# 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
balthazar.bauer@ens.fr
[2] TU Wien, Vienna, Austria
georg.fuchsbauer@tuwien.ac.at
[3] NTNU, Trondheim, Norway
chen.qian@ntnu.no

**Abstract.** Transferable e-cash is the most faithful digital analog of physical cash, as it allows users to transfer coins between them in isolation, that is, without interacting with a bank or a "ledger". Appropriate protection of user privacy and, at the same time, providing means to trace fraudulent behavior (double-spending of coins) have made instantiating the concept notoriously hard. Baldimtsi et al. (PKC'15) gave a first instantiation, but, as it relies on a powerful cryptographic primitive, the scheme is not practical. We also point out a flaw in their scheme.

In this paper we revisit the model for transferable e-cash and propose simpler yet stronger security definitions. We then provide the first concrete construction, based on bilinear groups, give rigorous proofs that it satisfies our model, and analyze its efficiency in detail.

## 1 Introduction

Contrary to so-called "crypto"-currencies like Bitcoin [Nak08], a central ambition of the predating cryptographic *e-cash* has been 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 name only a few). In e-cash, 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 readily achieved when all actors are online and connected (as with cryptocurrencies), since every spending is broadcast and payees simply refuse a coin that has already been spent.

Even in "anonymous" cryptocurrencies like *Monero* [vS13], which now also uses *confidential transactions* [Max15], or systems based on the Zerocoin/-cash [MGGR13, BCG+14] protocol, like *Zcash* [Zec20], or on Mimblewimble


J. A. Garay (Ed.): PKC 2021, LNCS 12711, pp. 559–590, 2021.
https://doi.org/10.1007/978-3-030-75248-4_20

[Poe16,FOS19], users must be connected when they accept a payment, in order to prevent double-spending.

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 for revealing a double-spender's identity 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. Note that cryptocurrencies are inherently online, and every transfer of a coin could be seen as depositing a coin (and marking it spent) and re-issuing a new one (in the ledger).

Transferable e-cash was first proposed by Okamoto and Ohta [OO89,OO91], but the constructions only guaranteed very weak forms of anonymity. It was then shown [CP93] that *unbounded* adversaries can recognize coins they owned earlier and that a coin must grow in size with every transfer (since information about potential double-spenders needs to be encoded in it).

While other schemes [Bla08,CGT08] only achieve unsatisfactory anonymity notions, Canard and Gouget [CG08] define a stronger notion (which we call *coin transparency*): it requires that a (polynomial-time) adversary cannot recognize a coin he has already owned when it is later given back to him. This is not achieved by physical cash, as banknotes can be marked by users (or the bank); however, if an e-cash scheme allowed a merchant to identify users by tracing the coins given out as change, then it would violate the central claim of e-cash, namely anonymous payments. (Anonymous cryptocurrencies also satisfy a notion analogous to coin transparency.) A limitation of this notion is that the bank (more specifically, the part dealing with deposits) must be honest, as it must be able to link occurrences of the same coin to detect double-spending.

**Prior Schemes.** The first scheme achieving coin transparency [CG08] was completely impractical, as at every transfer, the payer sends a proof of (a proof of (... (a proof of 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 must give up their anonymity.

Blazy et al. [BCF+11] overcome this problem and propose a scheme that assumes a trusted party (called the "judge") that can trace all coins and users in the system and has to actively intervene to identify double-spenders. The scheme thus reneges on the promise that users remain anonymous as long as they follow the protocol. Moreover, their proof of anonymity was flawed, as shown by Baldimtsi et al. [BCFK15].

Despite all its problems, Blazy et al.'s [BCF+11] scheme, which elegantly combined randomizable non-interactive zero-knowledge (NIZK) proofs [BCC+09] and commuting signatures [Fuc11], serves as starting point for our

construction. In their scheme a coin consists of a signature by the bank and at every transfer the spender adds her own signature (thereby committing to her spending). To achieve anonymity, these signatures are not given in the clear; instead, coins are NIZK proofs of knowledge of signatures. Since the proofs can be rerandomized (that is, from a proof, anyone can produce a new proof of the same statement that looks unrelated to the original proof), coins can change appearance after every transfer. Users will thus not recognize a coin when they see it again later, meaning the scheme satisfies coin transparency.

Baldimtsi et al. [BCFK15] give an instantiation that avoids the "judge" by using a double-spending-tracing mechanism from classical offline e-cash. They add "tags" to the coin that hide the identity of the owner of the coin, except when she spends the coin twice, then the bank can from two such tags compute the user's identity. Users must also include signatures in the coin during transfer, which represent irrefutable proof of double-spending.

The main drawback of their scheme is efficiency. They rely on the concept of *malleable signatures* [CKLM14], a generalization of digital signatures, where a signature on a message $m$ can be transformed into a signature on a message $T(m)$ for any allowed transformation $T$. *Simulation unforgeability* requires that from a signature one can extract all transformations it has undergone (even when the adversary that created it has seen "simulated" signatures).

In their scheme [BCFK15] a coin is a malleable signature computed by the bank, which can be *transformed* by a user 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 contained in the coin.

None of the previous schemes explicitly considers *denominations* of coins (and neither do we). This is because efficient ("compact") withdrawing and spending can be easily achieved if the bank associates different keys to different denominations (since giving *change* is readily supported in transferable e-cash). Note that, in contrast to cryptocurrencies, where every transaction is publicly posted, hiding the *amount* of a payment is meaningless in transferable e-cash.

**Our Contribution: Security Model.** We revisit the formal model for transferable e-cash, starting from [BCFK15], whose model was a refined version of earlier ones. We then 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 previous models did not guarantee that this coin will be accepted by other (honest) users when transferred. An adversary could thus send a malformed coin to a user, which the latter accepts but can then not spend.
- There were also no guarantees 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 must accuse a user of double-spending and provide a proof for this.

We then 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 existing formalizations we argue are not relevant in practice).

We also show that the proof of "StR anonymity" (a notion similar to coin transparency) of the scheme from [BCFK15] is flawed and that it only satisfies a weakening of the notion (Sect. 3.2).

**Our Contribution: Instantiation.** Our main contribution is a transferable e-cash scheme, which we prove satisfies our security model, and which is more efficient than the only previous realization [BCFK15]. Unfortunately, the authors do not provide concrete numbers, as they use malleable signatures in a black-box way. Arguably, these signatures are the main source of inefficiency, due to their generality and the strong security notions in the spirit of *simulation-sound extractability*, requiring that a coin (i.e., a malleable signature) stores every transformation it has undergone.

In contrast, we give a direct construction from the following primitives: 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, see Fig. 6). While we use signature schemes from the literature [AGHO11,Fuc11], we construct a new RCCA-secure encryption scheme that is tailored to our scheme, basing it on prior work [LPQ17]. Finally, our scheme also uses the (efficient) double-spending tags used previously [BCFK15].

Due to the existence of an omnipotent "judge", no such tags were required by Blazy et al. [BCF+11]. Interestingly, although we do not assume any active trusted parties, we achieve a comparable efficiency, which is a result of realizing the full potential of the tags: previously [BCFK15], tags had only served to *encode* a user's identity; but, as we show, they can in addition be used to *commit* the user. This allows us, contrary to all previous instantiations, to completely forgo the inclusion of user signatures in the coins, which considerably reduces their size. For a more detailed (informal) overview of our scheme see Sect. 5.1.

In terms of efficiency, our coins grow by around 100 elements from a bilinear group per transfer (see table on p. 28). We view this as practical by current standards, especially in view of numbers for deployed schemes: e.g., the parameters for *Zcash* consist of several 100 000 bilinear-group elements [Zec20].

## 2 Definition of Transferable E-Cash

The syntax and security definitions we present in the following are refinements of earlier work [CG08,BCF+11,BCFK15].

### 2.1 Algorithms and Protocols

An e-cash scheme is set up by running $\mathsf{ParamGen}$ and the bank generating its key pair via $\mathsf{BKeyGen}$. 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 $\Pi$, which can be verified using VfyGuilt.

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

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 receiver secret key when coins are deposited via Spend; and $sk_\mathcal{CK}$ is used for CheckDS.

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 $\mathcal{UL}$.

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 $\perp$ in case of error), while the bank outputs *ok* (in which case it debits the user's account) or $\perp$.

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 $\perp$).

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 $\Pi$.

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 $\Pi$) 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 issued 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 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 obtained by the adversary are not stored); *cds* counts how often this *specific instance* of the coin has been spent; *origin* is set 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, then *origin* indicates which original coin this transferred coin corresponds to.

Finally, we maintain a list of deposited coins $\mathcal{DCL}$.

**Oracles.** Our security games use oracles, 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 $(pk, \perp, uds = 0)$ 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.

$\mathbf{Expt}^{\mathtt{sound}}_{\mathcal{A}}(\lambda)$:
$par \leftarrow \mathsf{ParamGen}(1^\lambda)$; $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
$(b, i_1, i_2) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy}}$
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 and 0 otherwise

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

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

$\mathtt{S\&R}(j, i)$ 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 $(owner = i, c, cds = 0, origin = j)$ 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.

$\mathtt{BDepo}()$ lets $\mathcal{A}$ deposit a coin. It runs $\mathcal{U}'$ in $\mathsf{Spend}$ using the bank's secret key $sk_{\mathcal{D}}$ with the adversary playing $\mathcal{U}$. If successful, it runs $\mathsf{CheckDS}$ on the received coin and either updates $\mathcal{DCL}$ or returns a pair $(pk, \Pi)$.

$\mathtt{Depo}(j)$, the honest deposit oracle, runs $\mathsf{Spend}$ 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 $\mathsf{CheckDS}$ on the received coin and either updates $\mathcal{DCL}$ or returns a pair $(pk, \Pi)$.

(No "$\mathtt{UDepo}$" is needed since $\mathtt{Spd}$ lets user deposit at an adversarial bank.)

### 2.3 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 formalization of transferable e-cash in the literature and is analogous the property *clearing* defined for classical e-cash [BPS19].

**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.

**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 $\lambda$.*

**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, it 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 given in Fig. 2 and lets the adversary impersonate all users.

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

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

**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* $\lambda$ *for any PPT* $\mathcal{A}$.

**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 $\Pi^*$ 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 (and 0 otherwise):
– $\mathsf{VfyGuilt}(pk_{i^*}, \Pi^*) = 1$
– There was no call $\mathtt{Spy}(i^*)$
– $uds_{i^*} = 0$

**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* $\lambda$ *for any PPT* $\mathcal{A}$.

### 2.4 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 or not 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.

**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 transferred 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.

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

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

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

**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.

$\mathbf{Expt}^{\text{c-tr}}_{\mathcal{A},b}(\lambda)$:
$par \leftarrow \mathsf{ParamGen}(1^\lambda)$; $((sk_{\mathcal{W}}, sk_{\mathcal{D}}, sk_{\mathcal{CK}}), pk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}(par)$
$\mathcal{DCL}' \leftarrow \emptyset$ // *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)

**Coin Transparency.** This is arguably the strongest anonymity notion and it implies that a user that transfers a coin cannot recognize it if she receives it again. 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 transferred 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 that 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 often 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* $\mathtt{x} \in \{\mathtt{c-an}, \mathtt{u-an}, \mathtt{c-tr}\}$ *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* $\lambda$ *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].

**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", whose meaning is unclear, since an "unsuccessful deposit" could mean one of the following:

- 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 prevented so the bank does not lose money without being able to accuse the cheater. This is now guaranteed by our unforgeability notion in Definition 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:

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

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 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 it 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 $\mathcal{UL}$.

Finally, no prior scheme, including [BCFK15], achieves `StR-fa`, as shown next.

### 3.2 A Flaw in a Proof in BCFK15

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

In the game formalizing `StR-fa` [BCF+11] (analogously to $\mathbf{Expt}^{\texttt{c-tr}}$ in Fig. 5) the adversary receives $sk_{\mathcal{W}}$, which formalizes the notion that the part of the bank

that issues coins can be corrupt. In their scheme [BCFK15], $sk_{\mathcal{W}}$ contains the signing key for the malleable signatures. However, with this the adversary can easily compute a *fresh* signature, and thus no extractor can recover a transformation explaining the signed message. This shows that a scheme based on malleable signatures only satisfies a weaker notion of `StR-fa/c-tr`, where all parts of the bank must be honest.

In contrast, we prove that our scheme satisfies `c-tr`; it can therefore be seen as the first scheme to satisfy the "spirit" of `StR-fa`, as captured by `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, such as SXDH, defined in the full version [BFQ20], hold as well. We assume that there exists an algorithm $\mathsf{GrGen}$ that, on input the security parameter $\lambda$ 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 $\rho$.
$\mathsf{C.Extr}(xk, c)$, 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 $\pi$.

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

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

**$\mathcal{M}$-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 $\Sigma$. We require that $\Sigma$ 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 $\Sigma$, 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 $c_\Sigma$ and a proof $\pi$ that the signature in $c_\Sigma$ 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 $\Sigma$, outputs a committed signature $c_\Sigma$ and a proof $\pi$ of validity for $c_\Sigma$ and $c$ (the key $xk$ is needed to compute $\pi$ 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:* Committed values from a subset $\mathcal{V}' \subset \mathcal{V}$ can be efficiently extracted (e.g., $\mathcal{V}' = \mathbb{G}_1 \cup \mathbb{G}_2$ [GS08]). Let $(ck, xk) \leftarrow \mathsf{C.ExSetup}$; then $\mathsf{C.Extr}(xk, \cdot)$ is efficient for all $c = \mathsf{C.Cm}(ck, v, \rho)$ 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 (knowledge) soundness:* Let $(ck, xk) \leftarrow \mathsf{C.ExSetup}$, $E \subset \mathcal{E}$, $(c_1, \ldots, c_n) \in \mathcal{C}^n$. If $\mathsf{C.Verify}(ck, E, c_1, \ldots, c_n, \pi) = 1$ for some $\pi$, 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 $E \subset \mathcal{E}$; for all $(v_1, \ldots, v_n) \in \mathcal{V}^n$ satisfying $E$, and $\rho_1, \rho'_1, \ldots, \rho_n, \rho'_n \in \mathcal{R}$ the following are distributed equivalently:

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

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

*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 $M \in \mathcal{M}$; for $\rho, \rho' \xleftarrow{\$} \mathcal{R}$, 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 equivalence 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 $(M, \Sigma)$, 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 (Definition 4), we use rerandomizable encryption. 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 $\mathsf{E}$ consists of four 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 $\nu$ outputs a ciphertext.

$\mathsf{E.ReRand}(ek, C, \nu')$, on input an encryption key, a ciphertext and (partial) randomness, outputs a new ciphertext.

$\mathsf{E.Dec}(dk, C)$, on input a decryption key and a ciphertext, outputs either a message or $\perp$ indicating an error.

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 $\nu$. 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 $c_\nu$ is a commitment to the randomness $\nu$, and $\pi$ proves that the values in $c_M$ and $c_\nu$ 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 $(\pi, c_\nu)$ 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 keys, a commitment, a ciphertext, an equality proof (i.e., a proof and a commitment) and randomness, outputs a new equality proof $(\pi', c'_\nu)$.

**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 $\nu$ 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 $\nu$: $\mathsf{E.Verify}(ek, M, \nu, C) = 1$ implies $\mathsf{E.Dec}(dk, C) = M$. (These two notions imply the standard correctness notion.)
- For all $M \in \mathcal{M}$ and randomness $\nu, \nu'$, 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 (with $\rho'_\nu \xleftarrow{\$} \mathcal{R}$):

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

**Security Properties.** We require two properties: the standard (strongest possible) variant of CCA security; a new notion that is easier to achieve.

*Replayable-CCA (RCCA) Security.* We use the definition by 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* $\lambda$ *for any PPT* $\mathcal{A}$*.*

### 4.4 Double-Spending Tag Schemes

Our e-cash scheme follows 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 $tag = tag_1 \| \ldots \| tag_k$, 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 $sn_i$: then 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 their 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 $\mathsf{T}$ is composed of the following polynomial-time algorithms:

$\mathsf{T.Setup}(Gr)$, on input a group description, outputs the parameters $par_{\mathsf{T}}$ (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 $sn - pf$ of well-formedness.

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

$\mathsf{T.SVfy}(pk, sn, sn - pf)$, 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 generator, 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 $t - pf$.

$\mathsf{T.TVfy}(pk, sn, sn', tag, t - pf)$, 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 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 $\Pi$.

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

$\mathbf{Expt}^{\text{tag-anon}}_{\mathcal{A},b}(\lambda)$:
$Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
$par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
$k := 0$
$(sk_0, sk_1) \leftarrow \mathcal{A}(par_{\mathsf{T}})$
$b^* \leftarrow \mathcal{A}^{O_1(sk_b), O_2(sk_b,\cdot,\cdot)}$
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 $T[i] = \perp$, abort the oracle call
$n := T[i]$; $T[i] := \perp$
$(tag, t\text{-}pf) \leftarrow \mathsf{T.TGen}(sk, n, sn')$
Return $tag$

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

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

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

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

*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 $sn - pf_1$ and $sn - pf_2$ be SN proofs. Let $tag_1$ and $tag_2$ be tags, and $t - pf_1$ and $t - pf_2$ be tag proofs, and let $\mathcal{L}$ be a set of tag public keys with $pk_0 \in \mathcal{L}$. If

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

and $sn_1 \neq sn_2$ then $\mathsf{T.Detect}(sn_1, sn_2, tag_1, tag_2, \mathcal{L})$ extracts $(pk_0, \Pi)$ 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 a PPT 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 $\Pi$ 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:* Our anonymity notions for transferable e-cash should hold even against a malicious bank that gets to see the serial numbers and double-spending tags for deposited coins and the *secret keys* of the users. We require thus that as long as the nonce $n$ is random and only used once, 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* $\Pr[\mathbf{Expt}^{\mathtt{tag-anon}}_{\mathcal{A},1}(\lambda) = 1] - \Pr[\mathbf{Expt}^{\mathtt{tag-anon}}_{\mathcal{A},0}(\lambda) = 1]$ *is negligible in* $\lambda$ *for any PPT* $\mathcal{A}$.

## 5 Our Transferable E-Cash Construction

### 5.1 Overview

The bank validates new users in the system and creates money, and digital signatures can be used for both purposes: when a new user joins, the bank signs her public key, which serves as proof of being registered; 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 (chosen by the user withdrawing the coin), and this signature makes the coin 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 $sn_{k+1}$, for which the spender must generate a tag $tag_{k+1}$, which 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 cheater in case of double-spending, while they preserve honest users' anonymity, also towards the bank. A coin moreover contains the users' public key w.r.t. which the tags were created, as well as certificates from the bank on them. To provide anonymity, all these components are not given in the clear, but as a zero-knowledge proof of knowledge. As we use a commit-and-prove proof system, a coin contains commitments to

its serial number, its tags, the user public keys and their certificates and proofs that ensure all of them are consistent.

Recall that a coin also includes a signature by the bank on (a message related to) the initial SN component. To achieve anonymity towards the bank (*coin anonymity*), the bank must sign this message blindly, which is achieved by using the $\mathsf{SigCm}$ 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 adversarially controlled). In contrast, *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.

Coin transparency requires to hide the serial numbers (and the associated tags), and to use a randomizable proof system, since the appearance of a coin needs to change after every transfer. At the same time we need to provide the bank with access to them; we thus include encryptions, under the bank's public key, in the coin. And we add proofs of consistency of the encrypted values. Now all of this must interoperate with the randomization of the coin, which is why we require rerandomizable encryption. Moreover, this has to be tied into the machinery of updating 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 transferable e-cash scheme is a randomizable extractable NIZK commit-and-prove scheme $\mathsf{C}$ to which we add compatible schemes: an $\mathcal{M}$-structure-preserving signature scheme $\mathsf{S}$ that admits an $\mathcal{M}$-commuting signature add-on $\mathsf{SigCm}$, as well as a (standard) $\mathcal{M}'$-structure-preserving signature scheme $\mathsf{S}'$ (all defined in Sect. 4.2).

Our scheme moreover uses rerandomizable encryption (Sect. 4.3): a scheme $\mathsf{E}$, which only needs to be IACR-secure, and an RCCA-secure scheme $\mathsf{E}'$, which will only be used for a single ciphertext per coin. (One can instantiate $\mathsf{E}$ withmore efficient schemes.) Finally, we use a double-spending tag scheme $\mathsf{T}$ (Sect. 4.4). We require $\mathsf{E}$, $\mathsf{E}'$ and $\mathsf{T}$ to be compatible with the proof system $\mathsf{C}$, that is, the equations for $\mathsf{E.Verify}$ and $\mathsf{E}'.\mathsf{Verify}$, as well as $\mathsf{T.SVfy}$, $\mathsf{T.SVfy}_{\mathrm{init}}$ and $\mathsf{T.TVfy}$, are all in the set $\mathcal{E}$ of equations supported by $\mathsf{C}$.

**Auxiliary Functions.** To simplify the description of our scheme, we first define several auxiliary functions. We let $\mathsf{Rand}$ denote an algorithm that randomizes a given tuple of commitments and ciphertext, as well as proofs for them (and adapts the proofs to the randomizations) by internally running $\mathsf{C.RdCm}$, $\mathsf{E.ReRand}$, $\mathsf{C.AdptPrf}$ and $\mathsf{E.AdptPrf}$ with the same randomness.

Below, we define $\mathsf{C.Prv}_{\mathrm{sn,init}}$ that produces a proof that a committed initial serial number $sn$ was correctly generated w.r.t. a committed key $pk_{\mathsf{T}}$ and a committed message $M$ (given the randomness $\rho_{pk}$, $\rho_{sn}$ and $\rho_M$ used for the commitments). We also define $\mathsf{C.Verify}_{\mathrm{sn,init}}$ that verifies such proofs. $\mathsf{C.Prv}_{\mathrm{sn}}$ and $\mathsf{C.Verify}_{\mathrm{sn}}$ do the same for non-initial serial numbers (for which there are no messages, but which require a proof of well-formedness instead).

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

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

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

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

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

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

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

$\mathsf{C.E.Prv}_{\mathrm{enc}}$ produces a proof that a ciphertext $\tilde{c}$ of $M$ and $\mathsf{C.Cm}(ck, M, \rho_M)$ contain the same message; $\mathsf{C.E.Verify}_{\mathrm{enc}}$ verifies such proofs. (Note that the output of $\mathsf{C.E.Prv}_{\mathrm{enc}}$ is the same $\pi$ as in the input of $\mathsf{E.AdptPrf}$; moreover, since $\rho_\nu$ is not used outside of $\mathsf{C.E.Prv}_{\mathrm{enc}}$, it can be sampled locally.)

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

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

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

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

where the "$c$-values" are commitments to the withdrawer's key $pk$, her certificate $cert$, the initial serial number $sn$ and the related message $M$, the bank's signature $\sigma$ on $M$; and $\tilde{c}_{sn}$ is an encryption of $sn$. Moreover, $\pi_{cert}$ and $\pi_{sn}$ prove validity of $cert$ and $sn$, and $\tilde{\pi}_{sn}$ proves that $c_{sn}$ and $\tilde{c}_{sn}$ contain the same value. We use "empty values" $\varepsilon$ for padding so that both coin-component types have the same format. Validity of an initial component is verified w.r.t. an encryption key for $\mathsf{E}'$ and two signature verification keys for $\mathsf{S}$ and $\mathsf{S}'$:

$\mathsf{VER}_{\text{init}}\left(ek', vk, vk', coin_{\text{init}}\right)$: Return 1 iff the following hold: // $coin_{\text{init}}$ *as in* (1)

- $\mathsf{C.Verify}\left(ck, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, c^0_{pk}, c^0_{cert}, \pi^0_{cert}\right)$
- $\mathsf{C.Verify}\left(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_M, c^0_\sigma, \pi^0_\sigma\right)$
- $\mathsf{C.Verify}_{\text{sn,init}}\left(ck, c^0_{pk}, c^0_{sn}, c_M, \pi^0_{sn}\right) \wedge \mathsf{C.E}'.\mathsf{Verify}_{\text{enc}}\left(ck, ek', c^0_{sn}, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}\right)$

*Standard* components of a coin are of the form

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

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 $\pi_{tag}$ 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 $\varepsilon$; if it is a standard component then $c_M, c^{i-1}_\sigma$ and $\pi^{i-1}_\sigma$ are $\varepsilon$.

$\mathsf{VER}_{\text{std}}\left(ek, vk', \left(c^{i-1}_{pk}, c^{i-1}_{cert}, \pi^{i-1}_{cert}, c^{i-1}_{sn}, \pi^{i-1}_{sn}, \underline{c^{i-1}_{tag}}, \underline{\pi^{i-1}_{tag}}, c_M, c^{i-1}_\sigma, \pi^{i-1}_\sigma, \tilde{c}^{i-1}_{sn}, \tilde{\pi}^{i-1}_{sn}, \underline{\tilde{c}^{i-1}_{tag}}, \underline{\tilde{\pi}^{i-1}_{tag}}\right), coin_{\text{std}}\right)$: // $coin_{\text{std}}$ *as in* (2)
Return 1 iff the following hold:

- $\mathsf{C.Verify}\left(ck, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, c^i_{pk}, c^i_{cert}, \pi^i_{cert}\right)$
- $\mathsf{C.Verify}_{\text{sn}}\left(ck, c^i_{pk}, c^i_{sn}, \pi^i_{sn}\right) \wedge \mathsf{C.Verify}_{\text{tag}}\left(ck, c^{i-1}_{pk}, c^{i-1}_{sn}, c^i_{sn}, c^i_{tag}, \pi^i_{tag}\right)$
- $\mathsf{C.E.Verify}_{\text{enc}}\left(ck, ek, c^i_{sn}, \tilde{c}^i_{sn}, \tilde{\pi}^i_{sn}\right) \wedge \mathsf{C.E.Verify}_{\text{enc}}\left(ck, ek, c^i_{tag}, \tilde{c}^i_{tag}, \tilde{\pi}^i_{tag}\right)$

**Our Scheme.** We now formally define our transferable e-cash scheme.

$\underline{\mathsf{ParamGen}}(1^\lambda)$:

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

Recall that *par*, parsed as above, is an implicit input to all other algorithms.

$\underline{\mathsf{BKeyGen}}()$:

- $(sk, vk) \leftarrow \mathsf{S.KeyGen}(par_{\mathsf{S}})$ ; $(sk', vk') \leftarrow \mathsf{S'.KeyGen}(par_{\mathsf{S}'})$
- $(ek', dk') \leftarrow \mathsf{E'.KeyGen}(Gr)$ ; $(ek, dk) \leftarrow \mathsf{E.KeyGen}(Gr)$
- $(sk_{\mathsf{T}}, pk_{\mathsf{T}}) \leftarrow \mathsf{T.KeyGen}(par_{\mathsf{T}})$ // $(sk_{\mathsf{T}}, pk_{\mathsf{T}}, cert)$ *let the bank act...*
- $cert \leftarrow \mathsf{S'.Sign}(sk', pk_{\mathsf{T}})$ // *dots as* $\mathcal{U}'$ *in* Spend *during deposit*
- Return $\big(sk_{\mathcal{W}} = (sk, sk'), sk_{\mathcal{CK}} = (dk', dk),$
  $sk_{\mathcal{D}} = (cert, pk_{\mathsf{T}}, sk_{\mathsf{T}}), pk_{\mathcal{B}} = (ek', ek, vk, vk')\big)$

$\underline{\mathsf{Register}}\big\langle \mathcal{B}(sk_{\mathcal{W}} = (sk, sk')), \mathcal{U}(pk_{\mathcal{B}} = (ek', ek, vk, vk'))\big\rangle$:

$\mathcal{U}$: $(sk_{\mathsf{T}}, pk_{\mathsf{T}}) \leftarrow \mathsf{T.KeyGen}(par_{\mathsf{T}})$ ; send $pk_{\mathsf{T}}$ to $\mathcal{B}$

$\mathcal{B}$: $cert_{\mathcal{U}} \leftarrow \mathsf{S'.Sign}(sk', pk_{\mathsf{T}})$ ; send $cert_{\mathcal{U}}$ to $\mathcal{U}$ ; output $pk_{\mathsf{T}}$

$\mathcal{U}$: If $\mathsf{S'.Verify}(vk', pk_{\mathsf{T}}, cert_{\mathcal{U}}) = 1$, output $sk_{\mathcal{U}} \leftarrow (cert_{\mathcal{U}}, pk_{\mathsf{T}}, sk_{\mathsf{T}})$; else $\perp$

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

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

$\mathcal{B}$:
- if $\mathsf{C.Verify}(ck, \mathsf{S'.Verify}(vk', \cdot, \cdot) = 1, c_{pk}, c_{cert}, \pi_{cert}) = 0$ or
  $\mathsf{C.Verify}_{\text{sn,init}}(ck, c_{pk}, c_{sn}, c_M, \pi_{sn}) = 0$ then abort and output $\perp$
- $(c_\sigma, \pi_\sigma) \leftarrow \mathsf{SigCm}(ck, sk, c_M)$ ; send $(c_\sigma, \pi_\sigma)$ to $\mathcal{U}'$ ; return *ok*

$\mathcal{U}$:
- if $\mathsf{C.Verify}(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_M, c_\sigma, \pi_\sigma) = 0$ then abort and output $\perp$
- $\nu_{sn} \xleftarrow{\$} \mathcal{R}$ ; $\tilde{c}_{sn} \leftarrow \mathsf{E'.Enc}(ek', sn, \nu_{sn})$
- $\tilde{\pi}_{sn} \leftarrow \mathsf{C.E'.Prv}_{\text{enc}}(ck, ek', sn, \rho_{sn}, \nu_{sn}, \tilde{c}_{sn})$
- $\rho'_{pk}, \rho'_{cert}, \rho'_{sn}, \rho'_M, \rho'_\sigma, \nu'_{sn}, \rho'_{\tilde{\pi},sn} \xleftarrow{\$} \mathcal{R}$
  //*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 $\big(c^0, n, sn, \rho_{sn} + \rho'_{sn}, \rho_{pk} + \rho'_{pk}\big)$

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

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

$\mathcal{U}$:
- Parse $c$ as $\big(c^0, \big(c^j = (c^j_{pk}, c^j_{cert}, \pi^j_{cert}, c^j_{sn}, \pi^j_{sn}, c^j_{tag}, \pi^j_{tag}, \tilde{c}^j_{sn}, \tilde{c}^j_{tag}, \tilde{\pi}^j_{sn}, \tilde{\pi}^j_{tag})\big)^i_{j=1}, n, sn, \rho_{sn}, \rho_{pk}\big)$ // *i could be 0*
- $\rho_{tag}, \nu_{tag}, \rho_{t-pf} \xleftarrow{\$} \mathcal{R}$
- $(tag, t - pf) \leftarrow \mathsf{T.TGen}(par_{\mathsf{T}}, sk_{\mathsf{T}}, n, sn')$
- $c_{tag} \leftarrow \mathsf{C.Cm}(ck, tag, \rho_{tag})$ ; $\tilde{c}_{tag} \leftarrow \mathsf{E.Enc}(ek, tag, \nu_{tag})$
- $\pi_{tag} \leftarrow \mathsf{C.Prv}_{tag}(ck, pk_{\mathsf{T}}, sn, sn', tag, t - pf, \rho_{pk}, \rho_{sn}, \rho'_{sn}, \rho_{tag}, \rho_{t-pf})$
- $\tilde{\pi}_{tag} \leftarrow \mathsf{C.E.Prv}_{\mathrm{enc}}(ck, ek, tag, \rho_{tag}, \nu_{tag}, \tilde{c}_{tag})$
- 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 occur then abort and output $\perp$:
  - $\mathsf{VER}_{\mathrm{init}}(ek', vk, vk', c^0) = 0$
  - $\mathsf{VER}_{\mathrm{std}}(ek, vk, vk', c^{j-1}, c^j) = 0$, for some $j = 1, \ldots, i$
  - $\mathsf{C.Verify}_{\mathrm{tag}}(ck, c^i_{pk}, c^i_{sn}, c'_{sn}, c_{tag}, \pi_{tag}) = 0$
  - $\mathsf{C.E.Verify}_{\mathrm{enc}}(ck, ek, c_{tag}, \tilde{c}_{tag}, \tilde{\pi}_{tag}) = 0$
- pick uniform random $\vec{\rho''}$
- $c'' \leftarrow \mathsf{Rand}\big(((c^j)^i_{j=0}, c'_{pk}, c'_{cert}, \pi'_{cert}, c'_{sn}, \pi'_{sn}, c_{tag}, \pi_{tag}, \tilde{c}'_{sn}, \tilde{\pi}'_{sn}, \tilde{c}'_{tag}, \tilde{\pi}'_{tag}), \vec{\rho''}\big)$
- Output $\big(c'', n', sn', \rho'_{sn} + (\vec{\rho''})_{sn'}, \rho'_{pk} + (\vec{\rho''})_{pk'}\big)$

$\underline{\mathsf{CheckDS}}\big(sk_{\mathcal{CK}} = (dk', dk), \mathcal{DCL}, \mathcal{UL}, c\big)$:
- Parse $c$ as $\big(c^0 = (c^0_{pk}, c^0_{cert}, \pi^0_{cert}, c^0_{sn}, \pi^0_{sn}, c^0_M, c_\sigma, \pi_\sigma, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}),$ $(c^j = (c^j_{pk}, c^j_{cert}, \pi^j_{cert}, c^j_{sn}, \pi^j_{sn}, c^j_{tag}, \pi^j_{tag}, \tilde{c}^j_{sn}, \tilde{\pi}^j_{sn}, \tilde{c}^j_{tag}, \tilde{\pi}^j_{tag}))^i_{j=1}, n, sn, \rho_{sn}, \rho_{pk}\big)$
- $\vec{sn} \leftarrow \big(\mathsf{E'.Dec}(dk', \tilde{c}^0_{sn}), \mathsf{E.Dec}(dk, \tilde{c}^1_{sn}), \ldots, \mathsf{E.Dec}(dk, \tilde{c}^i_{sn})\big)$
- $\vec{tag} \leftarrow \big(\mathsf{E.Dec}(dk, \tilde{c}^1_{tag}), \ldots, \mathsf{E.Dec}(dk, \tilde{c}^i_{tag})\big)$
- If for all $(\vec{sn'}, \vec{tag'}) \in \mathcal{DCL} : sn_0 \neq sn'_0$ // *initial SN of checked coin...* then return $\mathcal{DCL} \,\|\, (\vec{sn}, \vec{tag})$ // *... different from those of deposited coins*
- Else let $j$ be minimal so that $sn_j \neq sn'_j$ // *double-spent at j-th transfer*
- $(pk_{\mathsf{T}}, \Pi) \leftarrow \mathsf{T.Detect}\big(sn_j, sn'_j, tag_j, tag'_j, \mathcal{UL}\big)$
- Return $(pk_{\mathsf{T}}, \Pi)$

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

### 5.3 Security Analysis

**Theorem 7.** *Our transferable e-cash scheme is perfectly* **sound**.

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 $\mathsf{S}$, $\mathsf{S}'$ and $\mathsf{C}$, as well as perfect soundness of $\mathsf{C}$ and verifiability of $\mathsf{T}$.

Detailed proofs of the following theorems are given in the full version [BFQ20]

**Theorem 8.** *Let $\mathcal{N}$ be the nonce space and $\mathcal{S}$ be the space of signatures of scheme $\mathsf{S}$. Let $\mathcal{A}$ be an adversary that wins the* **unforgeability** *game with advantage $\epsilon$ and makes at most $d$ calls to* BDepo. *Suppose that $\mathsf{C}$ is perfectly sound and $(\mathcal{M} \cup \mathcal{S})$-extractable. Then there exist adversaries against the unforgeability of the signature schemes $\mathsf{S}$ and $\mathsf{S}'$ with advantages $\epsilon_{\text{sig}}$ and $\epsilon'_{\text{sig}}$, 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 $\mathsf{T}.\mathsf{Detect}$, by soundness of $\mathsf{C}$ and two-extractability of $\mathsf{T}$, this will output a pair $(pk, \Pi)$, 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 $\mathsf{C}$, extract an $\mathsf{S}'$-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 $\mathsf{S}'$.
(2) $q_W < |\mathcal{DCL}|$: If the adversary creates a valid coin that has not been withdrawn, then by soundness of $\mathsf{C}$, we can extract a signature by the bank on a new initial serial number and therefore break unforgeability of $\mathsf{S}$.

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

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

An incrimination proof in our e-cash scheme is simply an incrimination proof of the tag scheme $\mathsf{T}$. 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 $\mathsf{T}$. The term $\epsilon_{\text{m-ind}}$ comes from the fact that we first need to switch $\mathsf{C}$ to hiding mode, so we can simulate $\pi_{sn}$ and $\pi_{tag}$ 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 (*$\mathtt{c\text{-}an}$*) with advantage $\epsilon$ and let $k$ be an upper-bound on the number of users transferring the challenge coins. Then there exist adversaries against mode-indistinguishability of* $\mathsf{C}$ *and tag-anonymity of* $\mathsf{T}$ *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 (*$\mathtt{u\text{-}an}$*) with advantage $\epsilon$ and let $k$ be a bound on the number of users transferring the challenge coin. Then there exist adversaries against mode-indistinguishability of* $\mathsf{C}$ *and tag-anonymity of* $\mathsf{T}$ *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 proofs reveal nothing either) and the only information contained in a coin are the serial numbers and tags. They are encrypted, but the adversary, impersonating the bank, can decrypt them.

We then argue that, by tag anonymity of $\mathsf{T}$, 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 switch can be detected by the adversary with probability at most $\epsilon_{\text{t-an}}$. Note that the additional factor 2 for $\epsilon_{\text{t-an}}$ in game $\mathtt{c\text{-}an}$ is due to the fact that there are two coins for which we switch users, whereas there is only one in game $\mathtt{u\text{-}an}$.

**Theorem 12.** *Let $\mathcal{A}$ be an adversary that wins the* **coin-transparency** *game (*$\mathtt{c-tr}$*) with advantage $\epsilon$, let $\ell$ be the size of the two 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* $\mathsf{C}$*, tag-anonymity of* $\mathsf{T}$*, IACR-security of* $\mathsf{E}$ *and RCCA-security of* $\mathsf{E}'$ *with advantages $\epsilon_{\text{m-ind}}$, $\epsilon_{\text{t-an}}$, $\epsilon_{\text{iacr}}$ and $\epsilon_{\text{rcca}}$, resp., such that*

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

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 $\mathtt{c\text{-}an}$ and $\mathtt{u\text{-}an}$, 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 $\mathsf{E}$. (Note that in the game these ciphertexts never need to be decrypted.) The factor $2\ell$ is due to the fact that there are at most $\ell$ 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 $\mathsf{E}'$: the oracle $\mathtt{Depo}'$ can be simulated by using the reduction's own oracles $\mathsf{Dec}$ and $\mathsf{GDec}$ (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 $\mathsf{GDec}$ outputs $\mathtt{replay}$, the reduction knows that a challenge coin was deposited, and uses this information to increase *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 $\mathsf{C}$ will be instantiated with the SXDH-based instantiation of Groth-Sahai proofs [GS08].

**Theorem 13 ([GS08]).** *The Groth-Sahai proof system, allowing to commit values from $\mathcal{V} := \mathbb{Z}_p \cup \mathbb{G} \cup \hat{\mathbb{G}}$ is perfectly complete, perfectly sound and randomizable; it is $(\mathbb{G} \cup \hat{\mathbb{G}})$-extractable, mode-indistinguishable assuming SXDH, and perfectly hiding in hiding mode.*

We note that moreover, all our proofs can be made zero-knowledge [GS08], and thus simulatable, 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 no scalars) in our reductions. (Note that for information-theoretic arguments concerning soundness, $\mathsf{Extr}$ can also be inefficient.)

**Signature Schemes.** For efficiency and type-compatibility reasons, we use two different signature schemes. The first one, $\mathsf{S}$, must support the functionality $\mathsf{SigCm}$, which imposes a specific format of messages. The second scheme, $\mathsf{S}'$, is less restrictive, which allows for more efficient instantiations. While all our other components rely on standard assumptions, we instantiate $\mathsf{S}$ with a scheme that relies on a non-interactive $q$-type assumption defined in [AFG+10].

**Theorem 14.** *The signature scheme from [AFG+10, Sect. 4] with message space $\mathcal{M} := \{(g^m, \hat{g}^m) \mid m \in \mathbb{Z}_p\}$ is (strongly) unforgeable assuming q-ADHSDH and AWFCDH (see [BFQ20]), and it supports the* $\mathsf{SigCm}$ *functionality [Fuc11].*

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

**Randomizable Encryption Schemes.** To instantiate the RCCA-secure scheme $\mathsf{E}'$ we follow the approach by Libert et al. [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 the full version [BFQ20].

Recall that the RCCA-secure scheme $\mathsf{E}'$ is only used to encrypt the initial part of the serial number; using a less efficient scheme thus has a minor impact on the efficiency of our scheme. From all other ciphertexts contained in a coin (which are under scheme $\mathsf{E}$) we only require IACR security, which standard ElGamal encryption satisfies under DDH(!). Thus, we instantiate $\mathsf{E}$ with ElGamal vector encryption. (Note that our instantiation of $\mathsf{E}'$ is also built on top of ElGamal). We prove the following in the full version [BFQ20].

**Theorem 16.** *Assuming SXDH, our randomizable encryption scheme [BFQ20] is RCCA-secure and ElGamal vector encryption 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 previously. We defer this to the full version [BFQ20].

### Efficiency Analysis

We conclude by summarizing the sizes of objects in our scheme in the table below and refer to the full version [BFQ20] for the details of our analysis.

For a group $G \in \{\mathbb{G}, \hat{\mathbb{G}}, \mathbb{Z}_p\}$, let $|G|$ denote the size of an element of $G$. Let $c_{\text{btsrap}}$ denote the coin output by $\mathcal{U}$ at the end of the Withdraw protocol (which corresponds to $c_{\text{init}}$ plus secret values, like $n$, $\rho_{sn}$, etc., to be used when transferring the coin), and let $c_{\text{std}}$ denote one (non-initial) component of the coin. 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. Abe, M., Fuchsbauer, G., Groth, J., Haralambiev, K., Ohkubo, M.: Structure-preserving signatures and commitments to group elements. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 209–236. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_12

AGHO11. Abe, M., Groth, J., Haralambiev, K., Ohkubo, M.: Optimal structure-preserving signatures in asymmetric bilinear groups. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 649–666. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_37

BCC+09. Belenkiy, M., Camenisch, J., Chase, M., Kohlweiss, M., Lysyanskaya, A., Shacham, H.: Randomizable proofs and delegatable anonymous credentials. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 108–125. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03356-8_7

BCF+11. Blazy, O., Canard, S., Fuchsbauer, G., Gouget, A., Sibert, H., Traoré, J.: Achieving optimal anonymity in transferable e-cash with a judge. In: Nitaj, A., Pointcheval, D. (eds.) AFRICACRYPT 2011. LNCS, vol. 6737, pp. 206–223. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21969-6_13

BCFK15. Baldimtsi, F., Chase, M., Fuchsbauer, G., Kohlweiss, M.: Anonymous transferable e-cash. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 101–124. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46447-2_5

BCG+14. Eli, B.-S., et al.: Zerocash: decentralized anonymous payments from Bitcoin. In: IEEE S&P 2014 (2014)

BCKL09. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: Compact e-cash and simulatable VRFs revisited. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 114–131. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03298-1_9

BFQ20. Bauer, B., Fuchsbauer, G., Qian, C.: Transferable e-cash: a cleaner model and the first practical instantiation. Cryptology ePrint Archive, Report 2020/1400 (2020)

Bla08. Blanton, M.: Improved conditional e-payments. In: Bellovin, S.M., Gennaro, R., Keromytis, A., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 188–206. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68914-0_12

BPS19. Bourse, F., Pointcheval, D., Sanders, O.: Divisible e-cash from constrained pseudo-random functions. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 679–708. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34578-5_24

Bra93. Brands, S.: Untraceable off-line cash in wallet with observers. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 302–318. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48329-2_26

CFN88. Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 319–327. Springer, New York (1990). https://doi.org/10.1007/0-387-34799-2_25

CG08. Canard, S., Gouget, A.: Anonymity in transferable e-cash. In: Bellovin, S.M., Gennaro, R., Keromytis, A., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 207–223. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68914-0_13

CGT08. Canard, S., Gouget, A., Traoré, J.: Improvement of efficiency in (unconditional) anonymous transferable e-cash. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 202–214. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85230-8_19

Cha83. Chaum, D.: Blind signature system. In: Chaum, D. (ed.) Advances in Cryptology. Springer, Boston (1984). https://doi.org/10.1007/978-1-4684-4730-9_14

CHL05. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_18

CKLM14. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Malleable signatures: new definitions and delegatable anonymous credentials. In: IEEE CSF 2014 (2004)

CKN03. Canetti, R., Krawczyk, H., Nielsen, J.B.: Relaxing chosen-ciphertext security. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 565–582. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_33

CP93. Chaum, D., Pedersen, T.P.: Transferred cash grows in size. In: Rueppel, R.A. (ed.) EUROCRYPT 1992. LNCS, vol. 658, pp. 390–407. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-47555-9_32

CPST16. Canard, S., Pointcheval, D., Sanders, O., Traoré, J.: Divisible e-cash made practical. IET Inf. Secur. **10**(6), 332–347 (2016)

FFHR19. Faonio, A., Fiore, D., Herranz, J., Ràfols, C.: Structure-preserving and re-randomizable RCCA-secure public key encryption and its applications. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11923, pp. 159–190. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34618-8_6

FHY13. Fan, C.-I., Huang, V.S.-M., Yu, Y.-C.: User efficient recoverable off-line e-cash scheme with fast anonymity revoking. Math. Comput. Model. **58**(1–2), 227–237 (2013)

FOS19. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: a cryptographic investigation of Mimblewimble. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 657–689. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_22

FP09. Fuchsbauer, G., Pointcheval, D.: Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 132–149. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03298-1_10

FPV09. Fuchsbauer, G., Pointcheval, D., Vergnaud, D.: Transferable constant-size fair e-cash. In: Garay, J.A., Miyaji, A., Otsuka, A. (eds.) CANS 2009. LNCS, vol. 5888, pp. 226–247. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10433-6_15

Fuc11. Fuchsbauer, G.: Commuting signatures and verifiable encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 224–245. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20465-4_14

GS08. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_24

LPQ17. Libert, B., Peters, T., Qian, C.: Structure-preserving chosen-ciphertext security with shorter verifiable ciphertexts. In: Fehr, S. (ed.) PKC 2017. LNCS, vol. 10174, pp. 247–276. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54365-8_11

Max15. Maxwell, G.: Confidential transactions (2015). https://people.xiph.org/~greg/confidential_values.txt

MGGR13. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: anonymous distributed e-cash from Bitcoin. In: IEEE S&P 2013 (2013)

Nak08. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash (2008). bitcoin.org/bitcoin.pdf

OO89. Okamoto, T., Ohta, K.: Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 481–496. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_43

OO91. Okamoto, T., Ohta, K.: Universal electronic cash. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 324–337. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_27

Poe16. Poelstra, A.: Mimblewimble (2016). https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf

vS13. van Saberhagen, N.: Cryptonote v 2.0 (2013). https://cryptonote.org/whitepaper.pdf

Zec20. Zcash Protocol Specification (15 January 2020). https://zips.z.cash/protocol/protocol.pdf