_\alpha^{r^i} \cdot s_\alpha^{e^x r_1^{\sigma,i}} \cdot z_\alpha^{e^x r_2^{\sigma,i}}$.
   (b) $\mathsf{U} \leftrightarrow \mathsf{TA}$: interact in the following ZKPoK protocol to prove that in each pair, $TT^{M,i}$ and $TT^{M',i}$ correspond to the same $TaxInfo$, i.e., that in both cases the exponents of $h_\alpha, k_\alpha, l_\alpha, m_\alpha$ are the same, or that $\frac{TT^{M,i}}{TT^{M',i}}$ is a factor of powers of s_α and z_α:

$$PK\{(\theta,\eta)\} : (\frac{TT^{M,i}}{TT^{M',i}})^2 = (s_\alpha^2)^\theta (z_\alpha^2)^\eta.$$

2. *Tax Report Ownership and non-Repetition Proof.* where U proves to the tax authority TA that each one of the tax reports he deposits had been created through his collaboration with B and that he has not deposit the same tax report twice. The latter is achieved through the one-time-use TRN s. For each one of $TT^{M,i}$s (or $TT^{M',i}$), U reveals the TRN^i to the TA, while he engages to a ZKPoK protocol for the TA to verify that the exponent of k_α in $TT^{M,i}$ (and thus,$TT^{M',i}$) matches the $\mathsf{ms_U}$ used in P^B, i.e.,

$$PK\{(\gamma,\delta,\tau,\varepsilon,\theta,\eta) : (\mathsf{P}^{\mathsf{reg}})^2 = (a_\alpha^2)^\gamma (b_\alpha^2)^\delta \wedge$$
$$\wedge \frac{TT^{M,i}}{l_\alpha^{\mathsf{TRN}^i}} = h_\alpha^\tau \cdot k_\alpha^\gamma \cdot m_\alpha^\varepsilon \cdot s_\alpha^\theta \cdot z_\alpha^\eta \ \wedge \gamma \in \Gamma \ \wedge \ \delta \in \Delta\}.$$

Total Tax Calculation. In this operation, TA confirms that U has deposited tax reports for all of his accounts and then uses them to extract the overall tax amount withheld by U's accounts. In particular, TA and U collaborate in an EC.Spend procedure for the latter to spend his unused ecoins from $\mathsf{WAcc}_\mathsf{U}^\mathsf{B}$ wallet. TA then estimates the exact number of U's accounts and computes the overall tax withheld as follows:

1. TA: computes the product of all $TT^{M',i}$, which because of the homomorphism of the commitment scheme used, equals to

$$\prod_{i=1,\ldots,N} TT^{M',i} = \prod_{i=1,\ldots,N} (h_\alpha^{\mathsf{tax}_i} \cdot k_\alpha^{\mathsf{ms_U}} \cdot l_\alpha^{trni} \cdot m_\alpha^{r^i} \cdot s_\alpha^{r_1^{M,i}} \cdot z_\alpha^{r_2^{M,i}}) =$$
$$h_\alpha^{TotalTax} \cdot k_\alpha^{N\mathsf{ms_U}} \cdot l_\alpha^{R_t} \cdot m_\alpha^{\sum_{i=1,\ldots,N} r^i} \cdot s_\alpha^{\sum_{i=1,\ldots,N} r_1^{M,i}} \cdot z_\alpha^{\sum_{i=1,\ldots,N} r_2^{M,i}}.$$

2. U reveals $TotalTax = \sum_{i=1\ldots N} \mathsf{tax}_i$, which is the overall tax withheld.
3. U and TA collaborate in a ZKPoK protocol to prove that $\frac{T_i'}{h_\alpha^{TotalTax} . l_\alpha^{R_t}}$ is correctly created and thus prove that $TotalTax$ is the required amount (note that TA knows R_t):

$$PK\{(\beta,\gamma,\delta)\mathsf{P}^{\mathsf{reg}2} = (a_\alpha^2)^\gamma (b_\alpha^2)^\delta \ \wedge \ \frac{T_i'}{h_\alpha^{TotalTax} l_\alpha^{R_t}} = (k_\alpha^N)^\gamma m_\alpha^\varepsilon \ \wedge \ \gamma \in \Gamma \ \wedge \ \delta \in \Delta\}.$$

4 Discussion

In this section, we will discuss how each one of our privacy and security requirements are satisfied. We also discuss deployability.

Security - Privacy The following theorem states the correctness, privacy and security of our general scheme:

Theorem. if the underlying primitives (anonymous credential system, e-cash system, blind signatures, commitments and ZKPoK) are secure, then our scheme satisfies *correctness*, *account–account unlinkability*, *account–account-owner unlinkability*, *fairness in tax reporting*, *tax report non transferabiliy*, *tax report unforgeability*, and *accountability*.

We use prove this theorem with the following lemmas.

Lemma 1. If the underlying primitives (anonymous credential system, e-cash system, commitments and ZKPoK) are secure, then our scheme satisfies *Correctness*.
Proof. The first condition of correctness is satisfied directly through the *correctness* of the underlying schemes of ecash and anonymous credentials and according to which if U is honest neither $\mathsf{EC.Spend}$ procedure of perm_α nor $\mathsf{PS.VerifyCredOnNym}$ (which take place at the $\mathsf{Account\ Open}$ will output an error message. The $\mathsf{correctness}$ and $\mathsf{verifiability}$ of the RSA signature scheme, its homomorphism and the $\mathsf{correctness}$ of the used ZKPoK protocols used to confirm that U is the owner of all tax reports and guarantee that $\mathsf{TaxReportDeposit}$ will not output an error message.

Lemma 2. If the underlying primitives (anonymous credential system, ecash system, and ZKPoK) are secure, then our scheme satisfies *account-account unlinkability*.
Proof. Account-account unlinkability is maintained in the $\mathsf{Account\ Open}$ procedure through the *unlinkability* property of the ecash scheme used for perm_α and the *unlinkability of pseudonyms* property of the underlying anonymous credential system. Account-account unlinkability is also maintained through the tax reporting: Let α^1 and α^2 two accounts of U for which he obtains tax reports T^1, T^2 respectively. Then T^1 and T^2 are unlinkable one to the other because of the *hiding* property of the commitment scheme used to generate them and the *zero knowledge* property of the ZKPoK scheme used to prove their correct construction.

Lemma 3. If the underlying primitives (anonymous credential system, ecash system, blind signatures, commitment and ZKPoK, transformation function F, Paillier cryptosystem) are secure, then our scheme satisfies *account–account-owner unlinkability*.
Proof. Let α^i an anonymous account of user U managed by pseudonym $\mathsf{P^i}$. Let T and TT be the tax report for α^i and its transformed version. Unlinkability of α^i and U at the $\mathsf{AccountOpen}$ procedure is achieved through the *anonymity* property of the ecash scheme realizing perm_αs and of and pseudonym system used for the generation of $\mathsf{P^i}$ as well as through the *blindness* of the blind signature scheme used for the acquisition of TRNs. T is unlinkable to U through the *hiding* property of the commitment scheme, which "hides" the $\mathsf{ms_U}$ committed in T and the security (*zero knowledge*) of the ZKPoK protocol used to validate the construction of T: no information is leaked neither TRN nor for $\mathsf{ms_U}$ contained in T. TT on the other hand, does not reveal anything regarding

T or the account because of the *hiding* property of transformation function F (see appendix for more details) used for its construction, the zero knowledge property of the ZKPoK protocol used at its validation and the re-randomization property of the Paillier cryptosystem used for blinding SigInfo.

Lemma 4. If the underlying primitives (anonymous credential system, digital signatures, commitment) are secure, then our scheme satisfies *Tax Report Unforgeability*.

Proof. Let that user U manages an account α_{U} through a pseudonym P and generates tax report $T^{\sigma/M}$, which is later transformed to $TT^{\sigma/M}$, through $F()$. We need to prove that the tax report remains unforgeable at all stages. T is an RSA-signature-based function on a commitment on TRN, tax_i and ms_{U}. To avoid B-signature forgeries exploiting RSA homomorphism, apply the signature scheme on T^M x number of times, while the RSA-signature verification key and x are kept secret to U. x is only revealed to TA only at the TT deposit procedure through SigInfo. We assume that the granularity of different x-es is very small w.r.t. the total number of tax reports so that linkability attacks do not apply. U has no incentive to alter SigInfo. To avoid such a forgery using the same tax report, we make use of TRN s, B-chosen numbers of a pre-specified range such that summations of two numbers in $\mathsf{Range}_{\mathsf{T}}$ result in an invalid number. *Bindness* property of the commitment scheme used in T generation guarantees that as long as the RSA signature is unforgeable, U cannot dispute the $TaxInfo$ he has committed to in T^M.

Lemma 5. If the underlying primitives (anonymous credential system, digital signatures, commitment and ZKPoK) are secure, then our scheme satisfies *Tax Report non transferability*.

Proof. In our system users are highly motivated not to share their ms_{U}. Thus, assuming that they are not doing so, Tax-Report non transferability is achieved through the need to prove knowledge of the ms_{U} at each step of the tax reporting. More specifically, account pseudonyms are required to show that their ms_{U} matches the one committed in T, which is then signed and -thus- cannot change (unforgeability of the signature scheme). The *proof of knowledge* property of the ZKPoK scheme used when depositing TT, guarantees that user depositing TT knows the corresponding ms_{U}, which should match the ms_{U} used in all tax reports deposited by the same user, as well as his registration pseudonym.

Lemma 6. If the underlying primitives (anonymous credential system, ecash, digital signatures, commitment and ZKPoK) are secure, then our scheme satisfies *Fairness*.

Proof. Because of Tax Report Unforgeability and non-transferability, users cannot change the tax reported in each report or use other users' tax reports. Because of the Identification of Violators and Violators' Traceability property of ecash implementing perm_{α}s, users cannot lie to the bank regarding the number of the accounts they have opened: if they try to prove they opened fewer accounts, some of the perm_{α}s in $\mathsf{WAcc}^{\mathsf{B}}$ will be doublespent. At the same time, because of the TRNs, users cannot avoid a tax report, by depositing another one twice.

Lemma 7. If the underlying primitives (anonymous credential system, ecash, digital signatures, commitment and ZKPoK) are secure, then our scheme satisfies *Accountability*.

Proof. Because of the *Identification of Violators* and Violators' Traceability property of ecash implementing perm_{α}s, users who lie regarding the anonymous accounts they opened are identified. Because of the *proof of knowledge* property of the ZKPoK protocols, the *non-transferability of cre-*

dentials property of the underlying pseudonym system and the *non-transferability* property of tax reports, users trying to use other users' tax reports are detected.

Lemma 8. The transformation function F, defined on $D_M x \mathbb{Z}$, where:

- $F(M,r) = M \cdot s_\alpha^{r_1} \cdot z_\alpha^{r_2} (mod n_\alpha)$, $n_\alpha = p_\alpha \cdot q_\alpha$, $p_\alpha,\ q_\alpha$ safe primes, $s_\alpha, z_\alpha \in QRn_\alpha$, $r = r_1 || r_2$ is a random number and M the message to be blinded;
- $D_M = \{x : \exists y, z, w, j : x = h_\alpha^y \cdot k_\alpha^z \cdot l_\alpha^w \cdot m_\alpha^j (mod n_\alpha)\}$, where $h_\alpha, k_\alpha, l_\alpha, m_\alpha \in QRn_\alpha$ are system parameters;

is computationally non-invertible and provides output indistinguishability w.r.t. M-inputs. More specifically, we claim that F supports:

- *Non Invertibility* Given an output f of $F()$ it is computationally impossible to compute $M \in D_M$ and r such that F(M,r) = f.
- *Input-Output Unlinkability* Given two messages M_1 and M_2 and an output f of $F()$ which corresponds to one of the messages, it is computationally hard to decide which message corresponds to f with a better probability than 1/2.

Proof. Both properties derive directly from the discrete log assumption modulo a safe prime product and strong RSA assumption.

Lemma 9. The function Com used, defined on $(\mathbb{Z}x\mathbb{Z})x\mathbb{Z}$, where

$$\mathsf{Com}(x, y; r) = k_\alpha^x \cdot l_\alpha^y \cdot m_\alpha^r (mod n_\alpha),$$

$n_\alpha = p_\alpha \cdot q_\alpha$, $p_\alpha,\ q_\alpha$ safe primes, $k_\alpha, l_\alpha, m_\alpha \in QR_{n_\alpha}$ is a commitment scheme on x, y with randomness r.

Proof. Function Com satisfies both properties *bindness* and *hiding* which derives from the discrete log assumption modulo a product of safe primes and factoring assumption.

Deployability Any real-world deployment of this scheme must be affordable, and must interface correctly with the existing worldwide banking system. Although this is not an implementation paper, we believe that these requirements are satisfied. First, it is easy to generate International Bank Account Numbers (IBANs) from our accounts (see Appendix A.2); these accounts can thus be used to send and receive payments. Second, rough calculations suggest that by using modern hardware designs, the hardware necessary for timely tax reporting and verification is affordable by the organizations concerned; furthermore, the expense for banks is proportional to the number of customers they have. Details are given in Appendix A.1.

5 Related Work

[JY96], [LMP96] are cases of protocols providing conditionally anonymous payments from user issued bank accounts. However, their work is different from ours as there is either a third trusted party involved for anonymity revocation purposes, or they do not offer privacy against coalitions of banks. In [AB09], the authors provide privacy in the management of anonymous accounts, even w.r.t. the bank through the use of anonymous credit cards. However, we take an additional step in addressing tax reporting for bank accounts, which is not an issue in credit-only systems.

Taxation has been addressed in the past in the stock market. In [XYZ00], the authors propose a scheme addressing a similar problem to ours: anonymous and taxable stock market trading accounts. As in our system, users are using a generated anonymous credential from a public credential to validate anonymous stock-transaction. However, their system differs from our own in two major ways. First, they only allow for each user to own one anonymous account, because of the extra complications to tax reporting the multiple accounts would cause. Addressing these complications is one of our major contributions. Secondly, they do not aim to prevent the Tax Authority from learning which accounts the reports are coming from. Thus if the TA were to collaborate with the Stock Exchange Center, they could re-link the users with their anonymous accounts. Preventing this is another contribution of our system.

6 Conclusion

In this paper we presented a privacy preserving bank account system, where individuals may open arbitarily anonymous and unlinkable accounts w.r.t. the bank and tax authority collaborations. All accounts are ultimately and in zero knowledge fashion connected to their owner. We emphasize on the bank account taxation mechanism, where individual users report the aggregated amount of tax all of their accounts have been withheld in a fair and accountable way.

References

[AB09] E. Androulaki and S. Bellovin. An anonymous credit card system. In *TrustBus '09: Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business*, pages 42–51, Berlin, Heidelberg, 2009. Springer-Verlag.

[B00] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431–444, 2000.

[BCDG88] E. F. Brickell, D. Chaum, I. Damgård, and J. v. d. Graaf. Gradual and verifiable release of a secret. In *CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, pages 156–166, London, UK, 1988. Springer-Verlag.

[C91] D. Chaum. Zero-knowledge undeniable signatures. In *Advances in Cryptology EUROCRYPT 90*, 1991.

[CEVDG88] D. Chaum, J.-H. Evertse, and J. van de Graaf. An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In *Advances in Cryptology EUROCRYPT 87*, 1988.

[CHL05] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer-Verlag, 2005.

[CL01] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology - EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer-Verlag, 2001.

[CM99] J. Camenisch and M. Michels. Separability and efficiency for generic group signature schemes. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 413–430, London, UK, 1999. Springer-Verlag.

[CP93] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO '92: Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, pages 89–105, London, UK, 1993. Springer-Verlag.

[CS97] J. Camenisch and M. Stadler. Effcient group signature schemes for large groups. In *Advances in Cryptology — CRYPTO '97*, volume 1296 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 1997.

[JY96] M. Jakobsson and M. Yung. Revokable and versatile electronic money (extended abstract). In *CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security*, pages 76–87, New York, NY, USA, 1996. ACM.

[LMP96] S. Low, N. F. Maxemchuk, and S. Paul. Anonymous credit cards and its collusion analysis. *IEEE Transactions on Networking*, December 1996.

[LRSW99] A. Lysyanskaya, R. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography '99*, volume 1758 of *Lecture Notes in Computer Science*, pages 184–199. Springer-Verlag, 1999.

[PP99] P. Paillier and D. Pointcheval. Efficient public-key cryptosystems provably secure against active adversaries. In K. Y. Lam and E. Okamoto, editors, *Advances in Cryptology ASIACRYPT 99*, volume 1716, page 165179. Springer-Verlag, 1999.

[XYZ00] S. Xu, M. Yung, and G. Zhang. Scalable, tax evasion-free anonymous investing, 2000.

A Technical Issues

Any real-world deployment of this scheme must be affordable, and must interface correctly with the existing worldwide banking system. Although this is not an implementation paper, we believe that these requirements are satisfied. First, it is easy to generate International Bank Account Numbers (IBANs) from our accounts (see Appendix A.2); these accounts can thus be used to send and receive payments. Second, rough calculations suggest that by using modern hardware designs, the hardware necessary for timely tax reporting and verification is affordable by the organizations concerned; furthermore, the expense for banks is proportional to the number of customers they have. Details are given in Appendix A.1.

A.1 Performance Issues

The protocols we describe are somewhat expensive, because they require a fair number of exponentiations. That said, we believe the cost is affordable.

Since opening accounts and subaccounts are uncommon operations, we ignore them. The real cost is in preparing and processing tax reports. The cost there is borne by all three parties: the individual, the bank, and the tax authority. We now analyze the cost to the latter two; the cost to the former is almost certainly minimal, since no one person will have that many accounts. Besides, the individual reaps the privacy benefits of the protocol, and hence is motivated to pay for it.

The total cost is determined by the number of subaccounts per person, and the number of individuals who use this protocol. We bound the latter by using the IRS's figure of about 150,000,000 individual tax return filers. Obviously, if not everyone is using these mechanisms, the total cost will be proportionally less.

The former is much more difficult to ascertain. Instead, we attempt to estimate the maximum number of subaccounts a typical individual would have, by assuming that a separate subaccount is used for each monthly check written; see Table 1. To set an upper bound, we double that. Assuming there is a tax report once per quarter, we estimate that no more than 22,500,000,000 ($2.3 \cdot 10^{10}$) accounts exist. Reporting for each account requires one exponentiation and two zero knowledge proofs of knowledge. We estimate that the latter require twenty exponentiations apiece; each quarter, the tax authority thus must perform about $4.8 \cdot 10^{11}$ exponentiations.

A reasonably modern CPU can do about 150 2048-bit exponentiations/second. However, we can use dedicated exponentiation chips, which can do about 25,000 operations a second, or we can use the graphics processor (GPU) to do calculations. According to Szerwinski and Guñeysu [**?**], a GPU can do about 100 2048-bit exponentiations/second. Newer GPUs are considerably faster, and have considerably more parallelisms; in particular, the forthcoming Nvidia GeForce 300 Series will have 512 cores and should operate considerably faster. It seems reasonable to assume that we can reach speeds of 1000 exponentiations/second on a GPU-equipped computer. We estimate that the fully loaded cost of such a 1U server to be about $5,000, counting the computer itself, the rack, power, cooling, and real estate. A data center with 10,000 such machines would therefore

Food	5
Car loan	2
Phone bils	2
Credit cards	6
Insurance	2
Cable TV	1
Rent/mortgage	1
Internet	1
Heat	1
Electricity	1
Water	1
Garbage	1
Lawn care	1
Total	25

Table 1. A conservative estimate of the maximum number of subaccounts, and hence checks, a typical individual will write each month.

cost about $50M, with the computers amortized over a three-year lifetime; the annual cost is thus about $17M. While not cheap, the cost is low compared to the current cost of running, say, the IRS, whose fiscal year 2008 budget was $11B. Such a complex could do $8.6 \cdot 10^{11}$ exponentiations/day; it would therefore take little more than half a day to process the tax reports, a value that is clearly acceptable. Our estimates could easily be off by a factor of 30 or more without changing the basic result: this protocol will not impose an undue processing burden on the tax authority.

It is rather more difficult to do the same calculation for banks, since figures on the number of depositors do not seem to be readily available. We can, nevertheless, perform some approximations. Preparing the tax reports requires a single zero knowledge proof of knowledge for each account; the cost per depositor (and hence the size of the data center) is thus roughly half of the tax authority's cost per filer. We approximate the number of customers who would desire such a service as the number of customers who use online banking today. According to their 2008 annual reports, Wells Fargo has 11,000,000 such customers; Bank of America, another large bank, had about 30,000,000 online customers. For the latter, then, the cost of the data center is about 10% of the tax authority's cost, or $5,000,000, an amount easily affordable for such a large institution.

A.2 Bank Account Number Generation

Although we have bank subaccounts which are identified by public keys, our system must be built on top of an existing bank account system. In this system, each sub account will correspond to a real world bank account that holds funds in the normal way, and will transact according to signed commands given by the account owner.

Because it is a real bank account, these sub accounts must also be associated with a bank account number. We cannot use a public key, which could be hundreds of decimal digits, as a bank account number. A bank account number is often handled by humans, it would not be practical for normal bank paperwork and operations.

For the sake of generality, we will assume that we will conform to the standard for International Bank Account Numbers (IBANs). An IBAN is prefixed by a 2-letter country code, followed by 2 check digits (for error correction), and up to thirty alphanumeric characters for domestic bank

account numbers. Since the country code and check digits are immutable, our aim is to map a public key to the remaining 30 alphanumeric characters.

To address this, we will use an idea based on Cryptographically Generated Addresses [**?**]. This protocol is intended for mapping cryptographic public keys to the last 64 bits of an IPv6 address. This is a similar problem; a server is associated with a public key, but must be identified by an identifier in a much smaller space than that which would be necessary to keep public keys secure.

The thirty alphanumeric characters of an IBAN are equivalent to slightly more than 141 bits of information. CGA is based on the SHA-1 hash function, and we will adapt it as follows.

1. Choose a security parameter l (an integer from 0-7).
2. Choose a random modifier.
3. Concatenate the modifier with the public key and take the SHA-1 hash.
4. Repeat step 3, incrementing the modifier until the leftmost $16l$ bits of the result are zero.
5. Set the collision count to zero.
6. Concatenate the final modifier, the collision count, the country code, and the public key.
7. Take the SHA-1 hash of this value, and reinterpret the leftmost 141 bits as a 30-alphanumeric character bank account number.
8. Check for collisions, if there exists one increment the collision count and repeat from step 7.

In many cases, the IBAN will actually be structured: the first several digits will identify a bank, while the remainder identifies an account within the bank. The modifications to accommodate fewer digits are straightforward and will not be discussed further here.

A.3 RSA Signature Scheme

As in a typical signature scheme, there are signers (let each be S) who produce signatures on messages of users (let each be U). The details of the procedures supported are the following:

- $(\mathsf{pk_S}, \mathsf{sk_S}) \leftarrow \mathsf{RSA.KeyGen}(1^k)$, where S chooses two safe large prime numbers $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are also large primes, and computes $n = pq$. S chooses also (e, d) such that $1 = ed(mod(\phi(pd)))$, publishes $\mathsf{pk_S} = (e, n)$ and stores $\mathsf{sk_S} = (d, p, q)$.
-
- $\langle\ \sigma/\perp\ \rangle \leftarrow \mathsf{RSA.Sign}(m)[\mathsf{S}(\mathsf{sk_S})]$, where S signs message m using the formula: $\mathsf{RSA.Sign}(m) = m^d(\mod n)$.
-
- $\langle\ \top/\perp\ \rangle \leftarrow \mathsf{RSA.Verify}(\sigma, m, \mathsf{pk}_{signer})[\mathsf{V}]$, where V checks whether $\sigma^e(\mod n)$ equals m.

Security Properties: The most important security property of a signature scheme is *Unforgeability*. Given a message m and assuming that factoring is hard, it is impossible to generate a valid signature that $\mathsf{RSA.Verify}$ accepts. In reality the way we defined the signature algorithm, it supports homomorphism, which means that given the signatures $\sigma_{1/2}$ of two messages $m_{1/2}$, one could construct a valid signature σ for $m = m_1 m_2$. We restrict that attack by putting limitations on valid message space such that multiplication of two messages falls outside the accepted message space.