Transferable E-cash: A Cleaner Model and the First Practical Instantiation

Balthazar Bauer[1], Georg Fuchsbauer[2], and Chen Qian[3]

[1] Inria, ENS, CNRS, PSL, Paris, France
[2] TU Wien, Austria
[3] NTNU, Trondheim, Norway
first.last@{ens.fr,tuwien.ac.at,ntnu.no}

Abstract. Transferable e-cash is the most faithful digital analog of physical cash, as it allows users to transfer coins between them without interacting with the bank (or a "ledger"). Strong anonymity requirements and the need for mechanisms to trace illegal behavior (double-spending of coins) have made instantiating the concept notoriously hard. Baldimtsi et al. (PKC'15) have given a first instantiation, which relied on a powerful cryptographic primitive that made the scheme non-practical.
In this paper we first revisit the model for transferable e-cash, proposing simpler yet stronger security definitions and then give the first concrete instantiation of the primitive, basing it on bilinear groups, and analyze its concrete efficiency.

Keywords. transferable/offline e-cash, strong anonymity.

1 Introduction

Contrary to so-called "crypto"-currencies like Bitcoin [Nak08], one of the main properties that the predating cryptographic *e-cash* aimed to achieve was user anonymity. Introduced by Chaum [Cha83], the goal was to realize a digital analog of physical cash, which allows users to pay without revealing their identity; and there has been a long line of research since [CFN88, Bra93, CHL05, BCKL09, FHY13, CPST16, BPS19] (to cite only a few). The scenario for e-cash is simple: a bank issues electronic coins to users, who can then spend them with merchants, who in turn can deposit them at the bank to get their account credited. User privacy should be protected in that not even the bank can link the withdrawing of a coin to its spending.

The main difference to the physical world is that digital coins can easily be duplicated, and therefore a so-called "double-spending" of a coin must be prevented. This can be easily achieved when all actors are online and connected (as for cryptocurrencies), since every spending is registered and payees can simply refuse a coin that has already been spent. (There are other means to ensure this in "anonymous" cryptocurrencies like *Monero* [vS13] and *Zcash* [BCG+14], but ultimately they all crucially depend on users being connected when they accept a payment.)

When users are allowed to spend coins to other users (or merchants) without continuous connectivity, then double-spending cannot be prevented; however,

starting with [CFN88], ingenious methods have been devised that allow to reveal double-spenders' identities while guaranteeing the privacy of all honest users.

Transferable e-cash. In all traditional e-cash schemes, including such "offline" e-cash, once a coin is spent (transferred) after withdrawal, it must be deposited at the bank by the payee. A more powerful concept, and much more faithful to physical e-cash, is *transferable e-cash*, which allows users to re-transfer obtained coins, while at the same time remaining offline. The concept was first proposed by Okamoto and Ohta [OO89, OO91], but the constructions only guaranteed weak notions of anonymity. It was also shown [CP93] that coins need to grow in size with every transfer (since information about potential double-spenders needs to be encoded) and that unbounded adversaries can recognize coins they owned before.

While other schemes [Bla08, CGT08] only satisfy weak anonymity guarantees, Canard and Gouget [CG08] analyzed possible (stronger) anonymity notions and gave an instantiation in which a polynomial-time adversary cannot recognize coins that he has already owned. However, this can only hold if the bank is honest, as the bank must be able to link occurrences of the same coin in order to detect double-spending. They also give a scheme achieving this notion, which is however completely impractical (as at every transfer, the payer sends a proof of (a proof of $(\dots)$) a coin that she received earlier).

The first practical scheme was given by Fuchsbauer et al. [FPV09], but it makes unacceptable compromises elsewhere: when a double-spending is detected, all (even innocent) users up to the double-spender lose their anonymity. Blazy et al. [BCF+11] avoid this problem and propose an anonymous scheme, but they have to assume a trusted party (called the "judge") which can trace all coins in the system and has to actively intervene in order to identify double-spenders.

While the need for fully trusted actors was still unsatisfactory, the authors [BCF+11] gave an elegant construction using rerandomizable non-interactive zero-knowledge (NIZK) proofs [BCC+09] and commuting signatures [Fuc11]. In their scheme a coin consists of a signature by the bank and at every transfer the spender adds her own signature (which commits her to her spending). To achieve anonymity, these signatures are not given in the clear; coins are NIZK proofs of knowledge of signatures. Since the proofs can be rerandomized (that is, given a proof, one can produce a proof of the same statement that looks unrelated to the original one), coins can change appearance after every transfer, and so a user will not recognize a coin when she sees it again later.

Starting from Blazy et al.'s [BCF+11] framework, Baldimtsi et al. [BCFK15] give an instantiation which avoids the "judge" by using double-spending-tracing mechanisms from classical offline e-cash. They add "tags" to the coin that hide the identity of the owner of the coin, except when he spends the coin twice, then the bank can from two such tags compute the user's identity. Since users must include their signature in the coin during transfer, these represent irrefutable proof that a double-spending was committed.

The main drawback of their scheme is efficiency. They use the concept of *malleable signatures* [CKLM14], which are a generalization of digital signatures:

a signature on a message m can be transformed to a signature on any message $T(m)$ for any allowed transformation T. A coin is a malleable signature computed by the bank, which can be transformed by a user if and only if she correctly encodes her identity in a double-spending tag, adds an encryption (under the bank's public key) to it and randomizes all encryptions of previous tags.

Our contribution. Our contributions are two-fold:

Security model. We first revisit the formal model for transferable e-cash, departing from [BCFK15], whose model was a refined version of earlier ones. We start with giving a definition of correctness, which was lacking in previous works. Moreover, we exhibit attacks against users who follow the protocol, against which previous models did not protect:

- When a user receives a coin (that is, the protocol accepts the received coin), then in previous models there is no guarantee that this coin will be accepted by other (honest) users when transferred. An adversary could thus send a mal-formed coin to a user, which the latter accepts but can then not spend.
- There are no guarantees for a user against a malicious bank which at coin deposit refuses to credit the user's account (e.g., by claiming that the coin was invalid or had been double-spent). In our model, when the bank refuses a coin, it is forced to accuse a user of double-spending and provide a proof for this.

We moreover simplify the anonymity definitions, which in earlier version had been cluttered with numerous oracles the adversary has access to, and for which the intuitive notion that they were formalizing was hard to grasp. While our definitions are simpler, they are stronger in that they imply previous definitions (except for the previous notion of "spend-then-receive (StR) anonymity", whose version we argue is not relevant in practice).

We also show that the proof of StR anonymity of the scheme from [BCFK15] is flawed.

Instantiation. Our main contribution is an instantiation of transferable e-cash, which we prove satisfies our security model, and which is much more efficient than the only previous realization [BCFK15]. To do so, we depart from the use of malleable signatures, which due to their generality and strong security guarantees in the spirit of simulation-sound extractability result in very inefficient schemes.

Instead, we give a direct instantiation based on Groth-Sahai proofs [GS08], which are randomizable, structure-preserving signatures [AFG+10], which are compatible with GS proofs, and rerandomizable encryption satisfying RCCA-security [CKN03] (the corresponding variant of CCA security). While we use signature schemes from the literature [AGHO11, Fuc11], we construct a new RCCA-secure encryption scheme based on [LPQ17] that is tailored to our scheme. Finally, we reuse the (efficient) tags introduced in [BCFK15] to trace double-spenders.

Due to the existence of an omnipotent "judge", no such tags were required in [BCF+11]. Surprisingly, although we do not assume any active trusted parties, we

achieve a comparable efficiency, which we do by realizing that the full potential of these tags had not been leveraged in [BCFK15]: they had only served to encode a user's identity; but, as we show, they can at the same time be used to commit the user. This means that, contrary to all previous instantiations, we can omit the inclusion of signatures by the users in the coins, which makes them lighter. For an informal, yet more detailed overview of our scheme see Sect. 5.1.

2 Definition of transferable e-cash

2.1 Algorithms and protocols

An e-cash scheme is set up by running ParamGen and the bank generating its key pair via BKeyGen. The bank maintains a list of users $\mathcal{UL}$ and a list of deposited coins $\mathcal{DCL}$. Users run the protocol Register with the bank to obtain their secret key, and their public keys are added to $\mathcal{UL}$. With her secret key a user can run Withdraw with the bank to obtain coins, which she can transfer to others via the protocol Spend.

Spend is also used when a user deposits a coin at the bank. After receiving a coin, the bank runs CheckDS (for "**d**ouble-**s**pending") on it and the previously deposited coins in $\mathcal{DCL}$, which determines whether to accept the coin. If so, it is added to $\mathcal{DCL}$; if not (in case of double-spending), CheckDS returns the public key of the accused user and a proof Π, which can be verified using VfyGuilt.

$\mathsf{ParamGen}(1^\lambda)$, on input the security parameter λ in unary, outputs public parameters par, which are an implicit input to all of the following algorithms.

$\mathsf{BKeyGen}()$ is run by the bank $\mathcal{B}$ and outputs its public key $pk_\mathcal{B}$ and its secret key $sk_\mathcal{B} = (sk_\mathcal{W}, sk_\mathcal{D}, sk_\mathcal{CK})$, where $sk_\mathcal{W}$ is used to issue coins in Withdraw and to register users in Register; $sk_\mathcal{D}$ is used as the secret key of the receiver when coins are deposited via Spend; and $sk_\mathcal{CK}$ is used for CheckDS.

$\mathsf{Register}\langle\mathcal{B}(sk_\mathcal{W}), \mathcal{U}(pk_\mathcal{B})\rangle$ is a protocol between the bank and a user. The user obtains a secret key sk and the bank gets pk, which it adds to $\mathcal{UL}$. In case of error, they both obtain $\perp$.

$\mathsf{Withdraw}\langle\mathcal{B}(sk_\mathcal{W}), \mathcal{U}(sk_\mathcal{U}, pk_\mathcal{B})\rangle$ is run between the bank and a user, who outputs a coin c (or $\perp$), while the bank outputs ok (in which case it debits the user's account) or $\perp$.

$\mathsf{Spend}\langle\mathcal{U}(c, sk, pk_\mathcal{B}), \mathcal{U}'(sk', pk_\mathcal{B})\rangle$ is run between two users and lets $\mathcal{U}$ spend a coin c to $\mathcal{U}'$ (who could be the bank). $\mathcal{U}'$ outputs a coin c' (or $\perp$), while $\mathcal{U}$ outputs ok (or $\perp$).

$\mathsf{CheckDS}(sk_\mathcal{CK}, \mathcal{UL}, \mathcal{DCL}, c)$, run by the bank, takes as input its checking key, the lists of registered users $\mathcal{UL}$ and of deposited coins $\mathcal{DCL}$ and a coin c. It outputs an updated list $\mathcal{DCL}$ (when the coin is accepted) or a user public key $pk_\mathcal{U}$ and an incrimination proof Π.

$\mathsf{VfyGuilt}(pk_\mathcal{U}, \Pi)$ can be executed by anyone. It takes a user public key and an incrimination proof and returns 1 (acceptance of Π) or 0 (rejection).

Note that we define a transferable e-cash scheme as stateless, in that there is no state information shared between the algorithms. A withdrawn coin, whether it was the first or the n-th coin issues to a specific user, is always distributed the same. Moreover, a received coin will only depend on the spent coin (and not on other spent or received coins). Thus, the bank and the users need not store anything about past transactions for transfer; the coin itself must be sufficient.

In particular, the bank can separate withdrawing from depositing, in that CheckDS, used during deposit, need not be aware of the withdrawn coins.

2.2 Correctness properties

These properties were not stated in previous models. They are important in that they preclude schemes that satisfy security notions by not doing anything.

Let par be an output of $\mathsf{ParamGen}(1^\lambda)$ and $(sk_\mathcal{B} = (sk_\mathcal{W}, sk_\mathcal{D}, sk_{\mathcal{CK}}), pk_\mathcal{B})$ be output by $\mathsf{BKeyGen}(par)$. Then the following holds:

- none of the outputs is $\perp$;
- any execution of $\mathsf{Register}\langle\mathcal{B}(sk_\mathcal{W}),\mathcal{U}(pk_\mathcal{B})\rangle$ yields output pk for $\mathcal{B}$ and sk for $\mathcal{U}$.

Further, let sk and sk' be two user outputs of Register; then:

- any execution of $\mathsf{Withdraw}\langle\mathcal{B}(sk_\mathcal{W}),\mathcal{U}(sk, pk_\mathcal{B})\rangle$ yields ok for $\mathcal{B}$ and c for $\mathcal{U}$;
- in an execution of $\mathsf{Spend}\langle\mathcal{U}(c, sk, pk_\mathcal{B}),\mathcal{U}'(sk', pk_\mathcal{B})\rangle$, no party outputs $\perp$;
- $sk_\mathcal{D}$ works as a user secret key sk'.

(Correctness of CheckDS and VfyGuilt is implied by the security notion below.)

2.3 Security definitions

Global variables. In our security games, we store all information about users and their keys in the user list $\mathcal{UL}$. Its entries are of the form (pk_i, sk_i, uds_i), where uds_i indicates how many times user $\mathcal{U}_i$ has double-spent.

In the coin list $\mathcal{CL}$, we keep information about the coins created in the system. For each withdrawn or spent coin c, we store a tuple $(owner, c, cds, origin)$, where $owner$ stores the index i of the user who withdrew or received the coin (coins withdrawn or received by the adversary are not stored). We also include cds, which counts how often this *specific instance* of the coin has been spent. We set $origin$ to "$\mathcal{B}$" if the coin was issued by the honest bank and to "$\mathcal{A}$" if it originates from the adversary; if the coin was originally spent by the challenger itself, we store a pointer indicating which original coin this transferred coin corresponds to. Finally, we maintain a list of deposited coins $\mathcal{DCL}$.

Oracles. We now define oracles used in the security definitions, which differ depending on whether the adversary impersonates a corrupt bank or users. If during the oracle execution an algorithm fails (i.e., it outputs $\perp$) then the oracle also stops. Otherwise the call to the oracle is considered successful; a successful deposit oracle call must also not detect any double-spending.

Registration and corruption of users. The adversary can instruct the creation of honest users and either play the role of the bank during registration, or passively observe registration. It can moreover "spy" on users, meaning it can learn the user's secret key. This will strengthen yet simplify our anonymity games compared to [BCFK15], where once the adversary had learned the secret key of a user (by "corrupting" her), the user could not be a challenge user in the anonymity games anymore (yielding *selfless anonymity*, while we achieve *full* anonymity).

`BRegist`() plays the bank side of Register and interacts with $\mathcal{A}$. If successful, it adds $(pk, \bot, uds = 0)$ to $\mathcal{UL}$ (where uds is the number of double-spends).

`URegist`() plays the user side of the Register protocol when the bank is controlled by the adversary. Upon successful execution, it adds $(pk, sk, 0)$ to $\mathcal{UL}$.

`Regist`() plays both parties in the Register protocol and adds $(pk, sk, 0)$ to $\mathcal{UL}$.

`Spy`(i), for $i \leq |\mathcal{UL}|$, returns user i's secret key sk_i.

Withdrawal oracles. The adversary can either withdraw a coin from the bank, play the role of the bank, or passively observe a withdrawal.

`BWith`() plays the bank side of the Withdraw protocol. Coins withdrawn by $\mathcal{A}$ (and thus unknown to the experiment) are not added to the coin list $\mathcal{CL}$.

`UWith`(i) plays user i in Withdraw when the bank is controlled by the adversary. Upon obtaining a coin c, it adds $(owner = i, c, cds = 0, origin = \mathcal{A})$ to $\mathcal{CL}$.

`With`(i) simulates a Withdraw protocol execution playing both $\mathcal{B}$ and user i. It adds $(owner = i, c, cds = 0, origin = \mathcal{B})$ to $\mathcal{CL}$.

Spend and deposit oracles.

`Spd`(j) spends the coin from the j-th entry $(owner_j, c_j, cds_j, origin_j)$ in $\mathcal{CL}$ to $\mathcal{A}$, who could be impersonating a user, or the bank during a deposit. The oracle plays $\mathcal{U}$ in the Spend protocol with secret key sk_{owner_j}. It increments the coin spend counter cds_j by 1. If afterwards $cds_j > 1$, then the owner's double-spending counter uds_{owner_j} is incremented by 1.

`Rcv`(i) makes honest user i receive a coin from $\mathcal{A}$. The oracle plays $\mathcal{U}'$ with user i's secret key in the Spend protocol. It adds a new entry $(owner = i, c, cds = 0, origin = \mathcal{A})$ to $\mathcal{CL}$.

`S&R`(j, i) spends the j-th coin in $\mathcal{CL}$ to user i. It runs $(ok, c) \leftarrow \mathsf{Spend}\langle\mathcal{U}(c_j, sk_{owner_j}, pk_{\mathcal{B}}), \mathcal{U}'(sk_i, pk_{\mathcal{B}})\rangle$ and adds $(owner = i, c, cds = 0, pointer = j)$ to $\mathcal{CL}$. It increments the coin spend counter cds_j by 1. If afterwards $cds_j > 1$, then uds_{owner_j} is incremented by 1.

`BDepo`() lets $\mathcal{A}$ deposit a coin. It runs $\mathcal{U}'$ in Spend using the bank's secret key $sk_{\mathcal{D}}$ with the adversary playing $\mathcal{U}$. If successful, it runs CheckDS on the received coin and updates $\mathcal{DCL}$ accordingly; else it outputs a pair (pk, Π).

`Depo`(j), the honest deposit oracle, runs Spend between the owner of the j-th coin in $\mathcal{CL}$ and an honest bank. If successful, it increments cds_j by 1; if afterwards $cds_j > 1$, it also increments uds_{owner_j}. It runs CheckDS on the received coin and either updates $\mathcal{DCL}$ or returns a pair (pk, Π).

(Note that no oracle "UDepo" is required, since Spd lets the adversarial bank have an honest user deposit a coin.)

2.4 Economic properties

We distinguish two types of security properties of transferable e-cash schemes. Besides anonymity notions, economic properties ensure that neither the bank nor users will incur an economic loss when participating in the system.

The following property was not required in any previous security definition of transferable e-cash in the literature.

Soundness. If an honest user accepted a coin during a withdrawal or a transfer, then she is guaranteed that the coin will be accepted by others, either honest users when transferring, or the bank when depositing. The game is formalized in Fig. 1 where i_2 plays the role of the receiver of a spending or the bank. For convenience, we define probabilistic polynomial-time (PPT) adversaries $\mathcal{A}$ to be stateful in all our security games.

$\mathbf{Expt}^{\mathtt{sound}}_{\mathcal{A}}(\lambda)$:
$par \leftarrow \mathsf{ParamGen}(1^\lambda)$; $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
$(b, i_1, i_2) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy}}$
If $b = 0$ then run $\mathtt{UWith}(i_1)$ with $\mathcal{A}$
Else run $\mathtt{Rcv}(i_1)$ with $\mathcal{A}$
If this outputs $\perp$ then return 0
Run $\mathtt{S\&R}(1, i_2)$; if one party outputs $\perp$ then return 1
Return 0

Fig. 1. Game for *soundness* (protecting users from financial loss)

Definition 1 (Soundness). *A transferable e-cash system is* sound *if for any PPT $\mathcal{A}$, we have* $\mathbf{Adv}^{\mathtt{sound}}_{\mathcal{A}}(\lambda) := \Pr[\mathbf{Expt}^{\mathtt{sound}}_{\mathcal{A}}(\lambda) = 1]$ *is negligible in λ.*

Unforgeability. This notion covers both *unforgeability* and *user identification* from [BCFK15] (which were not consistent as we explain in Sect. 3.2). It protects the bank, ensuring that no (coalition of) users can spend more coins than the number of coins they withdrew.

Unforgeability also guarantees that whenever a coin is deposited and refused by CheckDS, the latter also returns the identity of a registered user, who is accused of double-spending. (*Exculpability*, below, ensures that no innocent user will be accused.) The game is formalized in Fig. 2 and lets the adversary impersonate all users.

Definition 2 (Unforgeability). *A transferable e-cash system is* unforgeable *if* $\mathbf{Adv}^{\mathtt{unforg}}_{\mathcal{A}}(\lambda) := \Pr[\mathbf{Expt}^{\mathtt{unforg}}_{\mathcal{A}}(\lambda) = 1]$ *is negligible in λ for any PPT $\mathcal{A}$.*

$\mathbf{Expt}^{\mathtt{unforg}}_{\mathcal{A}}(\lambda)$:
$par \leftarrow \mathsf{ParamGen}(1^\lambda)$; $(sk_{\mathcal{B}}, pk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}(par)$
$\mathcal{A}^{\mathtt{BRegist},\mathtt{BWith},\mathtt{BDepo}}(par, pk_{\mathcal{B}})$
If in a `BDepo` call, $\mathsf{CheckDS}$ does not return a coin list:
Return 1 if any of the following hold:
– $\mathsf{CheckDS}$ outputs $\perp$
– $\mathsf{CheckDS}$ outputs (pk, Π) and $\mathsf{VfyGuilt}(pk, \Pi) = 0$
– $\mathsf{CheckDS}$ outputs (pk, Π) and $pk \notin \mathcal{UL}$
Let q_W be the number of calls to `BWith`
If $q_W < |\mathcal{DCL}|$, then return 1
Return 0

Fig. 2. Game for *unforgeability* (protecting the bank from financial loss)

Exculpability. This notion, a.k.a. *non-frameability*, ensures that the bank, even when colluding with malicious users, cannot wrongly accuse an honest user of double-spending. Specifically, it guarantees that an adversarial bank cannot produce a double-spending proof Π^* that verifies for the public key of a user i^* that has never double-spent. The game is formalized as in Fig. 3.

$\mathbf{Expt}^{\mathtt{excul}}_{\mathcal{A}}(\lambda)$:
$par \leftarrow \mathsf{ParamGen}(1^\lambda)$; $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
$(i^*, \Pi^*) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy},\mathtt{UWith},\mathtt{Rcv},\mathtt{Spd},\mathtt{S\&R},\mathtt{UDepo}}(par)$
Return 1 if **all** of the following hold:
– $\mathsf{VfyGuilt}(pk_{i^*}, \Pi^*) = 1$
– There was no call $\mathtt{Spy}(i^*)$
– $uds_{i^*} = 0$
Return 0

Fig. 3. Game for *exculpability* (protecting honest users from accusation)

Definition 3 (Exculpability). *A transferable e-cash system is* exculpable *if* $\mathbf{Adv}^{\mathtt{excul}}_{\mathcal{A}}(\lambda) := \Pr[\mathbf{Expt}^{\mathtt{excul}}_{\mathcal{A}}(\lambda) = 1]$ *is negligible in* λ *for any PPT* $\mathcal{A}$.

2.5 Anonymity properties

Instead of following previous anonymity notions [BCF+11, BCFK15], we introduce new ones which clearly distinguish between the adversary's capabilities; in particular, whether it is able to detect double-spending. When the adversary impersonates the bank, we consider two cases: user anonymity and coin anonymity (and explain why this distinction is necessary).

As transferred coins necessarily grow in size [CP93], we can only guarantee indistinguishability of *comparable* coins. We therefore define $\mathtt{comp}(c_1, c_2) = 1$ iff $\mathtt{size}(c_1) = \mathtt{size}(c_2)$, where $\mathtt{size}(c) = 1$ after c was withdrawn and it increases by 1 after each transfer.

$\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},b}(\lambda)$:
$par \leftarrow \mathsf{ParamGen}(1^\lambda)$
$pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
$i_0^{(0)} \leftarrow \mathcal{A}^{\texttt{URegist,Spy}}$; run $\texttt{UWith}(i_0^{(0)})$ with $\mathcal{A}$
$i_0^{(1)} \leftarrow \mathcal{A}^{\texttt{URegist,Spy}}$; run $\texttt{UWith}(i_0^{(1)})$ with $\mathcal{A}$
$\big((i_1^{(0)},\ldots,i_{k_0}^{(0)}),(i_1^{(1)},\ldots,i_{k_1}^{(1)})\big) \leftarrow \mathcal{A}^{\texttt{URegist,Spy}}$
If $k_0 \neq k_1$ then return 0
For $j = 1,\ldots,k_0$:
  Run $\texttt{S\&R}\big(2j-1, i_j^{(0)}\big)$
  Run $\texttt{S\&R}\big(2j, i_j^{(1)}\big)$
Run $\texttt{Spd}(2k_0+1+b)$ with $\mathcal{A}$
Run $\texttt{Spd}(2k_0+2-b)$ with $\mathcal{A}$
$b^* \leftarrow \mathcal{A}$
Return b^*

$\mathbf{Expt}^{\texttt{u-an}}_{\mathcal{A},b}(\lambda)$:
$par \leftarrow \mathsf{ParamGen}(1^\lambda)$
$pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
$(i_0^{(0)}, i_0^{(1)}) \leftarrow \mathcal{A}^{\texttt{URegist,Spy}}$
Run $\texttt{Rcv}(i_b)$ with $\mathcal{A}$
$\big((i_1^{(0)},\ldots,i_{k_0}^{(0)}),(i_1^{(1)},\ldots,i_{k_1}^{(1)})\big) \leftarrow \mathcal{A}^{\texttt{URegist,Spy}}$
If $k_0 \neq k_1$ then return 0
For $j = 1,\ldots,k_0$:
  Run $\texttt{S\&R}\big(j, i_j^{(b)}\big)$
Run $\texttt{Spd}(k_0+1)$ with $\mathcal{A}$
$b^* \leftarrow \mathcal{A}$
Return b^*

Fig. 4. Games for *coin* and *user anonymity* (protecting users from a malicious bank)

Coin anonymity. This notion is closest to (and implies) the anonymity notion of classical e-cash: an adversary, who also impersonates the bank, issues two coins to the challenger and when she later receives them (via a deposit in classical e-cash), she should not be able to associate them to their issuances. In transferable e-cash, we allow the adversary to determine two series of honest users via which the coins are respectively transfered before being given back to the adversary.

The experiment is specified on the left of Fig. 4: users $i_0^{(0)}$ and $i_0^{(1)}$ withdraw a coin from the adversarial bank, user $i_0^{(0)}$ passes it to $i_1^{(0)}$, who passes it to $i_2^{(0)}$, etc., In the end, the last users of the two chains spend the coins to the adversary, but the order in which this happens depends on a bit b that parametrizes the game, and which the adversary must decide.

User anonymity. Coin anonymity required that users who transfer the coin are honest. If one of the users through which the coin passes colluded with the bank, there would be a trivial attack: after receiving the two challenge coins, the bank simulates the deposit of one of them and the deposit of the coin intercepted by the colluding user. If a double-spending is detected, it knows that the received coin corresponds to the sequence of users which the colluder was part of.

Since double-spending detection is an essential feature of e-cash, attacks of this kind are impossible to prevent. However, we still want to guarantee that, while the bank can trace coins, the involved *users* remain anonymous. We formalize this in the game on the right of Fig. 4, where, in contrast to coin anonymity, there is only one coin and the adversary must distinguish the sequence of users through which the coin passes before returning to her. In contrast to coin anonymity, we now allow the coin to already have some "history", rather than being freshly withdrawn.

Coin transparency. This is in some sense the strongest anonymity notion and it implies that a user that transfers a coin cannot recognize it if at some point she receives it again. Note that this notion (as well as *coin anonymity*) is not

even achieved by physical cash, as banknotes can be marked by users (or the bank). As the bank can necessarily trace coins (for double-spending detection), it is assumed to be honest for this notion. Actually, only the detection key $sk_{\mathcal{CK}}$ must remain hidden from the adversary, while $sk_{\mathcal{W}}$ and $sk_{\mathcal{D}}$ can be given.

The game formalizing this notion, specified in Fig. 5, is analogous to coin anonymity, except that the challenge coins are not freshly withdrawn; instead, the adversary spends two coins of its choice to users of its choice, both are passed through a sequence of users of the adversary's choice and one of them is returned to the adversary.

There is another trivial attack that we need to exclude: the adversary could deposit the coin that is returned to him and one, say the first, of the coins he initially transfered to an honest user. Now if the deposit does not succeed because of double-spending, the adversary knows that it was the first coin that was returned to him. Again, this attack is unavoidable due to the necessity of double-spending detection. It is a design choice that lies outside of our model to implement sufficient deterrence from double-spending, so it would exceed the utility of breaking anonymity.

This is the reason why the game aborts if the adversary deposits twice a coin from the set of "challenge coins" (consisting of the two coins the adversary transfers and the one it receives). The variable *ctr* counts how many times a coin from this set was deposited. Note also that because $\mathcal{A}$ has $sk_{\mathcal{W}}$, and can therefore create unregistered users, we do not consider $\mathcal{UL}$ in this game.

Definition 4 (Anonymity). *For* $\mathtt{x} \in \{\mathtt{c\text{-}an}, \mathtt{u\text{-}an}, \mathtt{c\text{-}tr}\}$ *a transferable e-cash scheme satisfies* $\mathtt{x}$ *if* $\mathbf{Adv}^{\mathtt{x}}_{\mathcal{A}}(\lambda) := \Pr[\mathbf{Expt}^{\mathtt{x}}_{\mathcal{A},1}(\lambda) = 1] - \Pr[\mathbf{Expt}^{\mathtt{x}}_{\mathcal{A},0}(\lambda) = 1]$ *is negligible in* λ *for any PPT adversary* $\mathcal{A}$*.*

3 Comparison with previous work

3.1 Model comparison

In order to justify our new model, we start with discussing a security vulnerability of the previous model [BCFK15].

Issues with economical notions. As already pointed out in Sect. 2.2, the *correctness properties* were missing in previous models.

No soundness guarantees. In none of the previous models was there a security notion that guaranteed that an honest user could successfully transfer a coin to another honest user or the bank, even if the coin was obtained regularly.

Fuzzy definition of "unsuccessful deposit". Previous models defined a protocol called "Deposit", which we separated into an interactive (Spend) and a static part (CheckDS). In their definition of unforgeability, the authors [BCFK15] use the concept of "successful deposit", which was not clearly defined, since an "unsuccessful deposit" could mean one of the following:

$\mathbf{Expt}^{\text{c-tr}}_{\mathcal{A},b}(\lambda)$:

$par \leftarrow \mathsf{ParamGen}(1^\lambda)$; $((sk_{\mathcal{W}}, sk_{\mathcal{D}}, sk_{\mathcal{CK}}), pk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}(par)$
$\mathcal{DCL}' \leftarrow \emptyset$ // *lists the challenge coins*
$ctr \leftarrow 0$ // *counts how often a challenge coin was deposited*
$i^{(0)} \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
// $\mathtt{BDepo}'$ *uses* $\mathsf{CheckDS}'(\cdot,\cdot,\cdot,\cdot,\mathcal{DCL}')$ *(see below) instead of* $\mathsf{CheckDS}$
Run $\mathtt{Rcv}(i^{(0)})$ with $\mathcal{A}$; let c_0 be the received coin stored in $\mathcal{CL}[1]$
$x_0 \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_0)$
If $x_0 = \perp$ then $ctr \leftarrow ctr + 1$ //c_0 *had been deposited*
$\mathcal{DCL}' \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \emptyset, c_0)$ //*add* c_0 *to list of challenge coins*
$i^{(1)} \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo},\mathtt{Spy}}$
Run $\mathtt{Rcv}(i^{(1)})$ with $\mathcal{A}$; let c_1 be the received coin stored in $\mathcal{CL}[2]$
$x_1 \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_1)$
If $x_1 = \perp$ then $ctr \leftarrow ctr + 1$ //c_1 *had been deposited*
If $\mathtt{comp}(c_0, c_1) \neq 1$ then abort
$x_2 \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}', c_1)$ //*add* c_1 *to list of challenge coins*
If $x_2 \neq \perp$ then $\mathcal{DCL}' \leftarrow x_2$ // (c_1 *could be a double-spending of* c_0)
$\big((i_1^{(0)}, \ldots, i_{k_0}^{(0)}), (i_1^{(1)}, \ldots, i_{k_1}^{(1)})\big) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}$
If $k_0 \neq k_1$ then abort
If $(k_b \neq 0)$ then run $\mathtt{S\&R}\big(b+1, i_1^{(b)}\big)$ // *spend coin* c_b *to user* $i_1^{(b)}$ ...
For $j = 2, \ldots, k_0$: // ... *the received coin is placed in* $\mathcal{CL}[3]$
Run $\mathtt{S\&R}\big(j+1, i_j^{(b)}\big)$ // *spend coins consecutively*
Run $\mathtt{Spd}(k_0 + 2)$ with $\mathcal{A}$ // *and transfer it back to* $\mathcal{A}$
$b^* \leftarrow \mathcal{A}^{\mathtt{BDepo}'}$
Return b^*

$\mathsf{CheckDS}'(sk_{\mathcal{CK}}, \mathcal{UL}, \mathcal{DCL}, c, \mathcal{DCL}')$: // used by $\mathtt{BDepo}'$
$x \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}', c)$
If $x = \perp$: // *the deposited coin* c *is a double-spending of* c_0 *or* c_1
$ctr \leftarrow ctr + 1$
If $ctr > 1$ then abort
Output $\mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}, c)$

Fig. 5. Game for *coin transparency* (protecting users from malicious users)

- The bank detects a double-spending and provides a proof accusing the cheater (who could be different from the depositer).
- The user did not follow the protocol (e.g., by sending a malformed coin), in which case we cannot expect a proof of guilt from the bank.
- The user followed the protocol but using a coin that was double-spent (either earlier or during deposit); however, the bank does not obtain a valid proof of guilt and outputs $\perp$.

Our interpretation of the definition in [BCFK15] is that it does not distinguish the second and the third case. This is an issue, as the second case cannot be avoided (and must be dealt with outside the model, e.g. by having users sign their messages). But the third case *should* be avoided so the bank does not lose

money without being able to accuse the cheater. This is now guaranteed by our unforgeability notion in Def. 2.

Simplification of anonymity definitions. We believe that our notions are more intuitive and simpler (e.g. by reducing the number of oracles of previous work). Our notions imply prior notions from the literature: we can prove that the existence of an adversary in a game from a prior notion implies the existence of an adversary in one of our games. (The general idea is to simulate most of the oracles using the secret keys of the bank or users, which in our notions can be obtained via the `Spy` oracle.) In particular, the implications are the following:

$$\texttt{c-an} \Rightarrow \texttt{OtR-fa} \qquad \text{and} \qquad \texttt{u-an} \Rightarrow \texttt{StR*-fa}$$

where `OtR-fa` is *observe-then-receive full anonymity* [CG08, BCF+11, BCFK15] and `StR*-fa` is a variant of *spend-then-receive full anonymity* from [BCFK15].

The earlier notion `StR-fa` [CG08, BCF+11] is similar to our coin-transparency `c-tr`, with the following differences: in `StR-fa`, when the adversary deposits a coin, the bank provides a guilt proof when it can; and `StR-fa` lets the adversary obtain user secret keys. Coin-transparency would imply `StR-fa` if CheckDS replaced its argument $\mathcal{UL}$ by $\emptyset$. This change is justified since (in both `StR-fa` and `c-tr`) the adversary can create unregistered users (using $sk_{\mathcal{W}}$), and thus CheckDS could return $\perp$ because it cannot accuse anyone in $\mathcal{UL}$.

Moreover, no previous scheme, including the one in [BCFK15] achieves `StR-fa`, as we explain next.

3.2 An error in a proof in BCFK15

The authors of [BCFK15] claim that their scheme satisfies `StR-fa` as defined in [BCF+11] (after having discovered an error in the `StR-fa` proof of the scheme of that paper). To achieve this anonymity notion (the most difficult one, as they notice), they use malleable signatures, a primitive providing a strong security level: it guarantees that whenever the adversary, after obtaining simulated signatures, outputs a valid message/signature pair (m, σ), it must have derived the pair from received signatures. Formally, there exists an extractor that can extract a transformation from σ that links m to the messages on which the adversary queried signatures.

However, in the definition of `StR-fa` [BCF+11] the adversary receives $sk_{\mathcal{W}}$ (as in our `c-tr` notion), which in their instantiation [BCFK15] contains the signing key for the malleable signatures. Using this, the adversary can compute a fresh signature from which no extractor can recover a transformation explaining the signed message. Our scheme, which we prove satisfies `c-tr`, can therefore be seen as the first to satisfy the "spirit" of `StR-fa`, which is captured by our notion `c-tr`.

4 Primitives used in our construction

4.1 Bilinear groups

The building blocks of our scheme will be defined over a (Type-3, i.e., asymmetric) bilinear group, which is a tuple $Gr = (p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g, \hat{g})$, where $\mathbb{G}, \hat{\mathbb{G}}$ and $\mathbb{G}_T$ are groups of prime order p; $\langle g \rangle = \mathbb{G}$, $\langle \hat{g} \rangle = \hat{\mathbb{G}}$, and $e\colon \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ is a bilinear map (i.e., for all $a, b \in \mathbb{Z}_p$: $e(g^a, \hat{g}^b) = e(g, \hat{g})^{ab}$) so that $e(g, \hat{g})$ generates $\mathbb{G}_T$. We assume that the groups are discrete-log-hard and other computational assumptions (DDH, CDH, SXDH, etc. defined in Appendix D) hold as well. We assume that there exists an algorithm GrGen that, on input the security parameter λ in unary, outputs the description of a bilinear group with $p \geq 2^{\lambda-1}$.

4.2 Randomizable proofs of knowledge and signatures

Commit-and-prove proof systems. As coins must be unforgeable, at their core lie digital signatures. To achieve anonymity, these must be hidden, which can be achieved via non-interactive zero-knowledge (NIZK) proofs of knowledge; if these proofs are *re-randomizable*, then they can not even be recognized by a past owner. We will use Groth-Sahai NIZK proofs [GS08], which are randomizable [FP09, BCC+09] and include commitments to the witnesses.

We let $\mathcal{V}$ be set of values that can be committed, $\mathcal{C}$ be the set of commitments, $\mathcal{R}$ the randomness space and $\mathcal{E}$ the set of equations (containing equality) whose satisfiability can be proved. We assume that $\mathcal{V}$ and $\mathcal{R}$ are groups. We will use an extractable commitment scheme, which consists of the following algorithms:

$\mathsf{C.Setup}(Gr)$ takes as input a description of a bilinear group and returns a commitment key ck, which implicitly defines the sets $\mathcal{V}, \mathcal{C}, \mathcal{R}$ and $\mathcal{E}$.
$\mathsf{C.ExSetup}(Gr)$ returns an extraction key xk in addition to a commitment key ck.
$\mathsf{C.SmSetup}(Gr)$ returns a commitment key ck and a simulation trapdoor td.
$\mathsf{C.Cm}(ck, v, \rho)$, on input a key ck, a value $v \in \mathcal{V}$ and randomness $\rho \in \mathcal{R}$, returns a commitment in $\mathcal{C}$.
$\mathsf{C.ZCm}(ck, \rho)$, used when simulating proofs, is defined as $\mathsf{C.Cm}(ck, 0_{\mathcal{V}}, \rho)$.
$\mathsf{C.RdCm}(ck, c, \rho)$ randomizes a commitment c to a fresh c' using randomness ρ.
$\mathsf{C.Extr}(xk, c)$, on input extraction key xk and a commitment c, outputs a value in $\mathcal{V}$. (This is the only algorithm that might not be polynomial-time.)

We extend $\mathsf{C.Cm}$ to vectors in $\mathcal{V}^n$: for $M = (v_1, \ldots, v_n)$ and $\rho = (\rho_1, \ldots, \rho_n)$ we define $\mathsf{C.Cm}(ck, M, \rho) := \big(\mathsf{C.Cm}(ck, v_1, \rho_1), \ldots, \mathsf{C.Cm}(ck, v_n, \rho_n)\big)$ and likewise $\mathsf{C.Extr}(xk, (c_1, \ldots, c_n)) := \big(\mathsf{C.Extr}(xk, c_1), \ldots, \mathsf{C.Extr}(xk, c_n)\big)$.

We now define a NIZK proof system that proves that committed values satisfy given equations from $\mathcal{E}$. Given a proof for commitments, the proof can be adapted to a randomization (via $\mathsf{C.RdCm}$) of the commitments using $\mathsf{C.AdptPrf}$.

$\mathsf{C.Prv}(ck, E, (v_1, \rho_1), \ldots, (v_n, \rho_n))$, on input a key ck, a set of equations $E \subset \mathcal{E}$, values $(v_1, \ldots, v_n)$ and randomness $(\rho_1, \ldots, \rho_n)$, outputs a proof π.

$\mathsf{C.Verify}(ck, E, c_1, \ldots, c_n, \pi)$, on input a commitment key ck, a set of equations in $\mathcal{E}$, a commitment vector $(c_1, \ldots, c_n)$, and a proof π, outputs a bit b.

$\mathsf{C.AdptPrf}(ck, E, c_1, \rho_1, \ldots, c_n, \rho_n, \pi)$, on input a set of equations, commitments $(c_1, \ldots, c_n)$, randomness $(\rho_1, \ldots, \rho_n)$ and a proof π, outputs a proof π'.

$\mathsf{C.SmPrv}(td, E, \rho_1, \ldots, \rho_n)$, on input the simulation trapdoor, a set of equations E with n variables and randomness $(\rho_1, \ldots, \rho_n)$, outputs a proof π.

$\mathcal{M}$-structure-preserving signatures. To prove knowledge of signatures, we require a scheme that is compatible with Groth-Sahai proofs [AFG+10].

$\mathsf{S.Setup}(Gr)$, on input the bilinear group description, outputs signature parameters par_S, defining a message space $\mathcal{M}$. We require $\mathcal{M} \subseteq \mathcal{V}^n$ for some n.

$\mathsf{S.KeyGen}(par_S)$, on input the parameters par_S, outputs a signing key and a verification key (sk, vk). We require that vk is composed of values in $\mathcal{V}$.

$\mathsf{S.Sign}(sk, M)$, on input a signing key sk and a message $M \in \mathcal{M}$, outputs a signature Σ. We require that Σ is composed of values in $\mathcal{V}$.

$\mathsf{S.Verify}(vk, M, \Sigma)$, on input a verification key vk, a message M and a signature Σ, outputs a bit b. We require that $\mathsf{S.Verify}$ proceeds by evaluating equations from $\mathcal{E}$ (which we denote by $E_{\mathsf{S.Verify}(\cdot,\cdot,\cdot)}$).

$\mathcal{M}$-commuting signatures. As in a previous construction of transferable e-cash [BCF+11], we will use commuting signatures [Fuc11], which let the signer, given a commitment to a message, produce a commitment to a signature on that message, together with a proof, via the following functionality:

$\mathsf{SigCm}(ck, sk, c)$, given a signing key sk and a commitment c of a message $M \in \mathcal{M}$, outputs a committed signature c_Σ and a proof π that the signature in c_Σ is valid on the value in c, i.e., the committed values satisfy $\mathsf{S.Verify}(vk, \cdot, \cdot)$.

$\mathsf{SmSigCm}(xk, vk, c, \Sigma)$, on input the extraction key xk, a verification key vk, a commitment c and a signature Σ, outputs a committed signature c_Σ and a proof π of validity for c_Σ and c (the key xk is needed to compute π for c).

Correctness and soundness properties. We require the following properties of commitments, proofs and signatures, when the setup algorithms are run on any output $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$ for any $\lambda \in \mathbb{N}$:

Perfectly binding commitments: $\mathsf{C.Setup}$ and the first output of $\mathsf{C.ExSetup}$ are distributed equivalently. Let $(ck, xk) \leftarrow \mathsf{C.ExSetup}$; then for every $c \in \mathcal{C}$ there exists exactly one $v \in \mathcal{V}$ such that $c = \mathsf{C.Cm}(ck, v, \rho)$ for some $\rho \in \mathcal{R}$. Moreover, $\mathsf{C.Extr}(xk, c)$ extracts that value v.

$\mathcal{V}'$-extractability: We require that committed values from a subset $\mathcal{V}' \subset \mathcal{V}$ can be efficiently extracted. Let $(ck, xk) \leftarrow \mathsf{C.ExSetup}$; then $\mathsf{C.Extr}(xk, \cdot)$ is efficient on all values $c = \mathsf{C.Cm}(ck, v, \rho)$ for any $v \in \mathcal{V}'$ and $\rho \in \mathcal{R}$.

Proof completeness: Let $ck \leftarrow \mathsf{C.Setup}$; then for all $(v_1, \ldots, v_n) \in \mathcal{V}^n$ satisfying $E \subset \mathcal{E}$, and $(\rho_1, \ldots, \rho_n) \in \mathcal{R}^n$ and $\pi \leftarrow \mathsf{C.Prv}(ck, E, (v_1, \rho_1), \ldots, (v_n, \rho_n))$ we have $\mathsf{C.Verify}(ck, E, \mathsf{C.Cm}(ck, v_1, \rho_1), \ldots, \mathsf{C.Cm}(ck, v_n, \rho_n), \pi) = 1$.

Proof soundness: Let $(ck, xk) \leftarrow \mathsf{C.ExSetup}$, $E \subset \mathcal{E}$, and $(c_1, \ldots, c_n) \in \mathcal{C}^n$. If $\mathsf{C.Verify}(ck, E, c_1, \ldots, c_n, \pi) = 1$ for some π, then letting $v_i := \mathsf{C.Extr}(xk, c_i)$, for all i, we have that $(v_1, \ldots, v_n)$ satisfy E.

Randomizability: Let $ck \leftarrow \mathsf{C.Setup}$, and $E \subset \mathcal{E}$; then for all $(v_1, \ldots, v_n) \in \mathcal{V}^n$ that satisfy E and $\rho_1, \rho'_1, \ldots, \rho_n, \rho'_n \in \mathcal{R}$ the following two are distributed equivalently:

$$\big(\mathsf{C.RdCm}(\mathsf{C.Cm}(ck, v_1, \rho_1), \rho'_1), \ldots, \mathsf{C.RdCm}(\mathsf{C.Cm}(ck, v_n, \rho_n), \rho'_n), \\ \mathsf{C.AdptPrf}(ck, \mathsf{C.Prv}(ck, E, (v_1, \rho_1), \ldots, (v_n, \rho_n))), \rho'_1, \ldots, \rho'_n\big) \text{ and}$$

$$\big(\mathsf{C.Cm}(ck, v_1, \rho_1 + \rho'_1), \ldots, \mathsf{C.Cm}(ck, v_n, \rho_n + \rho'_n), \\ \mathsf{C.Prv}(ck, E, (v_1, \rho_1 + \rho'_1), \ldots, (v_n, \rho_n + \rho'_n))\big)$$

Signature correctness: Let $(sk, vk) \leftarrow \mathsf{S.KeyGen}(\mathsf{S.Setup})$ and $M \in \mathcal{M}$; then we have $\mathsf{S.Verify}(vk, M, \mathsf{S.Sign}(sk, M)) = 1$.

Correctness of signing committed messages: Let $(ck, xk) \leftarrow \mathsf{C.ExSetup}$ and let $(sk, vk) \leftarrow \mathsf{S.KeyGen}(\mathsf{S.Setup})$, and $M \in \mathcal{M}$; if $\rho, \rho' \xleftarrow{\$} \mathcal{R}$, then the following three are distributed equivalently:

$$\big(\mathsf{C.Cm}\big(ck, \mathsf{S.Sign}(sk, M), \rho'\big),\ \mathsf{C.Prv}\big(ck, E_{\mathsf{S.Verify}(vk,\cdot,\cdot)}, (M, \rho), (\Sigma, \rho')\big)\big) \text{ and}$$
$$\mathsf{SigCm}\big(ck, sk, \mathsf{C.Cm}(ck, M, \rho)\big) \text{ and}$$
$$\mathsf{SmSigCm}\big(xk, vk, \mathsf{C.Cm}(ck, M, \rho), \mathsf{S.Sign}(sk, M)\big)$$

The first equality also holds for $ck \leftarrow \mathsf{C.Setup}$, since it is distributed like ck output by $\mathsf{C.ExSetup}$.

Security properties

Mode indistinguishability: Let $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$; then the outputs of $\mathsf{C.Setup}(Gr)$ and the first output of $\mathsf{C.SmSetup}(Gr)$ are computationally indistinguishable.

Perfect zero-knowledge in hiding mode: Let $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$, $E \subset \mathcal{E}$ and $v_1, \ldots, v_n \in \mathcal{V}$ such that $E(v_1, \ldots, v_n) = 1$. For $\rho_1, \ldots, \rho_n \xleftarrow{\$} \mathcal{R}$ the following are distributed equivalently:

$$\big(\mathsf{C.Cm}(ck, v_1, \rho_1), \ldots, \mathsf{C.Cm}(ck, v_n, \rho_n), \mathsf{C.Prv}\big(ck, E, (v_1, \rho_1), \ldots, (v_n, \rho_n)\big)\big) \\ \text{and } \big(\mathsf{C.ZCm}(ck, \rho_1), \ldots, \mathsf{C.ZCm}(ck, \rho_n), \mathsf{C.SmPrv}\big(td, E, \rho_1, \ldots, \rho_n\big)\big)$$

Signature unforgeability (under chosen message attack): No PPT adversary that is given vk output by $\mathsf{S.KeyGen}$ and an oracle for adaptive signing queries on messages $M_1, M_2, \ldots$ of its choice can output a pair (M, Σ), such that $\mathsf{S.Verify}(vk, M, \Sigma) = 1$ and $M \notin \{M_1, M_2, \ldots\}$.

4.3 Rerandomizable encryption schemes

In order to trace double-spenders, some information must be retrievable from the coin by the bank. For anonymity, we encrypt this information. Since coins must change appearance in order to achieve coin transparency (Def. 4), we use rerandomizable encryption. In our e-cash scheme we will prove consistency of encrypted messages with values used elsewhere, and to produce such a proof, knowledge of *parts* of the randomness is required; we therefore make this an explicit input of some algorithms, which thus are still probabilistic.

A rerandomizable encryption scheme E consists of 4 poly.-time algorithms:

$\mathsf{E.KeyGen}(Gr)$, on input the group description, outputs an encryption key ek and a corresponding decryption key dk.
$\mathsf{E.Enc}(ek, M, \nu)$ is probabilistic and on input an encryption key ek, a message M and (partial) randomness ν outputs a ciphertext.
$\mathsf{E.ReRand}(ek, C, \nu')$, on input an encryption key, a ciphertext and (partial) randomness, outputs a new ciphertext. If no randomness is explicitly given to $\mathsf{E.Enc}$ or $\mathsf{E.ReRand}$ then it is assumed to be chosen uniformly.
$\mathsf{E.Dec}(dk, C)$, on input a decryption key and a ciphertext, outputs either a message or $\perp$ indicating an error.

In order to prove statements about encrypted messages, we add two functionalities: $\mathsf{E.Verify}$ lets one check that a ciphertext encrypts a given message M, for which it is also given partial randomness ν. This will allow us to prove that a commitment c_M and a ciphertext C contain the same message. For this, we require that the equations defining $\mathsf{E.Verify}$ are in the set $\mathcal{E}$ supported by $\mathsf{C.Prv}$.

This lets us define an equality proof $\tilde{\pi} = (\pi, c_\nu)$, where c_ν is a commitment of the randomness ν, and π proves that the values in c_M and c_ν verify the equations $\mathsf{E.Verify}(ek, \cdot, \cdot, C)$. To support rerandomization of ciphertexts, we define a functionality $\mathsf{E.AdptPrf}$, which adapts a proof (π, c_ν) to a rerandomization.

$\mathsf{E.Verify}(ek, M, \nu, C)$, on input an encryption key, a message, randomness and a ciphertext, outputs a bit.
$\mathsf{E.AdptPrf}(ck, ek, c_M, C, \tilde{\pi} = (\pi, c_\nu), \nu')$, a probabilistic algorithm which, on input a commitment key, an encryption key, a commitment, a ciphertext, an equality proof (i.e., a proof and a commitment) and randomness, outputs a new equality proof (π', c'_ν).

Correctness properties. We require the scheme to satisfy the following correctness properties for all key pairs $(ek, dk) \leftarrow \mathsf{E.KeyGen}(Gr)$ for $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$:

- For all $M \in \mathcal{M}$ and randomness ν we have: $\mathsf{E.Enc}(ek, M, \nu) = C$ if and only if $\mathsf{E.Verify}(ek, M, \nu, C) = 1$.
- For all $M \in \mathcal{M}$ and ν: $\mathsf{E.Verify}(ek, M, \nu, C) = 1$ implies $\mathsf{E.Dec}(dk, C) = M$. (These two notions imply the standard correctness notion.)
- For all $M \in \mathcal{M}$ and randomness ν, ν', if $C \leftarrow \mathsf{E.Enc}(ek, M, \nu)$ then the following are equally distributed: $\mathsf{E.ReRand}(ek, C, \nu')$ and $\mathsf{E.Enc}(ek, M, \nu + \nu')$.

– For all $ck \leftarrow \mathsf{C.Setup}$, all $(ek, dk) \leftarrow \mathsf{E.KeyGen}$, $M \in \mathcal{M}$ and randomness $\nu, \nu', \rho_M, \rho_\nu$, if we let

$$c_M \leftarrow \mathsf{C.Cm}(ck, M, \rho_M) \quad C \leftarrow \mathsf{E.Enc}(ek, M, \nu)$$
$$c_\nu \leftarrow \mathsf{C.Cm}(ck, \nu, \rho_\nu) \quad \pi \leftarrow \mathsf{C.Prv}\big(ck, \mathsf{E.Verify}(ek, \cdot, \cdot, C), (M, \rho_M), (\nu, \rho_\nu)\big)$$

then the following are equivalently distributed:

$$\mathsf{E.AdptPrf}\big(ck, ek, c_M, \mathsf{ReRand}(pk, C, \nu'), (\pi, c_\nu), \nu'\big) \quad \text{and}$$
$$\big(\mathsf{C.Prv}(ck, \mathsf{E.Verify}(ek, \cdot, \cdot, \mathsf{ReRand}(ek, C, \nu')), (M, \rho_M), (\nu + \nu', \rho_\nu)), c_\nu\big)$$

Security properties. We require two properties from rerandomizable encryption: the first one is the standard (strongest possible) variant of CCA security; the second one is a new notion, which is easier to achieve.

Replayable-CCA (RCCA) security. We use the definition from Canetti et al. [CKN03], formalized in Fig. 6.

$\mathbf{Expt}^{\mathtt{RCCA}}_{\mathcal{A},b}(\lambda)$:
$(ek, dk) \leftarrow \mathsf{E.KeyGen}(1^\lambda)$
$(m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{E.Dec}(dk,\cdot)}(ek)$
$C \leftarrow \mathsf{E.Enc}(ek, m_b)$
$b' \leftarrow \mathcal{A}^{\mathtt{GDec}(\cdot)}(C)$
Return b'.

$\mathtt{GDec}(C)$:
$m \leftarrow \mathsf{E.Dec}(dk, C)$
If $m \notin \{m_0, m_1\}$
  Return m
Else return `replay`

$\mathbf{Expt}^{\mathtt{IACR}}_{\mathcal{A},b}(\lambda)$:
$(ek, dk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
$(C_0, C_1) \leftarrow \mathcal{A}(ek)$
$C \leftarrow \mathsf{E.ReRand}(ek, C_b)$
$b' \leftarrow \mathcal{A}(ek, C)$
Return b'

Fig. 6. Security games for rerandomizable encryption schemes

Indistinguishability of adversarially chosen and randomized ciphertexts (IACR). An adversary that is given a public key, chooses two ciphertexts and is then given the randomization of one of them cannot, except with a negligible advantage, distinguish which one it was given. The game is formalized in Fig. 6.

Definition 5. *For* $\mathtt{x} \in \{\mathtt{RCCA}, \mathtt{IACR}\}$*, a rerandomizable encryption scheme is* x-secure *if* $\Pr[\mathbf{Expt}^{\mathtt{x}}_{\mathcal{A},1}(\lambda) = 1] - \Pr[\mathbf{Expt}^{\mathtt{x}}_{\mathcal{A},0}(\lambda) = 1]$ *is negligible in* λ *for any PPT* $\mathcal{A}$*.*

4.4 Double-spending tag schemes

Our e-cash scheme will follow earlier approaches [BCFK15], where the bank represents a coin in terms of its *serial number* $sn = sn_0 \| \ldots \| sn_k$, which grows with every transfer. In addition, a coin contains a tag $tag = tag_1 \| \ldots \| tag_k$, which enables tracing of double-spenders. The part sn_i is chosen by a user when she receives the coin, while the tag tag_i is computed by the sender as a function of sn_{i-1}, sn_i and her secret key.

Baldimtsi et al. [BCFK15] show how to construct such tags so they perfectly hide user identities, except when a user computes two tags with the same sn_{i-1} but different values sn_i, in which case her identity can be computed from the two tags. Note that this precisely corresponds to double-spending the coin that ends in sn_{i-1} to two users that choose different values for sn_i when receiving it.

We use the tags from [BCFK15], which we first formally define, and then show that its full potential had not been leveraged yet: in particular, we realize that the tag can also be used as method for users to *authenticate* the coin transfer. In earlier works [BCF+11, BCFK15], at each transfer the spender computed a signature that was included in a coin, and that committed the user to the spending (and made her accountable in case of double-spending). Our construction *does not require any user signatures* and thus gains in efficiency.

Furthermore, in [BCFK15] (there were no tags in [BCF+11]), the malleable signatures took care of ensuring well-formedness of the tags, while we give an explicit construction. To be compatible with Groth-Sahai proofs, we define structure-preserving proofs of well-formedness for serial numbers and tags.

Syntax. An $\mathcal{M}$-double-spending tag scheme T is composed of the following polynomial-time algorithms:

$\mathsf{T.Setup}(Gr)$, on input a group description, outputs the parameters par_{T} (which are an implicit input to all of the following).

$\mathsf{T.KeyGen}()$, on (implicit) input the parameters, outputs a tag key pair (sk, pk).

$\mathsf{T.SGen}(sk, n)$, the serial-number generation function, on input a secret key and a nonce $n \in \mathcal{N}$ (the nonce space), outputs a serial-number component sn and a proof $sn\text{-}pf$ of well-formedness.

$\mathsf{T.SGen}_{\text{init}}(sk, n)$ is a variant of $\mathsf{T.SGen}$ that outputs a message $M \in \mathcal{M}$ instead of a proof. ($\mathsf{SGen}_{\text{init}}$ is used for the first component of the serial number, which is signed by the bank using a signature scheme that requires messages to be in $\mathcal{M}$.)

$\mathsf{T.SVfy}(pk, sn, sn\text{-}pf)$, on input a public key, a serial number and a proof verifies that sn is consistent with pk by outputting a bit b.

$\mathsf{T.SVfy}_{\text{init}}(pk, sn, M)$, on input a public key, a serial number and a message in $\mathcal{M}$, checks their consistency by outputting a bit b.

$\mathsf{T.SVfy}_{\text{all}}$, depending on the type of the input, runs $\mathsf{T.SVfy}_{\text{init}}$ or $\mathsf{T.SVfy}$.

$\mathsf{T.TGen}(sk, n, sn)$, the double-spending tag function, takes as input a secret key, a nonce $n \in \mathcal{N}$ and a serial number, and outputs a double-spending tag $tag \in \mathcal{T}$ (the set of the double-spending tags) and a tag proof $t\text{-}pf$.

$\mathsf{T.TVfy}(pk, sn, sn', tag, t\text{-}pf)$, on input a public key, two serial numbers, a double-spending tag, and a proof, checks consistency of the tag w.r.t. the key and the serial numbers by outputting a bit b.

$\mathsf{T.Detect}(sn, sn', tag, tag', \mathcal{L})$, double-spending detection, takes as input two serial numbers sn and sn', two tags $tag, tag' \in \mathcal{T}$ and a list of public keys $\mathcal{L}$ and outputs a public key pk (of the accused user) and a proof Π.

$\mathsf{T.VfyGuilt}(pk, \Pi)$, the incrimination-proof verification function, takes as input a public key and a proof and outputs a bit b.

Correctness properties. For any double-spending tag scheme T we require that for all $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$ the following hold:

Verifiability: For every $n, n' \in \mathcal{N}$, and after computing
- $(sk, pk) \leftarrow \mathsf{T.KeyGen}$; $(sk', pk') \leftarrow \mathsf{T.KeyGen}$
- $(sn, X) \leftarrow \mathsf{T.SGen}(sk, n)$ **or** $(sn, X) \leftarrow \mathsf{T.SGen}_{\text{init}}(sk, n)$
- $(sn', sn\text{-}pf') \leftarrow \mathsf{T.SGen}(sk', n')$
- $(tag, t\text{-}pf) \leftarrow \mathsf{T.TGen}(sk, n, sn')$

we have $\mathsf{T.TVfy}(pk, sn, sn', tag, t\text{-}pf) = \mathsf{T.SVfy}_{\text{all}}(pk, sn, X) = 1$.

SN-identifiability: For all tag public keys pk_1 and pk_2, all serial numbers sn and all X_1 and X_2, which can be messages in $\mathcal{M}$ or SN proofs, if

$$\mathsf{T.SVfy}_{\text{all}}(pk_1, sn, X_1) = \mathsf{T.SVfy}_{\text{all}}(pk_2, sn, X_2) = 1$$

then $pk_1 = pk_2$.

Bootability: There do not exist an SN message M, serial numbers $sn_1 \neq sn_2$ and tag keys (not necessarily distinct) pk_1, pk_2 such that:

$$\mathsf{T.SVfy}_{\text{init}}(pk_1, sn_1, M) = \mathsf{T.SVfy}_{\text{init}}(pk_2, sn_2, M) = 1.$$

2-show extractability: Let pk_0, pk_1 and pk_2 be tag public keys, sn_0, sn_1 and sn_2 be serial numbers, X_0 be either an SN proof or a message in $\mathcal{M}$, and $sn\text{-}pf_1$ and $sn\text{-}pf_2$ be SN proofs. Let tag_1 and tag_2 be tags, and $t\text{-}pf_1$ and $t\text{-}pf_2$ be tag proofs, and let $\mathcal{L}$ be a set of tag public keys with $pk_0 \in \mathcal{L}$. If

$$\begin{aligned}
\mathsf{T.SVfy}_{\text{all}}\big(pk_0, sn_0, X_0\big) &= 1 \\
\mathsf{T.SVfy}\big(pk_1, sn_1, sn\text{-}pf_1\big) = \mathsf{T.SVfy}\big(pk_2, sn_2, sn\text{-}pf_2\big) &= 1 \\
\mathsf{T.TVfy}\big(pk_1, sn_0, sn_1, tag_1, t\text{-}pf_1\big) = \mathsf{T.TVfy}\big(pk_2, sn_0, sn_2, tag_2, t\text{-}pf_2\big) &= 1
\end{aligned}$$

and $sn_1 \neq sn_2$ then $\mathsf{T.Detect}(sn_1, sn_2, tag_1, tag_2, \mathcal{L})$ extracts (pk_0, Π) efficiently and we have $\mathsf{T.VfyGuilt}(pk_0, \Pi) = 1$.

$\mathcal{N}$-injectivity: For any secret key sk, the function $\mathsf{T.SGen}(sk, \cdot)$ is injective.

Security properties

Exculpability: This notion formalizes soundness of double-spending proofs, in that no honestly behaving user can be accused. Let $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}$ and $(sk, pk) \leftarrow \mathsf{T.KeyGen}(par_{\mathsf{T}})$. Then we require that for an adversary $\mathcal{A}$ that is given pk and can obtain SNs and tags for receiver SNs of its choice, both produced with sk (but no two tags for the same sender SN), is computationally hard to return a proof Π with $\mathsf{T.VfyGuilt}(pk, \Pi) = 1$. Formally, $\mathcal{A}$ gets access to oracles $O_1(sk)$ and $O_2(sk, \cdot, \cdot)$ defined in Fig. 7.

Tag anonymity: Finally, our anonymity notions for transferable e-cash should hold even against a malicious bank, which gets to see the serial numbers and double-spending tags for deposited coins, and the *secret keys* of the users.

$\mathbf{Expt}^{\texttt{tag-anon}}_{\mathcal{A},b}(\lambda)$:
$Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
$par_\mathsf{T} \leftarrow \mathsf{T.Setup}(Gr)$
$k := 0$
$(sk_0, sk_1) \leftarrow \mathcal{A}(par_\mathsf{T})$
$b^* \leftarrow \mathcal{A}^{O_1(sk_b), O_2(sk_b,\cdot,\cdot)}(par_\mathsf{T}, sk_0, sk_1)$
Return $(b = b^*)$

$O_1(sk)$:
$n \xleftarrow{\$} \mathcal{N}$; $T[k] := n$; $k := k + 1$
$(sn, sn\text{-}pf) \leftarrow \mathsf{T.SGen}(sk, n)$
Return sn.

$O_2(sk, sn', i)$:
If $T[i] = \bot$, abort the oracle call
$n := T[i]$; $T[i] := \bot$
$(tag, t\text{-}pf) \leftarrow \mathsf{T.TGen}(sk, n, sn')$
Return tag

Fig. 7. Game for *tag anonymity* (with oracles also used in *exculpability*) for double-spending tag schemes

Thus, we require that as long as the nonce n is random and only used once, the serial numbers and tags reveal nothing about the user-specific values, such as sk and pk, that were used to generate them. The game is given in Fig. 7.

Definition 6 (Tag anonymity). *A double-spending tag scheme is* anonymous *if* $\Pr[\mathbf{Expt}^{\texttt{tag-anon}}_{\mathcal{A},1}(\lambda) = 1] - \Pr[\mathbf{Expt}^{\texttt{tag-anon}}_{\mathcal{A},0}(\lambda) = 1]$ *is negligible in* λ *for any PPT* $\mathcal{A}$.

5 Instantiation

5.1 Overview

The bank creates money and validates new users in the system. Digital signatures can be used for these two functions: the bank signs the key of a new user, who can then prove that he is registered; and during a coin issuing, the bank signs a message M_{sn} that is associated to the initial serial-number (SN) component sn_0 of a coin, which makes coins unforgeable.

After a coin has been transferred k times, its core consists of a list of SNs $sn_0, sn_1, \ldots, sn_k$, together with a list of tags $tag_1, \ldots, tag_k$ (for a freshly withdrawn coin, we have $k = 0$). When a user spends such a coin, the receiver generates a fresh SN component sn_{k+1}, from which the spender must generate a tag tag_{k+1} that is also associated with her public key and the last serial number sn_k (which she generated when she received the coin.)

These tags allow the bank to identify the fraudster in case of double-spending, while they preserve the anonymity of honest users. A coin moreover contains the users' public key w.r.t. which the tags were created, as well as certificates on them issued by the bank. To provide anonymity, all these components are not given in the clear, but as a zero-knowledge proof of knowledge. That is, a coin is a proof of knowledge of its serial number, its tags, the user public keys and their certificates and ensures that all of them are consistent.

As we use a commit-and-prove proof system, all these values are included in the coin as commitments. Recall that a coin also includes a signature by the bank on (a message related to) the initial SN component. In order to achieve anonymity towards the bank (*coin anonymity*), the bank must sign this message

blindly, which is achieved by using the SigCm functionality: the user sends a commitment to the serial number, and the bank computes a committed signature on the committed value.

Finally, the bank needs to be able to *detect* whether a double-spending occurred and *identify* the user that committed it. One way would be to give the serial numbers and the tags (which protect the anonymity of honest users) in the clear. This would yield a scheme that satisfies *coin anonymity* and *user anonymity* (note that in these two notions the bank is corrupted). However, *coin transparency*, the most intricate anonymity notion, would not be achieved, since the owner of a coin could easily recognize it when she receives it again by looking at its serial number.

It is *coin transparency* that requires to hide the serial numbers (and the associated tags), and moreover to use a randomizable proof system, since the appearance of a coin needs to change after every transfer. To hide the serial numbers and the tags from users, but still provide access to them by the bank, we add encryptions, under the bank's public key, of them to the coin. However, these must interoperate with the randomization of the coin, which is why we require rerandomizable encryption that can be tied into the machinery of updating of the proofs, which is necessary every time the ciphertexts and the commitments contained in a coin are refreshed.

5.2 Technical description

Primitives used. The basis of our scheme is a randomizable extractable NIZK commit-and-prove scheme

$$\mathsf{C} = \big(\mathsf{C.Setup}, \mathsf{C.Cm}, \mathsf{C.RdCm}, \mathsf{C.ExSetup}, \mathsf{C.Extr}, \mathsf{C.Prv}, \mathsf{C.Verify}, \mathsf{C.AdptPrf}\big),$$

to which we add compatible schemes: an $\mathcal{M}$-structure-preserving signature scheme $\mathsf{S} = (\mathsf{S.Setup}, \mathsf{S.KeyGen}, \mathsf{S.Sign}, \mathsf{S.Verify})$, admitting an $\mathcal{M}$-commuting signature add-on SigCm, as well as an $\mathcal{M}'$-structure-preserving signature scheme S'; furthermore a double-spending tag scheme

$$\mathsf{T} = \big(\mathsf{T.Setup}, \mathsf{T.KeyGen}, \mathsf{T.SGen}, \mathsf{T.SGen}_{\mathrm{init}}, \mathsf{T.SVfy}, \mathsf{T.SVfy}_{\mathrm{init}}, \mathsf{T.TGen}, \mathsf{T.TVfy}, \mathsf{T.Detect}, \mathsf{T.VfyGuilt}\big),$$

as well as two randomizable encryption schemes $\mathsf{E} = (\mathsf{E.KeyGen}, \mathsf{E.Enc}, \mathsf{E.ReRand}, \mathsf{E.Dec}, \mathsf{E.Verify}, \mathsf{E.AdptPrf})$, and E'. (We require E' to satisfy RCCA security, whereas E need only be IACR-secure).

Auxiliary functions. In order to simplify the description of our scheme, we first define several auxiliary functions. To randomize a given tuple of commitments and ciphertext, and proofs for them (and adapting the proofs to the randomizations), we use an algorithm Rand, which internally runs $\mathsf{C.RdCm}$, $\mathsf{E.ReRand}$, $\mathsf{C.AdptPrf}$ and $\mathsf{E.AdptPrf}$ with the same randomness.

Assuming the equations for $\mathsf{T.SVfy}(\cdot,\cdot,\cdot) = 1$, $\mathsf{T.SVfy}_{\text{init}}(\cdot,\cdot,\cdot) = 1$ and $\mathsf{T.TVfy}(\cdot,\cdot,\cdot,\cdot,\cdot) = 1$, as well as $\mathsf{E.Verify}(pk,\cdot,\cdot,c) = 1$ are all in the set $\mathcal{E}$ of equations supported by the proof system, we define the following:

$\mathsf{C.Prv}_{\text{sn,init}}$ proves that a committed initial serial number sn has been honestly generated w.r.t. a committed key pk_{T} and a committed message M (given the used randomness ρ_{pk}, ρ_{sn} and ρ_M), while $\mathsf{C.Verify}_{\text{sn,init}}$ verifies such proofs. $\mathsf{C.Prv}_{\text{sn}}$ and $\mathsf{C.Verify}_{\text{sn}}$ do the same for non-initial serial numbers (for which there are no messages, but which require a proof of well-formedness instead).

$\mathsf{C.Prv}_{\text{sn,init}}(ck, pk_{\mathsf{T}}, sn, M, \rho_{pk}, \rho_{sn}, \rho_M)$:

- Return $\pi \leftarrow \mathsf{C.Prv}\big(ck, \mathsf{T.SVfy}_{\text{init}}(\cdot,\cdot,\cdot) = 1, (pk_{\mathsf{T}}, \rho_{pk}), (sn, \rho_{sn}), (M, \rho_M)\big)$

$\mathsf{C.Verify}_{\text{sn,init}}(ck, c_{pk}, c_{sn}, c_M, \pi_{sn})$:

- Return $(\mathsf{C.Verify}(ck, \mathsf{T.SVfy}_{\text{init}}(\cdot,\cdot,\cdot) = 1, c_{pk}, c_{sn}, c_M, \pi_{sn}))$

$\mathsf{C.Prv}_{\text{sn}}(ck, pk_{\mathsf{T}}, sn, sn\text{-}pf, \rho_{pk}, \rho_{sn}, \rho_{sn\text{-}pf})$:

- $\pi \leftarrow \mathsf{C.Prv}\big(ck, \mathsf{T.SVfy}(\cdot,\cdot,\cdot) = 1, (pk_{\mathsf{T}}, \rho_{pk}), (sn, \rho_{sn}), (sn\text{-}pf, \rho_{sn\text{-}pf})\big)$
- Return $(\pi, \mathsf{C.Cm}(ck, sn\text{-}pf, \rho_{sn\text{-}pf}))$

$\mathsf{C.Verify}_{\text{sn}}(ck, c_{pk}, c_{sn}, \tilde{\pi}_{sn} = (\pi_{sn}, c_{sn\text{-}pf}))$:

- Return $\mathsf{C.Verify}(ck, \mathsf{T.SVfy}(\cdot,\cdot,\cdot) = 1, c_{pk}, c_{sn}, c_{sn\text{-}pf}, \pi_{sn})$

$\mathsf{C.Prv}_{\text{tag}}$ produces a proof that a committed tag has been correctly generated w.r.t. committed serial numbers sn and sn'; and $\mathsf{C.Verify}_{\text{tag}}$ verifies such proofs.

$\mathsf{C.Prv}_{\text{tag}}(ck, pk_{\mathsf{T}}, sn, sn', tag, \rho_{pk}, \rho_{sn}, \rho'_{sn}, \rho_{tag}, t\text{-}pf, \rho_{t\text{-}pf})$

- $\pi \leftarrow \mathsf{C.Prv}\big(ck, \mathsf{T.TVfy}(\cdot,\cdot,\cdot,\cdot,\cdot) = 1, (pk_{\mathsf{T}}, \rho_{pk}), (sn, \rho_{sn}), (sn', \rho'_{sn}), (tag, \rho_{tag}), (t\text{-}pf, \rho_{t\text{-}pf})\big)$
- Return $(\pi, \mathsf{C.Cm}(ck, t\text{-}pf, \rho_{t\text{-}pf}))$

$\mathsf{C.Verify}_{\text{tag}}(ck, c_{pk}, c_{sn}, c'_{sn}, c_{tag}, \pi_{tag} = (\pi, c_{t\text{-}pf}))$:

- Return $\mathsf{C.Verify}(ck, \mathsf{T.TVfy}(\cdot,\cdot,\cdot,\cdot) = 1, c_{pk}, c_{sn}, c'_{sn}, c_{tag}, c_{t\text{-}pf}, \pi)$

$\mathsf{C.E.Prv}_{\text{enc}}$ proves that a ciphertext $\tilde{c}$ of M and $\mathsf{C.Cm}(ck, M, \rho_M)$ contain the same message; $\mathsf{C.E.Verify}_{\text{enc}}$ verifies such proofs. (Note that the output of $\mathsf{C.E.Prv}_{\text{enc}}$ is the same π as in the input of $\mathsf{E.AdptPrf}$.)

$\mathsf{C.E.Prv}_{\text{enc}}(ck, ek, M, \rho_M, \nu_M, \tilde{c})$:

- $\rho_\nu \xleftarrow{\$} \mathcal{R}$; $\pi \leftarrow \mathsf{C.Prv}(ck, \mathsf{E.Verify}\ (ek,\cdot,\cdot,\tilde{c}) = 1, (M, \rho_M), (\nu_M, \rho_\nu))$
- Return $(\pi, \mathsf{C.Cm}(ck, \nu_M, \rho_\nu))$

$\mathsf{C.E.Verify}_{\text{enc}}(ck, ek, c_M, \tilde{c}_M, \tilde{\pi}_{\text{eq}} = (\pi_{\text{eq}}, c_\nu))$:

- Return $\mathsf{C.Verify}(ck, \mathsf{E.Verify}(ek,\cdot,\cdot,\tilde{c}_M) = 1, c_M, c_\nu, \pi_{\text{eq}})$

Components of the coin. There are two types of components, the *initial* components $\mathcal{C}_{\text{init}}$, and the *standard* components $\mathcal{C}_{\text{std}}$. The first is of the form

$$coin_{\text{init}} = \big(c^0_{pk}, c^0_{cert}, \pi^0_{cert}, c^0_{sn}, \pi^0_{sn}, \varepsilon, \varepsilon, c_M, c^0_\sigma, \pi^0_\sigma, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}, \varepsilon, \varepsilon\big), \tag{1}$$

consisting of commitments (the c-values) to the withdrawer's key pk, her certificate $cert$, the initial serial number sn and the related message M, the bank's signature σ on M, and an encryption $\tilde{c}_{sn}$ of sn. It also contains proofs π_{cert} and π_{sn} of validity of $cert$ and sn and a proof $\tilde{\pi}_{sn}$ that c_{sn} and $\tilde{c}_{sn}$ contain the same value. We use ε to pad so that both types of component have the same format. Validity of an initial component is verified w.r.t. an encryption key (that can be for either E or E') and two signature verification keys for S and S':

$\mathsf{VER}_{\text{init}}\big(ek, vk, vk', coin_{\text{init}}\big)$: Return 1 iff the following hold: // $coin_{\text{init}}$ *as in* (1)
- $\mathsf{C.Verify}(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_M, c^0_\sigma, \pi^0_\sigma)$
- $\mathsf{C.Verify}\big(ck, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, c^0_{pk}, c^0_{cert}, \pi^0_{cert}\big)$
- $\mathsf{C.Verify}_{\text{sn,init}}\big(ck, c^0_{pk}, c^0_{sn}, c_M, \pi^0_{sn}\big) \wedge \mathsf{C.E}'.\mathsf{Verify}_{\text{enc}}\big(ck, ek, c^0_{sn}, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}\big)$

Standard components of a coin are of the form

$$coin_{\text{std}} = (c^i_{pk}, c^i_{cert}, \pi^i_{cert}, c^i_{sn}, \pi^i_{sn}, c^i_{tag}, \pi^i_{tag}, \varepsilon, \varepsilon, \varepsilon, \tilde{c}^i_{sn}, \tilde{\pi}^i_{sn}, \tilde{c}^i_{tag}, \tilde{\pi}^i_{tag}), \tag{2}$$

and instead of M and the bank's signature they contain a commitment c_{tag} and an encryption $\tilde{c}_{tag}$ of the tag produced by the spender (and a proof π_{tag} of validity and $\tilde{\pi}_{tag}$ proving that the values in c_{tag} and $\tilde{c}_{tag}$ are equal). A coin is verified by checking the validity and consistency of each two consecutive components. If the first is an initial component then the values $\underline{c^{i-1}_{tag}}, \underline{\pi^{i-1}_{tag}}, \underline{\tilde{c}^{i-1}_{tag}}$ and $\underline{\tilde{\pi}^{i-1}_{tag}}$ are ε; if it is a standard component then c_M, c^{i-1}_σ and π^{i-1}_σ are ε.

$\mathsf{VER}_{\text{body}}\big(ek, vk_{\mathcal{B}}, \big(c^{i-1}_{pk}, c^{i-1}_{cert}, \pi^{i-1}_{cert}, c^{i-1}_{sn}, \pi^{i-1}_{sn}, \underline{c^{i-1}_{tag}}, \underline{\pi^{i-1}_{tag}}, c_M, c^{i-1}_\sigma, \pi^{i-1}_\sigma, \tilde{c}^{i-1}_{sn}, \tilde{\pi}^{i-1}_{sn}, \underline{\tilde{c}^{i-1}_{tag}}, \underline{\tilde{\pi}^{i-1}_{tag}}\big), coin_{\text{std}}\big)$: // $coin_{\text{std}}$ *as in* (2)

Return 1 iff the following hold:
- $\mathsf{C.Verify}\big(ck, \mathsf{S}'.\mathsf{Verify}(vk_{\mathcal{B}}, \cdot, \cdot) = 1, c^i_{pk}, c^i_{cert}, \pi^i_{cert}\big)$
- $\mathsf{C.Verify}_{\text{sn}}\big(ck, c^i_{pk}, c^i_{sn}, \pi^i_{sn}\big) \wedge \mathsf{C.Verify}_{\text{tag}}\big(ck, c^{i-1}_{pk}, c^{i-1}_{sn}, c^i_{sn}, c^i_{tag}, \pi^i_{tag}\big)$
- $\mathsf{C.E.Verify}_{\text{enc}}\big(ck, ek, c^i_{sn}, \tilde{c}^i_{sn}, \tilde{\pi}^i_{sn}\big) \wedge \mathsf{C.E.Verify}_{\text{enc}}\big(ck, ek, c^i_{tag}, \tilde{c}^i_{tag}, \tilde{\pi}^i_{tag}\big)$

Our scheme. Using the above, we now give the formal definition of our transferable e-cash scheme. (Recall that par is an implicit input to all algorithms.)

$\underline{\mathsf{ParamGen}}(1^\lambda)$:
- $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
- $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$
- $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$

– $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
– $ck \leftarrow \mathsf{C.Setup}(Gr)$
– Return $par = (1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$

$\underline{\mathsf{BKeyGen}}()$:
– Parse par as $(1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
– $(sk, vk) \leftarrow \mathsf{S.KeyGen}(par_{\mathsf{S}})$
– $(sk', vk') \leftarrow \mathsf{S}'.\mathsf{KeyGen}(par_{\mathsf{S}'})$
– $(ek_{\text{init}}, dk_{\text{init}}) \leftarrow \mathsf{E}'.\mathsf{KeyGen}(Gr)$
– $(ek, dk) \leftarrow \mathsf{E.KeyGen}(Gr)$
– $(sk_{\mathsf{T}}, pk_{\mathsf{T}}) \leftarrow \mathsf{T.KeyGen}(par_{\mathsf{T}})$
– $cert \leftarrow \mathsf{S}'.\mathsf{Sign}(sk', pk_{\mathsf{T}})$
– Return $\big(sk_{\mathcal{W}} = (sk, sk'), sk_{\mathcal{CK}} = (dk_{\text{init}}, dk),$
$sk_{\mathcal{D}} = (cert, pk_{\mathsf{T}}, sk_{\mathsf{T}}), pk_{\mathcal{B}} = (ek_{\text{init}}, ek, vk, vk')\big)$

$\underline{\mathsf{Register}}\big\langle \mathcal{B}(sk_{\mathcal{W}} = (sk, sk')), \mathcal{U}(pk_{\mathcal{B}} = (ek_{\text{init}}, ek, vk, vk'))\big\rangle$:
$\mathcal{U}$: $(sk_{\mathsf{T}}, pk_{\mathsf{T}}) \leftarrow \mathsf{T.KeyGen}(1^\lambda)$; send pk_{T} to $\mathcal{B}$
$\mathcal{B}$: $cert_{\mathcal{U}} \leftarrow \mathsf{S}'.\mathsf{Sign}(sk', pk_{\mathsf{T}})$; send $cert_{\mathcal{U}}$ to $\mathcal{U}$; output pk_{T}
$\mathcal{U}$: If $\mathsf{S}'.\mathsf{Verify}(vk', pk_{\mathsf{T}}, cert_{\mathcal{U}}) = 1$, output $sk_{\mathcal{U}} \leftarrow (cert_{\mathcal{U}}, pk_{\mathsf{T}}, sk_{\mathsf{T}})$; else $\perp$

$\underline{\mathsf{Withdraw}}\big\langle \mathcal{B}(sk_{\mathcal{W}} = (sk, sk'), pk_{\mathcal{B}} = (ek_{\text{init}}, ek, vk, vk')),$
$\mathcal{U}(sk_{\mathcal{U}} = (cert_{\mathcal{U}}, pk_{\mathsf{T}}, sk_{\mathsf{T}}), pk_{\mathcal{B}})\big\rangle$:

$\mathcal{U}$:
– $n \xleftarrow{\$} \mathcal{N}$; $\rho_{sn}, \rho_{cert}, \rho_{pk}, \rho_M \xleftarrow{\$} \mathcal{R}$
– $(sn, M_{sn}) \leftarrow \mathsf{T.SGen}_{\text{init}}(sk_{\mathsf{T}}, n)$
– $c_{cert} \leftarrow \mathsf{C.Cm}(ck, cert_{\mathcal{U}}, \rho_{cert})$
– $c_{sn} \leftarrow \mathsf{C.Cm}(ck, sn, \rho_{sn})$
– $c_{pk} \leftarrow \mathsf{C.Cm}(ck, pk_{\mathsf{T}}, \rho_{pk})$
– $c_M \leftarrow \mathsf{C.Cm}(ck, M_{sn}, \rho_M)$
– $\pi_{cert} \leftarrow \mathsf{C.Prv}(ck, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, (pk_{\mathsf{T}}, \rho_{pk}), (cert_{\mathcal{U}}, \rho_{cert}))$
– $\pi_{sn} \leftarrow \mathsf{C.Prv}_{\text{sn,init}}(ck, pk_{\mathsf{T}}, sn, M_{sn}, \rho_{pk}, \rho_{sn}, \rho_M)$
– Send $(c_{pk}, c_{cert}, \pi_{cert}, c_{sn}, c_M, \pi_{sn})$ to $\mathcal{B}$

$\mathcal{B}$:
– if $\mathsf{C.Verify}(ck, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, c_{pk}, c_{cert}, \pi_{cert})$ or
$\mathsf{C.Verify}_{\text{sn,init}}(ck, c_{pk}, c_{sn}, c_M, \pi_{sn})$ fail then abort and output $\perp$.
– $(c_\sigma, \pi_\sigma) \leftarrow \mathsf{SigCm}(ck, sk, c_M)$; send (c_σ, π_σ) to $\mathcal{U}'$; return ok

$\mathcal{U}$:
– if $\mathsf{C.Verify}(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_M, c_\sigma, \pi_\sigma)$ fails, abort and output $\perp$.
– $\nu_{sn} \xleftarrow{\$} \mathcal{R}$
– $\tilde{c}_{sn} \leftarrow \mathsf{E}'.\mathsf{Enc}(ek_{\text{init}}, sn, \nu_{sn})$
– $\tilde{\pi}_{sn} \leftarrow \mathsf{C.E}'.\mathsf{Prv}_{\text{enc}}(ck, ek_{\text{init}}, sn, \rho_{sn}, \nu_{sn}, \tilde{c}_{sn})$
– $\rho'_{pk}, \rho'_{cert}, \rho'_{sn}, \rho'_M, \rho'_\sigma, \nu'_{sn}, \rho'_{\tilde{\pi},sn} \xleftarrow{\$} \mathcal{R}$ *// since $\tilde{\pi}_{sn}$ contains a commitment, we also sample randomness for it*

– $c^0 \leftarrow \mathsf{Rand}\big((c_{pk}, c_{cert}, \pi_{cert}, c_{sn}, \pi_{sn}, c_M, c_\sigma, \pi_\sigma, \tilde{c}_{sn}, \tilde{\pi}_{sn}),$
$(\rho'_{pk}, \rho'_{cert}, \rho'_{sn}, \rho'_M, \rho'_\sigma, \nu'_{sn}, \rho'_{\tilde{\pi},sn})\big)$
– Output $\big(c^0, n, sn, \rho_{sn} + \rho'_{sn}, \rho_{pk} + \rho'_{pk}\big)$

$\underline{\mathsf{Spend}}\langle \mathcal{U}(c, sk_{\mathcal{U}} = (cert, pk_{\mathsf{T}}, sk_{\mathsf{T}}), pk_{\mathcal{B}} = (ek_{\text{init}}, ek, vk, vk')),$
$\mathcal{U}'(sk'_{\mathcal{U}} = (cert', pk'_{tag}, sk'_{tag}), pk_{\mathcal{B}})\rangle$:

$\mathcal{U}'$: – $n' \xleftarrow{\$} \mathcal{N}$; $\rho'_{sn}, \rho'_{cert}, \rho'_{pk}, \rho'_{sn\text{-}pf}, \nu'_{sn} \xleftarrow{\$} \mathcal{R}$
– $(sn', sn\text{-}pf') \leftarrow \mathsf{T.SGen}(par_{\mathsf{T}}, sk'_{tag}, n')$
– $c'_{cert} \leftarrow \mathsf{C.Cm}(ck, cert', \rho'_{cert})$
– $c'_{pk} \leftarrow \mathsf{C.Cm}(ck, pk', \rho'_{pk})$
– $c'_{sn} \leftarrow \mathsf{C.Cm}(ck, sn', \rho'_{sn})$
– $c'_{sn\text{-}pf} \leftarrow \mathsf{C.Cm}(ck, sn\text{-}pf', \rho'_{sn\text{-}pf})$
– $\tilde{c}'_{sn} \leftarrow \mathsf{E.Enc}(ek, sn', \nu'_{sn})$
– $\pi'_{cert} \leftarrow \mathsf{C.Prv}(ck, \mathsf{S.Verify}(vk_{\mathcal{B}}, \cdot, \cdot) = 1, (pk'_{tag}, \rho'_{pk}), (cert', \rho'_{cert}))$
– $\pi'_{sn} \leftarrow \mathsf{C.Prv}_{\text{sn}}(ck, pk'_{\mathsf{T}}, sn', sn\text{-}pf, \rho'_{pk}, \rho'_{sn}, \rho'_{sn\text{-}pf})$
– $\tilde{\pi}'_{sn} \leftarrow \mathsf{C.E.Prv}_{\text{enc}}(ck, ek, sn', \rho'_{sn}, \nu'_{sn}, \tilde{c}'_{sn})$
– Send (sn', ρ'_{sn}) to $\mathcal{U}$

$\mathcal{U}$: – Parse c as $\big(c^0, (c^j = (c^j_{pk}, c^j_{cert}, \pi^j_{cert}, c^j_{sn}, \pi^j_{sn}, c^j_{tag}, \pi^j_{tag}, \tilde{c}^j_{sn}, \tilde{c}^j_{tag}, \tilde{\pi}^j_{sn}, \tilde{\pi}^j_{tag}))^i_{j=1}$
$n, sn, \rho_{sn}, \rho_{pk}\big)$
– $\rho_{tag}, \nu_{tag}, \rho_{t\text{-}pf} \xleftarrow{\$} \mathcal{R}$
– $(tag, t\text{-}pf) \leftarrow \mathsf{T.TGen}(par_{\mathsf{T}}, sk_{\mathsf{T}}, n, sn')$
– $c_{tag} \leftarrow \mathsf{C.Cm}(ck, tag, \rho_{tag})$
– $\tilde{c}_{tag} \leftarrow \mathsf{E.Enc}(ek, tag, \nu_{tag})$
– $\pi_{tag} \leftarrow \mathsf{C.Prv}_{tag}(ck, pk_{\mathsf{T}}, sn, sn', tag, t\text{-}pf, \rho_{pk}, \rho_{sn}, \rho'_{sn}, \rho_{tag}, \rho_{t\text{-}pf})$
– $\tilde{\pi}_{tag} \leftarrow \mathsf{C.E.Prv}_{\text{enc}}(ck, ek, tag, \rho_{tag}, \nu_{tag}, \tilde{c}_{tag})$
– Send $c' = \big(c^0, (c^j)^i_{j=1}, c_{tag}, \pi_{tag}, \tilde{c}_{tag}, \tilde{\pi}_{tag}\big)$ to $\mathcal{U}'$; output ok

$\mathcal{U}'$: – If any of the following fail then abort and output $\perp$:
– $\mathsf{VER}_{\text{init}}(ek_{\text{init}}, c^0)$
– $\mathsf{VER}_{\text{body}}(ek, vk, vk', c^{j-1}, c^j)$, for $j = 1, \ldots, i$
– $\mathsf{C.Verify}_{\text{tag}}(ck, c^i_{pk}, c^i_{sn}, c'_{sn}, c_{tag}, \pi_{tag})$
– $\mathsf{C.E.Verify}_{\text{enc}}(ck, ek, c_{tag}, \tilde{c}_{tag}, \tilde{\pi}_{tag})$
– pick uniformly at random $\vec{\rho'}$
– $c_{\text{rand}} \leftarrow \mathsf{Rand}\big(((c^j)^i_{j=0}, c'_{pk}, c'_{cert}, \pi'_{cert}, c'_{sn}, \pi'_{sn}, c_{tag}, \pi_{tag}, \tilde{c}'_{sn}, \tilde{\pi}'_{sn}, \tilde{c}'_{tag}, \tilde{\pi}'_{tag}), \vec{\rho'}\big)$
– Output $\big(c_{\text{rand}}, n', sn', \rho'_{sn} + (\vec{\rho'})_{sn'}, \rho'_{pk} + (\vec{\rho'})_{pk'}\big)$.

$\underline{\mathsf{CheckDS}}\big(sk_{\mathcal{CK}} = (dk_{\text{init}}, dk), \mathcal{DCL}, \mathcal{UL}, c\big)$:

– Parse c as $\big(c^0 = (c^0_{pk}, c^0_{cert}, \pi^0_{cert}, c^0_{sn}, \pi^0_{sn}, c^0_M, c_\sigma, \pi_\sigma, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}),$
$(c^j = (c^j_{pk}, c^j_{cert}, \pi^j_{cert}, c^j_{sn}, \pi^j_{sn}, c^j_{tag}, \pi^j_{tag}, \tilde{c}^j_{sn}, \tilde{\pi}^j_{sn}, \tilde{c}^j_{tag}, \tilde{\pi}^j_{tag}))^i_{j=1}, n, sn, \rho_{sn}, \rho_{pk}\big)$
– $\vec{sn} \leftarrow \big(\mathsf{E.Dec}(dk_{\text{init}}, \tilde{c}^0_{sn}), \mathsf{E.Dec}(dk, \tilde{c}^1_{sn}), \ldots, \mathsf{E.Dec}(dk, \tilde{c}^i_{sn})\big)$

– $\vec{tag} \leftarrow \big(\mathsf{E.Dec}(dk, \tilde{c}^1_{tag}), \ldots, \mathsf{E.Dec}(dk, \tilde{c}^i_{tag})\big)$
– If for all $(\vec{sn'}, \vec{tag'}) \in \mathcal{DCL}$: $(\vec{sn})_0 \neq (\vec{sn'})_0$
  then return $\mathcal{DCL} \,\|\, (\vec{sn}, \vec{tag})$
– Else let j be minimal so that $(\vec{sn})_j \neq (\vec{sn'})_j$
– $(pk_\mathsf{T}, \Pi) \leftarrow \mathsf{T.Detect}\big((\vec{sn})_j, (\vec{sn'})_j, (\vec{tag})_j, (\vec{tag'})_j, \mathcal{UL}\big)$
– Return (pk_T, Π)

$\underline{\mathsf{VfyGuilt}}(pk_\mathsf{T}, \Pi)$: Return $\mathsf{T.VfyGuilt}(pk_\mathsf{T}, \Pi)$

5.3 Correctness and security analysis

Theorem 7. *Our transferable e-cash scheme satisfies all* **correctness** *properties and is perfectly* **sound**.

The first four correctness properties follow in a straightforward way from the correctness properties of S, S' and C, and verifiability of T. The fifth property follows from the fact that $sk_\mathcal{D}$ has the form of a user secret key.

Because a user verifies the validity of all components of a coin before accepting it, perfect soundness of our scheme is a direct consequence of the correctness properties of S, S' and C, and in particular perfect soundness of C, as well as verifiability of T.

Detailed proofs of the following theorems can be found in Appendix A. We omit the proof for `u-an` as it is analogous to the one for `c-an`.

Theorem 8. *Let $\mathcal{N}$ be the nonce space and $\mathcal{S}$ be the space of signatures of scheme S. Let $\mathcal{A}$ be an adversary that wins the* **unforgeability** *game with advantage ϵ and makes at most d calls to* `BDepo`. *Suppose that C is perfectly sound and $(\mathcal{M} \cup \mathcal{S})$-extractable. Then there exist adversaries against the unforgeability of the signature schemes S and S' with advantages ϵ_{sig} and ϵ'_{sig}, resp., such that*

$$\epsilon \;\leq \epsilon_{\text{sig}} + \epsilon'_{\text{sig}} + d^2/|\mathcal{N}|.$$

Assume that during the adversary's deposits the bank never picks the same final nonce twice. (The probability that there is a collision is at most $d^2/|\mathcal{N}|$.) In this case, there are two ways for the adversary to win:
(1) $\mathsf{CheckDS}$ outputs $\perp$, or an invalid proof, or an unregistered user: Suppose that, during a `BDepo` call for a coin c, $\mathsf{CheckDS}$ does not return a coin list. Recall that, by assumption, the final part (chosen by the bank at deposit) of the serial number of c is fresh. Since $\mathsf{CheckDS}$ runs $\mathsf{T.Detect}$, by soundness of C and two-extractability of T, this will output a pair (pk, Π), such that $\mathsf{VfyGuilt}(pk, \Pi) = 1$. Since a coin contains a commitment to a certificate for the used tag key (and proofs of validity), we can, again by soundness of C, extract an S'-signature on pk. Now if pk is not in $\mathcal{UL}$, then it was never signed by the bank, and $\mathcal{A}$ has thus broken unforgeability of S'.
(2) $q_W < |\mathcal{DCL}|$: If the adversary creates a valid coin that has not been withdrawn, then by soundness of C, we can extract a signature by the bank on a new initial serial number and therefore break unforgeability of S.

Theorem 9. *Let $\mathcal{A}$ be an adversary that wins the game* **exculpability** *with advantage ϵ and makes u calls to the oracle* `URegist`*. Then there exist adversaries against mode-indistinguishability of* C *and tag-exculpability of* T *with advantages $\epsilon_{\text{m-ind}}$ and $\epsilon_{\text{t-exc}}$, resp., such that*

$$\epsilon \;\leq\; u \cdot \epsilon_{\text{t-exc}} + \epsilon_{\text{m-ind}}.$$

An incrimination proof in our e-cash scheme is simply an incrimination proof of the tag scheme T. Thus, if the reduction correctly guesses the user u that will be wrongfully incriminated by $\mathcal{A}$ (which it can with probability $1/u$), then we can construct an adversary against exculpability of T. The term $\epsilon_{\text{m-ind}}$ comes from the fact that we first need to switch C to hiding mode, so we can simulate π_{sn} and π_{tag} for the target user, since the oracles O_1 and O_2 in the game for tag exculpability (see Fig. 7) do not return *sn-pf* and *t-pf*.

Theorem 10. *Let $\mathcal{A}$ be an adversary that wins the* **coin anonymity** *game (*`c-an`*) with advantage ϵ and let k be an upper-bound on the number of users transferring the challenge coins. Then there exist adversaries against mode-indistinguishability of* C *and tag-anonymity of* T *with advantages $\epsilon_{\text{m-ind}}$ and $\epsilon_{\text{t-an}}$, resp., such that*

$$\epsilon \;\leq\; 2\left(\epsilon_{\text{m-ind}} + (k+1)\,\epsilon_{\text{t-an}}\right).$$

Theorem 11. *Let $\mathcal{A}$ be an adversary that wins the* **user anonymity** *game (*`u-an`*) with advantage ϵ and let k be a bound on the number of users transferring the challenge coin. Then there exist adversaries against mode-indistinguishability of* C *and tag-anonymity of* T *with advantages $\epsilon_{\text{m-ind}}$ and $\epsilon_{\text{t-an}}$, resp., such that*

$$\epsilon \;\leq\; 2\,\epsilon_{\text{m-ind}} + (k+1)\,\epsilon_{\text{t-an}}.$$

In the proof of both theorems, we first define a hybrid game in which the commitment key is switched to hiding mode (hence the loss $\epsilon_{\text{m-ind}}$, which occurs twice for $b = 0$ and $b = 1$). All commitments are then perfectly hiding and the only information available to the adversary are the serial numbers and tags. (They are encrypted in the coin, but the adversary, impersonating the bank, can decrypt them.)

We then argue that, by tag anonymity of T, the adversary cannot link a user to a pair (sn, tag), even when it knows the users' secret keys. We define a sequence of $k+1$ hybrid games (as k transfers involve $k+1$ users); going through the user vector output by the adversary, we can switch, one-by-one all users from the first two the second vector. Each switching can be detected by the adversary with probability at most $\epsilon_{\text{t-an}}$. Note the additional factor 2 for $\epsilon_{\text{t-an}}$ in game `c-an`, which is due to the fact that there are two coins in witch we switch users, whereas there is only one in game `u-an`.

Theorem 12. *Let $\mathcal{A}$ be an adversary that wins the* **coin-transparency** *game (*`c-tr`*) with advantage ϵ, let ℓ be the size of the challenge coins, and k be an upper-bound on the number of users transferring the challenge coins. Then there*

exist adversaries against mode-indistinguishability of C*, tag-anonymity of* T*, IACR-security of* E *and RCCA-security of* E' *with advantages* $\epsilon_{\text{m-ind}}$*,* $\epsilon_{\text{t-an}}$*,* ϵ_{iacr} *and* ϵ_{rcca}*, resp., such that*

$$\epsilon \;\leq\; 2\,\epsilon_{\text{m-ind}} + (k+1)\,\epsilon_{\text{t-an}} + 2\,\ell\,\epsilon_{\text{iacr}} + \epsilon_{\text{rcca}}.$$

The crucial difference to the previous anonymity theorems is that the bank is honest (which makes this strong notion possible). We therefore must rely on the security of the encryptions, for which the reduction thus does not know the decryption key. At the same time, the reduction must be able to detect double-spendings, when the adversary deposits coins. Since we use RCCA encryption, the reduction can do so by using its own decryption oracle.

As for `c-an` and `u-an`, the reduction first makes all commitments perfectly hiding and proofs perfectly simulatable (which loses $\epsilon_{\text{m-ind}}$ twice). Since all ciphertexts in the challenge coin given to the adversary are randomized, the reduction can replace all of them, except the initial one, by IACR-security of E. (Note that in the game these ciphertexts never need to be decrypted.) The factor 2ℓ is due to the fact that there are at most ℓ encryptions of SN/tag pairs. Finally, replacing the initial ciphertext (the one that enables detection of double-spending) can be done by a reduction to RCCA-security of E': the oracle $\mathtt{Depo}'$ can be simulated by using the reduction's own oracles Dec and GDec oracles (depending on whether $\mathtt{Depo}'$ is called before or after the reduction receives the challenge ciphertext) in the RCCA-security game. Note that, when during a simulation of $\mathsf{CheckDS}$, oracle GDec outputs `replay`, the reduction knows that a challenge coin was deposited, and thus increases *ctr*.

6 Instantiation of the building blocks and efficiency

The instantiations we use are all proven secure in the standard model under non-interactive hardness assumptions.

Commitments and proofs. The commit-and-prove system C will be instantiated with Groth-Sahai proofs [GS08], of which we use the instantiation based on SXDH.

Theorem 13 ([GS08]). *The Groth-Sahai scheme with values* $\mathcal{V} := \mathbb{Z}_p \cup \mathbb{G}_1 \cup \mathbb{G}_2$ *is perfectly complete, sound and randomizable; it is* $(\mathbb{G}_1 \cup \mathbb{G}_2)$*-extractable, mode-indistinguishable assuming SXDH, and perfectly hiding in the hiding mode.*

We note that moreover, all our proofs can be made zero-knowledge [GS08], because all pairing-product equations we use are homogeneous (i.e., the right-hand term is the neutral element) We have (efficient) extractability, as we only need to efficiently extract group elements from commitments (and not scalars) in our reductions. (Note that for information-theoretic arguments concerning soundness, Extr can also be inefficient.)

Signature schemes. For efficiency and type-compatibility reasons, we use two different signature schemes. The first one needs to support the functionality SigCm, which implies a specific format of messages. The second scheme is less restrictive, which simplifies the description of our scheme and makes its instantiation more efficient. While all our other components rely on standard assumptions, the following scheme is secure under a non-interactive q-type assumption defined in [AFG+10].

Theorem 14. *The signature scheme from [AFG+10] with message space* $\mathcal{M} := \{(g^m, \hat{g}^m) \,|\, m \in \mathbb{Z}_p\}$ *is (strongly) unforgeable assuming q-ADHSDH and AWFCDH (defined in Appendix D), and it supports the* SigCm *functionality [Fuc11].*

Theorem 15. *The signature scheme described in [AGHO11, Section 5] is structure-preserving with message space* $\mathcal{M}' := \mathbb{G}_2$ *and (strongly) unforgeable assuming SXDH.*

Randomizable encryption schemes. To instantiate the RCCA-secure scheme E' we follow the approach from [LPQ17]. Their construction is only for one group element, but by adapting the scheme, it can support encryption of a vector in $\mathbb{G}^n$ for arbitrary n.

In our e-cash scheme, we need to encrypt a vector in $\mathbb{G}^2$, and since it is not clear whether more recent efficient schemes like [FFHR19] can be adapted to this, we give an explicit construction, which we detail in Appendix B.2.

Recall that the RCCA-secure scheme E' is only used to encrypt the initial part of the serial number; using a less efficient scheme does thus not have a big impact on the efficiency of our scheme.

From all other ciphertexts contained in a coin (which are under scheme E) we only require IACR security, which standard ElGamal encryption satisfies under DDH. Thus, we instantiate E with ElGamal vector encryption. (Note that our instantiation of E' is also built on top of ElGamal). We prove the following in the appendix.

Theorem 16. *Assuming SXDH, our randomizable encryption scheme in Appendix B.2 is RCCA-secure and the one in Appendix B.3 is IACR-secure.*

Double-spending tags. We will use a scheme that builds on the one given in [BCFK15]. We have optimized the size of the tags and made explicit all the functionalities not given in the previous version. We defer this to Appendix B.1.

Efficiency analysis

For a group $G \in \{\mathbb{G}, \hat{\mathbb{G}}, \mathbb{Z}_p\}$, let $|G|$ denote the size of one element of G. Let c_{btsrap} denote the coin output by $\mathcal{U}$ at the end of the $\mathsf{Withdraw}$ protocol (which corresponds to c_{init} plus secret values, like n, ρ_{sn}, etc., to be used when transferring the coin), and let c_{std} one (non-initial) component of the coin. We conclude by summarizing the characteristics of our scheme in the following table and refer to Appendix C for the details of our analysis.

After k transfers the size of a coin is $|c_{\text{btsrap}}| + k|c_{\text{std}}|$.

$\lvert sk_{\mathcal{B}}\rvert$	$9\lvert\mathbb{Z}_p\rvert + 2\lvert\mathbb{G}\rvert + 2\lvert\hat{\mathbb{G}}\rvert$	$\lvert\Pi_{\text{guilt}}\rvert$	$2\lvert\mathbb{G}\rvert$
$\lvert pk_{\mathcal{B}}\rvert$	$15\lvert\mathbb{G}\rvert + 8\lvert\hat{\mathbb{G}}\rvert$	$\lvert c_{\text{btstrap}}\rvert$	$6\lvert\mathbb{Z}_p\rvert + 147\lvert\mathbb{G}\rvert + 125\lvert\hat{\mathbb{G}}\rvert$
$\lvert sk_{\mathcal{U}}\rvert$	$\lvert\mathbb{Z}_p\rvert + 2\lvert\mathbb{G}\rvert + 2\lvert\hat{\mathbb{G}}\rvert$	$\lvert c_{\text{std}}\rvert$	$54\lvert\mathbb{G}\rvert + 50\lvert\hat{\mathbb{G}}\rvert$
$\lvert pk_{\mathcal{U}}\rvert$	$\lvert\hat{\mathbb{G}}\rvert$	$\lvert(\vec{sn}, \vec{tag})\rvert$	$(4t+2)\lvert\mathbb{G}\rvert$

Acknowledgements. The first two authors were supported by the French ANR EfTrEC project (ANR-16-CE39-0002). This work is funded in part by the MSR–Inria Joint Centre. The second author is supported by the Vienna Science and Technology Fund (WWTF) through project VRG18-002.

References

AFG+10. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. *CRYPTO'10*.

AGHO11. Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. *CRYPTO'11*.

BCC+09. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. *CRYPTO'09*.

BCF+11. Olivier Blazy, Sébastien Canard, Georg Fuchsbauer, Aline Gouget, Hervé Sibert, and Jacques Traoré. Achieving optimal anonymity in transferable e-cash with a judge. *AFRICACRYPT'11*.

BCFK15. Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. Anonymous transferable E-cash. *PKC'15*.

BCG+14. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. *IEEE S&P'14*.

BCKL09. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. Compact e-cash and simulatable VRFs revisited. *Pairing'09*.

Bla08. Marina Blanton. Improved conditional e-payments. *ACNS'08*.

BPS19. Florian Bourse, David Pointcheval, and Olivier Sanders. Divisible e-cash from constrained pseudo-random functions. In *ASIACRYPT'19*

Bra93. Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). *CRYPTO'93*.

CFN88. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. *CRYPTO'88*.

CG08. Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. *ACNS'08*.

CGT08. Sébastien Canard, Aline Gouget, and Jacques Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. *Financial Cryptography'08*.

Cha83. David Chaum. Blind signature system. *CRYPTO'83*.

CHL05. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. *EUROCRYPT'05*.

CKLM12. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. *EUROCRYPT'12*.
CKLM14. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. *IEEE CSF'14*.
CKN03. Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. *CRYPTO'03*.
CP93. David Chaum and Torben P. Pedersen. Transferred cash grows in size. *EUROCRYPT'92*
CPST16. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. *IET Information Security*, 10(6):332–347, 2016.
FFHR19. Antonio Faonio, Dario Fiore, Javier Herranz, and Carla Ràfols. Structure-preserving and re-randomizable rcca-secure public key encryption and its applications. *ASIACRYPT'19*.
FHY13. Chun-I Fan, Vincent Shi-Ming Huang, and Yao-Chun Yu. User efficient recoverable off-line e-cash scheme with fast anonymity revoking. *Mathematical and Computer Modelling*, 58(1-2):227–237, 2013.
FP09. Georg Fuchsbauer and David Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. *PAIRING'09*.
FPV09. Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable constant-size fair e-cash. *CANS'09*.
Fuc11. Georg Fuchsbauer. Commuting signatures and verifiable encryption. *EUROCRYPT'11*.
GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. *EUROCRYPT'08*.
LPJY13. Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. *CRYPTO'13*.
LPQ17. Benoît Libert, Thomas Peters, and Chen Qian. Structure-preserving chosen-ciphertext security with shorter verifiable ciphertexts. *PKC'17*.
Nak08. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash. bitcoin.org/bitcoin.pdf, 2008.
OO89. Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. *CRYPTO'89*.
OO91. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. *CRYPTO'91*.
vS13. Nicolas van Saberhagen. Cryptonote v 2.0, 2013. https://cryptonote.org/whitepaper.pdf.