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

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

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

Abstract. Transferable e-cash is the most faithful digital analog of physical cash, as it allows users to transfer coins between them 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.

Keywords: Transferable/offline e-cash, strong anonymity.

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 [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 in order to reveal 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 before 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 needs to link occurrences of the same coin when detecting 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 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 in order 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 one), coins can change appearance after every transfer. Users will thus not recognize a coin when they see it again later, that is, 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 cointained 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. Our contribution is two-fold:

Security model. We revisit the formal model for transferable e-cash, starting from [BCFK15], whose model was a refined version of earlier ones. We first give a definition of correctness, which was lacking in previous works. 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).

Instantiation. Our main contribution is a transferable e-cash scheme, which we prove satisfies our security model, and which is much more efficient than the only previous realization [BCFK15]. Unfortunately, the authors do not provide concrete numbers, as they use malleable signatures in a blackbox way. 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], they 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. 30). 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 Π, which can be verified using VfyGuilt.

ParamGen(1^λ), on input the security parameter λ 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 secret key of the receiver 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 $\mathcal{UL}$. In case of error, they both obtain $\perp$.

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

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 Π.

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

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

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

2.2 Correctness properties

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

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

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

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

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

(Note that correctness of CheckDS and VfyGuilt is implied by the security notions below.)

2.3 Security definitions

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

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

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

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

BRegist() plays the bank side of Register and interacts with $\mathcal{A}$. If successful, it adds $(pk, \perp, uds = 0)$ 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 $\mathsf{Withdraw}$ 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 $\mathsf{Withdraw}$ when the bank is controlled by the adversary. Upon obtaining a coin c, it adds $(\mathit{owner}{=}i, c, \mathit{cds}{=}0, \mathit{origin}{=}\mathcal{A})$ to $\mathcal{CL}$.

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

Spend and deposit oracles.

`Spd`(j) spends the coin from the j-th entry $(\mathit{owner}_j, c_j, \mathit{cds}_j, \mathit{origin}_j)$ 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 $\mathit{sk}_{\mathit{owner}_j}$. It increments the coin spend counter cds_j by 1. If afterwards $\mathit{cds}_j > 1$, then the owner's double-spending counter $\mathit{uds}_{\mathit{owner}_j}$ is incremented by 1.

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

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

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

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

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

2.4 Economic properties

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

The following property was not required in any previous formalization of transferable e-cash in the literature and is analogous the property *clearing* defined for classical e-cash [BPS19].

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

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

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 $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{sound}}(\lambda) := \Pr[\mathbf{Expt}_{\mathcal{A}}^{\mathsf{sound}}(\lambda) = 1]$ is negligible in λ.*

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

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

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

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

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

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

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

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

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

2.5 Anonymity properties

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

As transferred coins necessarily grow in size [CP93], we can only guarantee indistinguishability of *comparable* coins. We therefore define $\texttt{comp}(c_1, c_2) = 1$ iff $\texttt{size}(c_1) = \texttt{size}(c_2)$, where $\texttt{size}(c) = 1$ 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 transfered before being given back to the adversary.

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

User anonymity. Coin anonymity required that users who transfer the coin are honest. If one of the users through which the coin passes colluded with the bank,

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

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

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

there would be a trivial attack: after receiving the two challenge coins, the bank simulates the deposit of one of them and the deposit of the coin intercepted by the colluding user. If a double-spending is detected, it knows that the received coin corresponds to the sequence of users which the colluder was part of.

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

Coin transparency. This is in some sense the strongest anonymity notion and it implies that a user that transfers a coin cannot recognize it if 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 transfered to an honest user. Now if the deposit does not succeed because of double-spending, the adversary knows that it was the first coin that was returned to him. Again, this attack is unavoidable due to the necessity of

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

$\mathsf{CheckDS}'(sk_{\mathcal{CK}}, \mathcal{UL}, \mathcal{DCL}, c, \mathcal{DCL}')$:     // used by $\texttt{BDepo}'$
    $x \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}', c)$
    If $x = \bot$:   // 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)

double-spending detection. It is a design choice that lies outside of our model to implement sufficient deterrence from double-spending, so it would exceed the utility of breaking anonymity.

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

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

3 Comparison with previous work

3.1 Model comparison

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

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

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

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

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

Our interpretation of the definition in [BCFK15] is that it does not distinguish the second and the third case. This is an issue, as the second case cannot be avoided (and must be dealt with outside the model, e.g. by having users sign their messages). But the third case *should* be avoided so the bank does not lose money without being able to accuse the cheater. This is now guaranteed by our unforgeability notion in Def. 2.

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

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

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

The earlier notion `StR-fa` [CG08, BCF$^+$11] is similar to our coin transparency `c-tr`, with the following differences: in `StR-fa`, when the adversary deposits a coin, the bank provides a guilt proof when it can; and `StR-fa` lets the adversary obtain user secret keys. Coin transparency would imply `StR-fa` if $\mathsf{CheckDS}$ 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 $\mathsf{CheckDS}$ could return $\perp$ because it cannot accuse anyone in $\mathcal{UL}$.

Moreover, no previous scheme, including [BCFK15] achieves `StR-fa`, as we show next.

3.2 A flaw in a proof in BCFK15

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

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`, and it can therefore be seen as the first scheme to satisfy the "spirit" of `StR-fa`, which is captured by our notion `c-tr`.

4 Primitives used in our construction

4.1 Bilinear groups

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

4.2 Randomizable proofs of knowledge and signatures

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

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

$\mathsf{C.Setup}(Gr)$ takes as input a description of a bilinear group and returns a commitment key ck, which implicitly defines the sets $\mathcal{V}, \mathcal{C}, \mathcal{R}$ and $\mathcal{E}$.

$\mathsf{C.ExSetup}(Gr)$ returns an extraction key xk in addition to a commitment key ck.

$\mathsf{C.SmSetup}(Gr)$ returns a commitment key ck and a simulation trapdoor td.

$\mathsf{C.Cm}(ck, v, \rho)$, on input a key ck, a value $v \in \mathcal{V}$ and randomness $\rho \in \mathcal{R}$, returns a commitment in $\mathcal{C}$.

$\mathsf{C.ZCm}(ck, \rho)$, used when simulating proofs, is defined as $\mathsf{C.Cm}(ck, 0_{\mathcal{V}}, \rho)$.

$\mathsf{C.RdCm}(ck, c, \rho)$ randomizes a commitment c to a fresh c' using randomness ρ.

$\mathsf{C.Extr}(xk, c)$, on input extraction key xk and a commitment c, outputs a value in $\mathcal{V}$. (This is the only algorithm that might not be polynomial-time.)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 $M \in \mathcal{M}$; if $\rho, \rho' \xleftarrow{\$} \mathcal{R}$, then the following three are distributed equivalently:

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

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

Security properties

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

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

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

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

4.3 Rerandomizable encryption schemes

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

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

$\mathsf{E.KeyGen}(Gr)$, on input the group description, outputs an encryption key ek and a corresponding decryption key dk.

$\mathsf{E.Enc}(ek, M, \nu)$ is probabilistic and on input an encryption key ek, a message M and (partial) randomness ν outputs a ciphertext.

$\mathsf{E.ReRand}(ek, C, \nu')$, on input an encryption key, a ciphertext and (partial) randomness, outputs a new ciphertext. If no randomness is explicitly given to $\mathsf{E.Enc}$ or $\mathsf{E.ReRand}$ then it is assumed to be chosen uniformly.

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

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

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

$\mathsf{E.Verify}(ek, M, \nu, C)$, on input an encryption key, a message, randomness and a ciphertext, outputs a bit.

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

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

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

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

then the following are equivalently distributed (with ρ'_ν is picked uniformly at random in $\mathcal{R}$):

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

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

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

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

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

Fig. 6. Security games for rerandomizable encryption schemes

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

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* $\mathtt{x}$-secure *if* $\Pr[\mathbf{Expt}^{\mathtt{x}}_{\mathcal{A},1}(\lambda) = 1] - \Pr[\mathbf{Expt}^{\mathtt{x}}_{\mathcal{A},0}(\lambda) = 1]$ *is negligible in* λ *for any PPT* $\mathcal{A}$*.*

4.4 Double-spending tag schemes

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Verifiability: For every $n, n' \in \mathcal{N}$, and after computing

- $(sk, pk) \leftarrow \mathsf{T.KeyGen}$; $(sk', pk') \leftarrow \mathsf{T.KeyGen}$
- $(sn, X) \leftarrow \mathsf{T.SGen}(sk, n)$ **or** $(sn, X) \leftarrow \mathsf{T.SGen}_{\mathrm{init}}(sk, n)$
- $(sn', sn\text{-}pf') \leftarrow \mathsf{T.SGen}(sk', n')$
- $(tag, t\text{-}pf) \leftarrow \mathsf{T.TGen}(sk, n, sn')$

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

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

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

then $pk_1 = pk_2$.

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

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

$O_2(sk, sn', i)$:
    If $T[i] = \perp$, 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

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:

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

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

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

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

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

Security properties.

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

Tag anonymity: Finally, our anonymity notions for transferable e-cash should hold even against a malicious bank, which gets to see the serial numbers and double-spending tags for deposited coins, and the *secret keys* of the users. Thus, we require 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* $\Pr[\mathbf{Expt}^{\texttt{tag-anon}}_{\mathcal{A},1}(\lambda) = 1] - \Pr[\mathbf{Expt}^{\texttt{tag-anon}}_{\mathcal{A},0}(\lambda) = 1]$ *is negligible in* λ *for any PPT* $\mathcal{A}$.

5 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 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 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 C to which we add compatible schemes: an $\mathcal{M}$-structure-preserving signature scheme S that admits an $\mathcal{M}$-commuting signature add-on SigCm, as well as a (standard) $\mathcal{M}'$-structure-preserving signature scheme S' (all defined in Sect. 4.2).

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

Auxiliary functions. To simplify the description of our scheme, we first define several auxiliary functions. We let 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_{T} and a committed message M (given the used randomness ρ_{pk}, ρ_{sn} and ρ_M); and $\mathsf{C.Verify}_{\mathrm{sn,init}}$ 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 $(\mathsf{C.Verify}(ck, \mathsf{T.SVfy}_{\mathrm{init}}(\cdot,\cdot,\cdot) = 1, c_{pk}, c_{sn}, c_M, \pi_{sn}))$

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

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

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

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

$\mathsf{C.Prv}_{\mathrm{tag}}$ 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.

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

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

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

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

$\mathsf{C.E.Prv}_{\mathrm{enc}}$ 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 π as in the input of $\mathsf{E.AdptPrf}$; moreover, since ρ_ν 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 $coin_{\mathrm{init}}$, and the *standard* components $coin_{\mathrm{std}}$. The first is of the form

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

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 σ on M; and $\tilde{c}_{sn}$ is an encryption of sn. Moreover, π_{cert} and π_{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" ε to pad so that both coin-component types have the same format. Validity of an initial component is verified w.r.t. an encryption key for E' and two signature verification keys for S and S':

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

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

Standard components of a coin are of the form

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

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

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

Return 1 iff the following hold:

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

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 is an implicit input to all other algorithms; we assume that they parse par as $(1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$.

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

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

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

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

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

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

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

$\mathcal{U}$:

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

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

$\mathcal{U}$:  – if $\mathsf{C.Verify}(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_M, c_\sigma, \pi_\sigma)$ fails, abort and output $\perp$.
    – $\nu_{sn} \xleftarrow{\$} \mathcal{R}$ ; $\tilde{c}_{sn} \leftarrow \mathsf{E}'.\mathsf{Enc}(ek', sn, \nu_{sn})$
    – $\tilde{\pi}_{sn} \leftarrow \mathsf{C.E}'.\mathsf{Prv}_{\mathrm{enc}}(ck, ek', sn, \rho_{sn}, \nu_{sn}, \tilde{c}_{sn})$
    – $\rho'_{\mathrm{pk}}, \rho'_{\mathrm{cert}}, \rho'_{\mathrm{sn}}, \rho'_M, \rho'_\sigma, \nu'_{\mathrm{sn}}, \rho'_{\tilde{\pi},\mathrm{sn}} \xleftarrow{\$} \mathcal{R}$     // since $\tilde{\pi}_{sn}$ contains a commitment,
                                                   we also sample randomness for it
    – $c^0 \leftarrow \mathsf{Rand}\big((c_{pk}, c_{\mathrm{cert}}, \pi_{\mathrm{cert}}, c_{\mathrm{sn}}, \pi_{\mathrm{sn}}, c_M, c_\sigma, \pi_\sigma, \tilde{c}_{\mathrm{sn}}, \tilde{\pi}_{\mathrm{sn}}),$
                                $(\rho'_{\mathrm{pk}}, \rho'_{\mathrm{cert}}, \rho'_{\mathrm{sn}}, \rho'_M, \rho'_\sigma, \nu'_{\mathrm{sn}}, \rho'_{\tilde{\pi},\mathrm{sn}})\big)$
    – Output $\big(c^0, n, sn, \rho_{sn} + \rho'_{sn}, \rho_{pk} + \rho'_{pk}\big)$

Spend$\langle \mathcal{U}(c, sk_{\mathcal{U}} = (cert, pk_{\mathsf{T}}, sk_{\mathsf{T}}), pk_{\mathcal{B}} = (ek', ek, vk, vk')),$
                                   $\mathcal{U}'(sk'_{\mathcal{U}} = (cert', pk'_{\mathsf{T}}, sk'_{\mathsf{T}}), pk_{\mathcal{B}})\rangle$:
$\mathcal{U}'$: – $n' \xleftarrow{\$} \mathcal{N}$ ; $\rho'_{pk}, \rho'_{cert}, \rho'_{sn}, \rho'_{sn\text{-}pf}, \nu'_{sn} \xleftarrow{\$} \mathcal{R}$
    – $(sn', sn\text{-}pf') \leftarrow \mathsf{T.SGen}(par_{\mathsf{T}}, sk'_{tag}, n')$
    – $c'_{pk} \leftarrow \mathsf{C.Cm}(ck, pk'_{\mathsf{T}}, \rho'_{pk})$ ; $c'_{cert} \leftarrow \mathsf{C.Cm}(ck, cert', \rho'_{cert})$
    – $c'_{sn} \leftarrow \mathsf{C.Cm}(ck, sn', \rho'_{sn})$ ; $c'_{sn\text{-}pf} \leftarrow \mathsf{C.Cm}(ck, sn\text{-}pf', \rho'_{sn\text{-}pf})$
    – $\tilde{c}'_{sn} \leftarrow \mathsf{E.Enc}(ek, sn', \nu'_{sn})$
    – $\pi'_{cert} \leftarrow \mathsf{C.Prv}(ck, \mathsf{S.Verify}(vk', \cdot, \cdot) = 1, (pk'_{\mathsf{T}}, \rho'_{pk}), (cert', \rho'_{cert}))$
    – $\pi'_{sn} \leftarrow \mathsf{C.Prv}_{\mathrm{sn}}(ck, pk'_{\mathsf{T}}, sn', sn\text{-}pf, \rho'_{pk}, \rho'_{sn}, \rho'_{sn\text{-}pf})$
    – $\tilde{\pi}'_{sn} \leftarrow \mathsf{C.E.Prv}_{\mathrm{enc}}(ck, ek, sn', \rho'_{sn}, \nu'_{sn}, \tilde{c}'_{sn})$
    – 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
    – $\rho_{tag}, \nu_{tag}, \rho_{t\text{-}pf} \xleftarrow{\$} \mathcal{R}$
    – $(tag, t\text{-}pf) \leftarrow \mathsf{T.TGen}(par_{\mathsf{T}}, sk_{\mathsf{T}}, n, sn')$
    – $c_{tag} \leftarrow \mathsf{C.Cm}(ck, tag, \rho_{tag})$ ; $\tilde{c}_{tag} \leftarrow \mathsf{E.Enc}(ek, tag, \nu_{tag})$
    – $\pi_{tag} \leftarrow \mathsf{C.Prv}_{tag}(ck, pk_{\mathsf{T}}, sn, sn', tag, t\text{-}pf, \rho_{pk}, \rho_{sn}, \rho'_{sn}, \rho_{tag}, \rho_{t\text{-}pf})$
    – $\tilde{\pi}_{tag} \leftarrow \mathsf{C.E.Prv}_{\mathrm{enc}}(ck, ek, tag, \rho_{tag}, \nu_{tag}, \tilde{c}_{tag})$
    – 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 fail then abort and output $\perp$:
        – $\mathsf{VER}_{\mathrm{init}}(ek', vk, vk', c^0)$
        – $\mathsf{VER}_{\mathrm{std}}(ek, vk, vk', c^{j-1}, c^j)$, for $j = 1, \ldots, i$
        – $\mathsf{C.Verify}_{\mathrm{tag}}(ck, c^i_{pk}, c^i_{sn}, c'_{sn}, c_{tag}, \pi_{tag})$
        – $\mathsf{C.E.Verify}_{\mathrm{enc}}(ck, ek, c_{tag}, \tilde{c}_{tag}, \tilde{\pi}_{tag})$
    – 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 $\big(c^0 = (c^0_{pk}, c^0_{cert}, \pi^0_{cert}, c^0_{sn}, \pi^0_{sn}, c^0_M, c_\sigma, \pi_\sigma, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}),$
      $(c^j = (c^j_{pk}, c^j_{cert}, \pi^j_{cert}, c^j_{sn}, \pi^j_{sn}, c^j_{tag}, \pi^j_{tag}, \tilde{c}^j_{sn}, \tilde{\pi}^j_{sn}, \tilde{c}^j_{tag}, \tilde{\pi}^j_{tag}))^i_{j=1}, n, \mathrm{sn}, \rho_{sn}, \rho_{\mathrm{pk}}\big)$
  – $\vec{sn} \leftarrow \big(\mathsf{E}'.\mathsf{Dec}(dk', \tilde{c}^0_{sn}), \mathsf{E}.\mathsf{Dec}(dk, \tilde{c}^1_{sn}), \ldots, \mathsf{E}.\mathsf{Dec}(dk, \tilde{c}^i_{sn})\big)$
  – $\vec{tag} \leftarrow \big(\mathsf{E}.\mathsf{Dec}(dk, \tilde{c}^1_{tag}), \ldots, \mathsf{E}.\mathsf{Dec}(dk, \tilde{c}^i_{tag})\big)$
  – If for all $(\vec{sn}', \vec{tag}') \in \mathcal{DCL}$: $(\vec{sn})_0 \neq (\vec{sn}')_0$        // 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 $(\vec{sn})_j \neq (\vec{sn}')_j$      // double-spent at $j$-th transfer
  – $(pk_{\mathsf{T}}, \varPi) \leftarrow \mathsf{T}.\mathsf{Detect}\big((\vec{sn})_j, (\vec{sn}')_j, (\vec{tag})_j, (\vec{tag}')_j, \mathcal{UL}\big)$
  – Return $(pk_{\mathsf{T}}, \varPi)$

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

5.3 Correctness and security analysis

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

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

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

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

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

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

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 C and two-extractability of T, this will output a pair $(pk, \varPi)$, such that $\mathsf{VfyGuilt}(pk, \varPi) = 1$. Since a coin contains a commitment to a certificate for the used tag key (and proofs of validity), we can, again by soundness of C, extract an S'-signature on

pk. Now if pk is not in $\mathcal{UL}$, then it was never signed by the bank, and $\mathcal{A}$ has thus broken unforgeability of S'.
(2) $q_W < |\mathcal{DCL}|$: If the adversary creates a valid coin that has not been withdrawn, then by soundness of C, we can extract a signature by the bank on a new initial serial number and therefore break unforgeability of S.

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

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

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

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

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

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

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

In the proof of both theorems, we first define a hybrid game in which the commitment key is switched to hiding mode (hence the loss $\epsilon_{\text{m-ind}}$, which occurs twice for $b = 0$ and $b = 1$). All commitments are then perfectly hiding (and 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 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 `c-an` is due to the fact that there are two coins for which we switch users, whereas there is only one in game `u-an`.

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

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

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 $\texttt{c-an}$ and $\texttt{u-an}$, the reduction first makes all commitments perfectly hiding and proofs perfectly simulatable (which loses $\epsilon_{\text{m-ind}}$ twice). Since all ciphertexts in the challenge coin given to the adversary are randomized, the reduction can replace all of them, except the initial one, by IACR-security of E. (Note that in the game these ciphertexts never need to be decrypted.) The factor 2ℓ is due to the fact that there are at most ℓ encryptions of SN/tag *pairs*. Finally, replacing the initial ciphertext (the one that enables detection of double-spending) can be done by a reduction to RCCA-security of E': the oracle $\texttt{Depo}'$ can be simulated by using the reduction's own oracles Dec and GDec (depending on whether $\texttt{Depo}'$ is called before or after the reduction receives the challenge ciphertext) in the RCCA-security game. Note that, when during a simulation of $\mathsf{CheckDS}$, oracle GDec outputs $\texttt{replay}$, 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 C will be instantiated with Groth-Sahai proofs [GS08], of which we use the instantiation based on SXDH (defined in Appendix D).

Theorem 13 ([GS08]). *The Groth-Sahai scheme, 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, Extr can also be inefficient.)

Signature schemes. For efficiency and type-compatibility reasons, we use two different signature schemes. The first one, S, must support the functionality SigCm, which imposes a specific format of messages. The second scheme, S', is less restrictive, which allows for more efficient instantiations. While all our other components rely on standard assumptions, we instantiate 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 $\mathcal{M} := \{(g^m, \hat{g}^m) \,|\, m \in \mathbb{Z}_p\}$ is (strongly) unforgeable assuming q-ADHSDH and AWFCDH (see Appendix D), and it supports the* 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 E' we follow the approach from 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 Appendix B.2.

Recall that the RCCA-secure scheme E' is only used to encrypt the initial part of the serial number; using a less efficient scheme does thus not have a big impact on the efficiency of our scheme. From all other ciphertexts contained in a coin (which are under scheme E) we only require IACR security, which standard ElGamal encryption satisfies under DDH(!). Thus, we instantiate E with ElGamal vector encryption. (Note that our instantiation of E' is also built on top of ElGamal). We prove the following in the appendix.

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

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

Efficiency analysis

We conclude by summarizing the sizes of objects in our scheme in the table below and refer to Appendix C 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_{btsrap} denote the coin output by $\mathcal{U}$ at the end of the $\mathsf{Withdraw}$ protocol (which corresponds to c_{init} plus secret values, like n, ρ_{sn}, etc., to be used when transferring the coin), and let c_{std} denote one (non-initial) component of the coin. After k transfers the size of a coin is $|c_{\mathrm{btsrap}}| + k|c_{\mathrm{std}}|$.

$\lvert sk_{\mathcal{B}}\rvert$	$9\lvert\mathbb{Z}_p\rvert + 2\lvert\mathbb{G}\rvert + 2\lvert\hat{\mathbb{G}}\rvert$
$\lvert pk_{\mathcal{B}}\rvert$	$15\lvert\mathbb{G}\rvert + 8\lvert\hat{\mathbb{G}}\rvert$
$\lvert sk_{\mathcal{U}}\rvert$	$\lvert\mathbb{Z}_p\rvert + 2\lvert\mathbb{G}\rvert + 2\lvert\hat{\mathbb{G}}\rvert$
$\lvert pk_{\mathcal{U}}\rvert$	$\lvert\hat{\mathbb{G}}\rvert$

$\lvert \Pi_{\mathrm{guilt}}\rvert$	$2\lvert\mathbb{G}\rvert$
$\lvert c_{\mathrm{btstrap}}\rvert$	$6\lvert\mathbb{Z}_p\rvert + 147\lvert\mathbb{G}\rvert + 125\lvert\hat{\mathbb{G}}\rvert$
$\lvert c_{\mathrm{std}}\rvert$	$54\lvert\mathbb{G}\rvert + 50\lvert\hat{\mathbb{G}}\rvert$
$\lvert(\vec{sn}, \vec{tag})\rvert$	$(4t+2)\lvert\mathbb{G}\rvert$

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

CKLM12. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. *EUROCRYPT'12*.

CKLM14. Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable signatures: New definitions and delegatable anonymous credentials. *IEEE CSF'14*.

CKN03. Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. *CRYPTO'03*.

CP93. David Chaum and Torben P. Pedersen. Transferred cash grows in size. *EUROCRYPT'92*

CPST16. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. *IET Inf. Security*, 10(6):332–347, 2016.

FFHR19. Antonio Faonio, Dario Fiore, Javier Herranz, and Carla Ràfols. Structure-preserving and re-randomizable RCCA-secure public key encryption and its applications. *ASIACRYPT'19*.

FHY13. Chun-I Fan, Vincent Shi-Ming Huang, and Yao-Chun Yu. User efficient recoverable off-line e-cash scheme with fast anonymity revoking. *Mathematical and Computer Modelling*, 58(1-2):227–237, 2013.

FOS19. Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. *EUROCRYPT'19*.

FP09. Georg Fuchsbauer and David Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. *PAIRING'09*.

FPV09. Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable constant-size fair e-cash. *CANS'09*.

Fuc11. Georg Fuchsbauer. Commuting signatures and verifiable encryption. *EUROCRYPT'11*.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. *EUROCRYPT'08*.

LPJY13. Benoît Libert, Thomas Peters, Marc Joye, and Moti Yung. Linearly homomorphic structure-preserving signatures and their applications. *CRYPTO'13*.

LPQ17. Benoît Libert, Thomas Peters, and Chen Qian. Structure-preserving chosen-ciphertext security with shorter verifiable ciphertexts. *PKC'17*.

Max15. Gregory Maxwell. Confidential Transactions, 2015. Available at https://people.xiph.org/~greg/confidential_values.txt.

MGGR13. Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. *IEEE S&P'13*.

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

OO89. Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. *CRYPTO'89*.

OO91. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. *CRYPTO'91*.

Poe16. Andrew Poelstra. Mimblewimble, 2016. Available at https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

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

Zec20. Zcash Protocol Specification 2020.1.15. https://zips.z.cash/protocol/protocol.pdf.

A Security proofs

Some of our theorems in the appendix are more general than stated in the body of the paper: they also work for a scheme C that only satisfies computational soundness (whereas in the body we assumed perfect soundness).

A.1 Unforgeability

Theorem 17. *Suppose that there exists an adversary $\mathcal{A}$ against unforgeability (Def. 2) of our transferable e-cash scheme with advantage ϵ_{unforg} making at most d calls to oracle* $\mathtt{BDepo}$*. Suppose that $\mathcal{M}$ and the signature space of S are contained in $\mathcal{V}'$. Then we can build a polynomial-time adversary $\mathcal{B}_1$ against the unforgeability of the signature scheme S with advantage ϵ_{sig}, an adversary $\mathcal{B}_2$ against the unforgeability of S' with advantage ϵ'_{sig}, and $\mathcal{B}_3$ and $\mathcal{B}_4$ against the soundness of the commitment scheme C with advantage $\epsilon_{h,1}$ and $\epsilon_{h,2}$. Then*

$$\epsilon_{\text{unforg}} \leq \epsilon_{h,1} + \epsilon_{h,2} + \epsilon_{\text{sig}} + \epsilon'_{\text{sig}} + \frac{d^2}{|\mathcal{N}|}.$$

Proof. Note that the adversary has two possibilities to win the game: either it creates a counterfeit (i.e., $q_W < |\mathcal{CL}|$), or it wins by making a deposit fail (i.e., $\mathsf{CheckDS}$ does neither output a list nor a valid pair with a registered user key). In our proof we will consider these two aspects separately. First we will prove in Proposition 18, that creating counterfeit is harder than breaking the unforgeability of S, or proving a false statement in C. In Proposition 19, we prove that if fresh nonces are picked during each deposits, then it is harder to make $\mathsf{Deposit}$ fail than breaking the unforgeability of S', or proving a false statement in C.

We first recall the unforgeability against the e-cash system:

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

and the unforgeability game against a signature scheme:

```
$\mathbf{Expt}^{\texttt{sig-uf}}_{\mathsf{S},\mathcal{B}}(\lambda)$
    $par \leftarrow \mathsf{S.Setup}\left(1^\lambda\right)$
    $(sk, vk) \leftarrow \mathsf{S.KeyGen}\,(par)$
    $Q := \emptyset$
    $(m, \sigma) \leftarrow \mathcal{B}^{\mathsf{S.Sign}^*(sk,\cdot,Q)}(par, vk)$
    Return $\big(m \notin Q \,\wedge\, \mathsf{S.Verify}(vk, m, \sigma)\big)$
```

```
Oracle: $\mathsf{S.Sign}^*\,(sk, m, Q)$:
    $Q := Q \cup \{m\}$
    Return $\mathsf{S.Sign}\,(sk, m)$
```

Finally, the soundness of the commitment scheme:

```
$\mathbf{Expt}^{\textsf{soundness}}_{\mathsf{C},\mathcal{B}}(\lambda)$:
    $(ck, xk) \leftarrow \mathsf{C.ExSetup}(1^\lambda)$
    $Q := \emptyset$; $(E, c_1, \dots, c_n) \leftarrow \mathcal{B}(ck)$
    Return $\big(\mathsf{C.Verify}\,(ck, E, c_1, \dots, c_n) \,\wedge\, \neg E\,(\mathsf{C.Extr}\,(xk, c_1), \dots, \mathsf{C.Extr}\,(xk, c_n))\,\big)$
```

Let E_{unforg} be the event that $\mathcal{A}$ wins the game, that is, at some point after a call to BDepo, CheckDS did not output a list, or $q_W < |\mathcal{DCL}|$. We partition E_{unforg} as follows:

- $E_{\text{Decrypt-fails}}$: In CheckDS, a decryption fails or does not output any serial number and tag, when it is supposed to
- E_{same}: In CheckDS, there is no j such that $\vec{sn}_j \neq \vec{sn}'_j$
- $E_{\text{DDS-fails}}$: In CheckDS, algorithm T.Detect does not output any (pk_{T}, Π_G)
- $E_{\text{incorrect}}$: CheckDS outputs (pk_{i^*}, Π_G) such that $\mathsf{VfyGuilt}\,(pk_{i^*}, \Pi_G) = 0$
- $E_{\text{not-register}}$: $pk_{i^*} \notin \mathcal{UL}$
- $E_{\text{counterfeit}}$: $q_W < |\mathcal{DCL}|$

We first build an adversary $\mathcal{B}_1$ against the unforgeability of S, which will bet on $E_{\text{counterfeit}}$: $q_W < |\mathcal{DCL}|$ (i.e., $\mathcal{A}$ creates valid money). Thus, $\mathcal{A}$ has produced a committed signature for a fresh serial number and thus forged a signature for a fresh serial number or a false proof for the equation S.Verify (In this second case adversary $\mathcal{B}_3$ will break soundness). Note that to simulate SigCm, adversary $\mathcal{B}_1$ needs xk and SmSigCm.

```
Adversary $\mathcal{B}_1^{\mathcal{A},\mathsf{S.Sign}^*(sk,\cdot)}\,(par_{\mathsf{S}}, vk)$:
    Obtain $Gr$ from $par_{\mathsf{S}}$
    $(ck, xk) \leftarrow \mathsf{C.ExSetup}\,(Gr)$
    $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}\,(Gr)$
    $\left(sk', vk'\right) \leftarrow \mathsf{S}'.\mathsf{KeyGen}\,(par_{\mathsf{S}'})$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}\,(Gr)$
    $par \leftarrow \left(1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck\right)$
    $\texttt{Coins}_{\texttt{D}} := \emptyset$
    $\texttt{Coins}_{\texttt{W}} := \emptyset$
    $(ek, dk) \leftarrow \mathsf{E.KeyGen}\,(Gr)$
    $\left(ek', dk'\right) \leftarrow \mathsf{E}'.\mathsf{KeyGen}\,(Gr)$
    $pk_{\mathcal{B}} \leftarrow \left(ek', ek, vk, vk'\right)$
    $(sk_{\mathsf{T}}, pk_{\mathsf{T}}) \leftarrow \mathsf{T.KeyGen}(Gr)$
    $sk_{\mathcal{D}} \leftarrow (\varepsilon, sk_{\mathsf{T}}, pk_{\mathsf{T}})$
    Run $\mathcal{A}^{\texttt{BRegist},\texttt{BWith}^*,\texttt{BDepo}}\left(par, pk_{\mathcal{B}}\right)$
    In each call of $\texttt{BWith}$, add the coin received to the list $\texttt{Coins}_{\texttt{W}}$
    In each call of $\texttt{BDepo}$, add the coin received to the list $\texttt{Coins}_{\texttt{D}}$
    $\texttt{BWith}^*$ is similar to $\texttt{BWith}$, except instead of using $\mathsf{SigCm}$, it uses $\mathsf{SigCm}^*$:
        $\mathsf{SigCm}^*(c)$:
            $m \leftarrow \mathsf{C.Extr}\left(xk, c\right)$
            Use the oracle to obtain $\Sigma \leftarrow \mathsf{S.Sign}^*(sk, m)$
            $(c_\sigma, \pi) \leftarrow \mathsf{SmSigCm}(xk, vk, c, \Sigma)$
            Return $(c_\sigma, \pi)$
    Let $q_W$ be the number of successful calls to $\texttt{BWith}$
    If $q_W \geq |\mathcal{DCL}|$ then abort
    Let $D := \emptyset$
    Let $W := \emptyset$
    For $c \in \texttt{Coins}_{\texttt{W}}$:
        Parse $c$ as $\left(c^0, (c^j)_{j=1}^i, n, sn, \rho_{sn}, \rho_{pk}\right)$
        Parse $c^0$ as $\left(c^0_{pk}, c^0_{cert}, \pi^0_{cert}, c^0_{sn}, \pi^0_{sn}, c_M, c^0_\sigma, \pi^0_\sigma, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}\right)$
        $M := \mathsf{C.Extr}\left(xk, c_M\right)$
        $\sigma := \mathsf{C.Extr}\left(xk, c^0_\sigma\right)$
        $W := W \cup \{(M, \sigma)\}$
    For $c \in \texttt{Coins}_{\texttt{D}}$:
        Parse $c$ as $\left(c^0, \left(c^j\right)_{j=1}^i, n, sn, \rho_{sn}, \rho_{pk}\right)$
        Parse $c^0$ as $\left(c^0_{pk}, c^0_{cert}, \pi^0_{cert}, c^0_{sn}, \pi^0_{sn}, c_M, c^0_\sigma, \pi^0_\sigma, \tilde{c}^0_{sn}, \tilde{\pi}^0_{sn}\right)$
        $M := \mathsf{C.Extr}\left(xk, c_M\right)$
        $\sigma := \mathsf{C.Extr}\left(xk, c^0_\sigma\right)$
        $D := D \cup \{(M, \sigma)\}$
    If $\exists\, (M, \sigma) \in W \setminus D$:
        then return $(M, \sigma)$
    Else abort
```

We let ε identify the part of the secret key that is ignored in the entire game (because the bank never spent a coin). By correctness of the committed signature of S, the simulation will be perfect. And by $(\mathcal{M} \cup \mathcal{S})$-extractability of C, we deduce that $\mathcal{B}_1$ is efficient.

We now construct a first adversary $\mathcal{B}_3$ against soundness:

```
Adversary $\mathcal{B}_3^{\mathcal{A}}\left(ck\right)$:
    Obtain $Gr$ from $ck$
    $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}\left(Gr\right)$
    $par_{\mathsf{S}'} \leftarrow \mathsf{S}'\mathsf{.Setup}\left(Gr\right)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}\left(Gr\right)$
    $par \leftarrow \left(1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck\right)$
    $S := \emptyset$
    $\left(pk_{\mathcal{B}} = \left(ek', ek, vk, vk'\right), sk_{\mathcal{W}} = (sk, sk'), sk_{\mathcal{D}}, sk_{\mathcal{CK}}\right) \leftarrow \mathsf{BKeyGen}()$
    Run $\mathcal{A}^{\texttt{BRegist},\texttt{BWith},\texttt{BDepo}}\left(par, pk_{\mathcal{B}}\right)$
    In each call of $\texttt{BDepo}$, add the entire coin received in the list $\texttt{Coins}_{\texttt{D}}$
    Let $q_W$ be the number of successful calls to $\texttt{BWith}$
```

```
If $q_W \geq |\mathcal{DCL}|$, then abort
Let $D := \emptyset$
For $c \in \mathtt{Coins}_{\mathtt{D}}$:
    Parse $c$ as $\left(c_0, (c_j)_{j=1}^{i}, n, sn, \rho_{sn}, \rho_{pk}\right)$
    Add $c_0$ to $S$
Parse $S$ as $\left\{\left(c^i_{pk_\mathsf{T}}, c^i_{cert}, \pi^i_{cert}, c^i_{sn}, \pi^i_{sn}, c^i_M, c^i_\sigma, \pi^i_\sigma, \tilde{c}^i_{sn}, \tilde{\pi}^i_{sn}\right)\right\}_{1\leq i\leq |S|}$
Parse $\tilde{\pi}^i_{sn}$ as $\left(c^i, \pi^i\right)$
Return $\Big( \bigwedge_{i=1}^{|S|} \Big(\mathsf{E}'.\mathsf{Verify}\left(ek', X_1^i, X_2^i, \tilde{c}^i_{sn}\right) \wedge \mathsf{S}.\mathsf{Verify}\left(vk, X_3^i, X_4^i\right) = 1 \wedge$
        $\mathsf{T}.\mathsf{SVfy}_{\mathrm{init}}\left(X_5^i, X_1^i, X_3^i\right)\Big),$
    $c^1_{sn}, c^1, c^1_M, c^1_\sigma, c^1_{pk_\mathsf{T}}, \ldots, c^{|S|}_{sn}, c^{|S|}, c^{|S|}_M, c^{|S|}_\sigma, c^{|S|}_{pk_\mathsf{T}}, \bigwedge_{i=1}^{|S|}\left(\pi^i \wedge \pi^i_\sigma \wedge \pi^i_{sn}\right)\Big)$
```

We define the following events:
E_{sig}: $\mathcal{B}_1$ breaks the unforgeability of the signature scheme S; and
$E_{\mathrm{com,1}}$: $\mathcal{B}_3$ breaks the soundness of the commitment scheme C.

Proposition 18. *$E_{counterfeit} \subset E_{sig} \cup E_{com,1}$.*

Suppose that we are in case $E_{\mathrm{counterfeit}} \setminus E_{\mathrm{sig}}$. Because the coin has been accepted during Spend in a call to $\mathtt{BDepo}$, the proofs output by $\mathcal{B}_3$ are correct. Let $\left(sn^1, \nu^1, M^1, pk^1_\mathsf{T}, \sigma^1, \ldots, sn^{|S|}, \nu^{|S|}, M^{|S|}, pk^{|S|}_\mathsf{T}, \sigma^{|S|}\right)$ be the values that the challenger of the soundness game extracts from the commitments output by $\mathcal{B}_3$. If for some i: $\mathsf{S}.\mathsf{Verify}\left(vk, M^i, \sigma^i\right) \neq 1$, then $\mathcal{B}_3$ wins the soundness game.

Suppose that $\bigwedge_{i=1}^{|S|} \mathsf{S}.\mathsf{Verify}\left(vk, M^i, \sigma^i\right) = 1$. Since we are not in E_{sig}, all values M^i correspond to coins that have been withdrawn. But there are only q_W such coins and thus q_W such messages M. Thus, we have $\left|\{M^i\}_{i=1}^{|S|}\right| \leq q_W$.

If $\bigwedge_{i=1}^{|S|} \mathsf{T}.\mathsf{SVfy}_{\mathrm{init}}\left(pk^i_\mathsf{T}, sn^i, M^i\right) \neq 1$, then $\mathcal{B}_3$ won the soundness game.

Assume $\bigwedge_{i=1}^{|S|} \mathsf{T}.\mathsf{SVfy}_{\mathrm{init}}\left(pk^i_\mathsf{T}, sn^i, M^i\right) = 1$. Since T is bootable, it must hold that $\left|\{sn^i\}_{i=1}^{|S|}\right| \leq \left|\{M^i\}_{i=1}^{|S|}\right|$, from which we get $\left|\{sn^i\}_{i=1}^{|S|}\right| \leq q_W < |\mathcal{DCL}|$ (the last inequality follows since we assumed to be in $E_{\mathrm{counterfeit}}$).

Note that by construction of $\mathcal{DCL}$, all the initial serial numbers of the elements of $\mathcal{DCL}$ are different. Let us call this set I. From $|I| = |\mathcal{DCL}|$, we deduce $|I| > \left|\{sn^i\}_{i=1}^{|S|}\right|$. By construction, $|I| = \left|\{\mathsf{E}'.\mathsf{Dec}(dk', \tilde{c}^i_{sn})\}_{i=1}^{|S|}\right|$, and thus $\left|\{\mathsf{E}'.\mathsf{Dec}(dk', \tilde{c}^i_{sn})\}_{i=1}^{|S|}\right| > \left|\{sn^i\}_{i=1}^{|S|}\right|$. Let i_0 be such that $\mathsf{E}'.\mathsf{Dec}(dk', \tilde{c}^{i_0}_{sn}) \notin \left|\{sn^i\}_{i=1}^{|S|}\right|$. By correctness of E', we have $\mathsf{E}'.\mathsf{Enc}\left(pk, sn^{i_0}, \nu^{i_0}\right) \neq \tilde{c}^{i_0}_{sn}$, and thus (again by correctness of E') $\mathsf{E}'.\mathsf{Verify}\left(pk, sn^{i_0}, \nu^{i_0}, \tilde{c}^{i_0}_{sn}\right) \neq 1$.

We deduce that $\bigwedge_{i=1}^{|S|} \mathsf{E}'.\mathsf{Verify}\left(pk, sn^i, \nu^i, \tilde{c}^i_{sn}\right) \neq 1$, and consequently $\mathcal{B}_3$ won the soundness game. We thus have $E_{\mathrm{counterfeit}} \setminus E_{\mathrm{sig}} \subset E_{\mathrm{com,1}}$. □

We build an algorithm to break unforgeability of S':

```
Adversary $\mathcal{B}_2^{\mathcal{A},\mathsf{S}'.\mathsf{Sign}^*(sk',\cdot)}\left(par_{\mathsf{S}'}, vk'\right)$:
    Initialize $\mathcal{UL}$ as empty list
    Obtain $Gr$ from $par_{\mathsf{S}'}$
    $(ck, xk) \leftarrow \mathsf{C.ExSetup}(Gr)$
    $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$
    $(sk, vk) \leftarrow \mathsf{S.KeyGen}(Gr)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
    $par \leftarrow \left(1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck\right)$
    $\left(ek', dk'\right) \leftarrow \mathsf{E}'.\mathsf{KeyGen}(Gr)$
    $(ek, dk) \leftarrow \mathsf{E.KeyGen}(Gr)$
    $(sk_{\mathsf{T}}, pk_{\mathsf{T}}) \leftarrow \mathsf{T.KeyGen}(Gr)$
    $sk_{\mathcal{D}} \leftarrow (\varepsilon, pk_{\mathsf{T}}, sk_{\mathsf{T}})$
    $pk_{\mathcal{B}} \leftarrow \left(ek', ek, vk, vk'\right)$
    Run $\mathcal{A}^{\mathtt{BRegist},\mathtt{BWith},\mathtt{BDepo}}(par, pk_{\mathcal{B}})$
    Each time we would use $\mathsf{S}'.\mathsf{Sign}$ in an oracle call we use
        $\mathsf{S}'.\mathsf{Sign}^*\left(sk', \cdot\right)$, and we add the input to $\mathcal{UL}$
    Let $(pk, \mathit{\Pi})$ be the output of the last call to BDepo
    If such a pair is never returned by BDepo, then abort
    Let $c_1$ be the last coin sent by the user
    Parse $c_1$ as $\left(c^0, (c^j)_{j=1}^i, n, sn, \rho_{sn}, \rho_{pk}\right)$
    Let $j$ be minimal such that $(\vec{sn})_{j-1} \neq (\vec{sn'})_{j-1}$
        (using the notation from CheckDS)
    Parse $c^{j-1}$ as $\Big(c_{pk_{\mathsf{T}}}^{j-1}, c_{cert}^{j-1}, \pi_{cert}^{j-1}, c_{sn}^{j-1}, \pi_{sn}^{j-1}, \underline{c_{tag}^{j-1}}, \underline{\pi_{tag}^{j-1}}, c_M, c_\sigma^{j-1}, \pi_\sigma^{j-1},$
                                        $\tilde{c}_{sn}^{j-1}, \tilde{\pi}_{sn}^{j-1}, \underline{\tilde{c}_{tag}^{j-1}}, \underline{\tilde{\pi}_{tag}^{j-1}}\Big)$
    $pk_{\mathsf{T}} := \mathsf{C.Extr}\left(xk, c_{pk_{\mathsf{T}}}^{j-1}\right)$
    $\sigma := \mathsf{C.Extr}\left(xk, c_{cert}^{j-1}\right)$
    If $pk_{\mathsf{T}} \notin \mathcal{UL}$:
        then return $(pk_{\mathsf{T}}, \sigma)$
    Else abort
```

We denote by ε the part of the secret key that could be ignored in the protocols (e.g., the certificate *cert* of a receiver is never used). Let E'_{sig} be the event that $\mathcal{B}_2$ breaks the unforgeability of S'.

We construct a second adversary against soundness of C:

```
Adversary $\mathcal{B}_4^{\mathcal{A}}(ck)$:
    Obtain $Gr$ from $ck$
    $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$ ; $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$ ; $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
    $par \leftarrow \left(1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck\right)$
    $\left(pk_{\mathcal{B}} = \left(ek', ek, vk, vk'\right), sk_{\mathcal{W}} = (sk, sk'), sk_{\mathcal{D}}, sk_{\mathcal{CK}}\right) \leftarrow \mathsf{BKeyGen}()$
    Run $\mathcal{A}^{\mathtt{BRegist},\mathtt{BWith},\mathtt{BDepo}}(par, pk_{\mathcal{B}})$
    Let $(pk, \mathit{\Pi})$ be the output of the last call to BDepo
    If such a pair is never returned by BDepo, then abort
    Let $c$ be the last coin sent by the user and $i$ be its size
```

```
Parse $c$ as $\left(c^0, (c^k)_{k=1}^{i}, n, sn, \rho_{sn}, \rho_{pk}\right)$
Let $j$ be minimal such that $(\vec{sn})_{j-1} \neq (\vec{sn'})_{j-1}$
    (using the notation from $\mathsf{CheckDS}$)
Parse $c^{(j-1)}$ as $\big(c_{pk}^{(j-1)}, c_{cert}^{(j-1)}, \pi_{cert}^{(j-1)}, c_{sn}^{(j-1)}, \pi_{sn}^{(j-1)}, \underline{c_{tag}^{(j-1)}}, \underline{\pi_{tag}^{(j-1)}}, c_M,$
        $c_\sigma^{(j-1)}, \pi_\sigma^{(j-1)}, \tilde{c}_{sn}^{(j-1)}, \tilde{\pi}_{sn}^{(j-1)}, \underline{\tilde{c}_{tag}^{(j-1)}}, \underline{\tilde{\pi}_{tag}^{(j-1)}}\big)$
Do the same for all $k \in \{0, \dots, i\}$
If $j \neq 1$, then parse $\pi_{sn}^{(j-1)}$ as $\left(\pi_{sn,\mathrm{valid}}^{(j-1)}, c_{sn\text{-}pf}^{(j-1)}\right)$
    Else $\left(\pi_{sn,\mathrm{valid}}^{(j-1)}, c_{sn\text{-}pf}^{(j-1)}\right) \leftarrow \left(\pi_{sn}^{(j-1)}, c_M\right)$
Parse $\pi_{sn}^{j}$ as $\left(\pi_{sn,\mathrm{valid}}^{j}, c_{sn\text{-}pf}^{j}\right)$ and $\pi_{tag}^{j}$ as $\left(\pi_{tag,\mathrm{valid}}^{j}, c_{t\text{-}pf}^{j}\right)$
For all $k \in \{0, \dots, i\}$:
    Parse $\tilde{\pi}_{sn}^{k}$ as $\left(c_{\nu_{sn}}^{k}, \pi_{sn,\mathrm{eq}}^{k}\right)$ and parse $\tilde{\pi}_{tag}^{k}$ as $\left(c_{\nu_{tag}}^{k}, \pi_{tag,\mathrm{eq}}^{k}\right)$
Let $c'$ be the coin that collides with $c$ and $i'$ be its size
Parse $c'$ as $\left(c'^0, (c'^k)_{k=1}^{i'}, n', sn', \rho'_{sn}, \rho'_{pk}\right)$
Parse $c'^{(j-1)}$ as $\Big(c_{pk}'^{(j-1)}, c_{cert}'^{(j-1)}, \pi_{cert}'^{(j-1)}, c_{sn}'^{(j-1)}, \pi_{sn}'^{(j-1)}, \underline{c_{tag}'^{(j-1)}},$
        $\underline{\pi_{tag}'^{(j-1)}}, c_M'^{(j-1)}, c_\sigma'^{(j-1)}, \pi_\sigma'^{(j-1)}, \tilde{c}_{sn}'^{(j-1)}, \tilde{\pi}_{sn}'^{(j-1)}, \underline{\tilde{c}_{tag}'^{(j-1)}}, \underline{\tilde{\pi}_{tag}'^{(j-1)}}\Big)$
Do the same for all $k \in \{0, \dots, i'\}$
Parse $\pi_{sn}'^{j}$ as $\left(\pi_{sn,\mathrm{valid}}'^{j}, c_{sn\text{-}pf}'^{j}\right)$
Parse $\pi_{tag}'^{j}$ as $\left(\pi_{tag,\mathrm{valid}}'^{j}, c_{t\text{-}pf}'^{j}\right)$
Parse $\tilde{\pi}_{sn}'^{(j-1)}$ as $\left(c_{\nu_{sn}}'^{(j-1)}, \pi_{sn,\mathrm{eq}}'^{(j-1)}\right)$
Parse $\tilde{\pi}_{sn}'^{j}$ as $\left(c_{\nu_{sn}}'^{j}, \pi_{sn,\mathrm{eq}}'^{j}\right)$
Parse $\tilde{\pi}_{tag}'^{j}$ as $\left(c_{\nu_{tag}}'^{j}, \pi_{tag,\mathrm{eq}}'^{j}\right)$
Return $\Big(\mathsf{E}'.\mathsf{Verify}(ek', Y_0, Y_1, \tilde{c}_{sn}^{0}) \wedge \bigwedge_{k=1}^{i} \mathsf{E}.\mathsf{Verify}\left(ek, Y_{2k}, Y_{2k+1}, \tilde{c}_{sn}^{k}\right) \wedge$
    $\bigwedge_{k=1}^{i} \mathsf{E}.\mathsf{Verify}\left(ek, Y_{2i+2k}, Y_{2i+2k+1}, \tilde{c}_{tag}^{k}\right) \wedge$
    $\mathsf{S}'.\mathsf{Verify}\left(vk', X_1, X_2\right) = 1 \wedge$
    $\mathsf{E}.\mathsf{Verify}\left(ek, X_5, X_6, \tilde{c}_{sn}^{(j-1)}\right) \wedge \mathsf{E}.\mathsf{Verify}\left(ek, X_7, X_8, \tilde{c}_{sn}'^{(j-1)}\right) \wedge$
    $\mathsf{E}.\mathsf{Verify}\left(ek, X_9, X_{10}, \tilde{c}_{sn}^{j}\right) \wedge \mathsf{E}.\mathsf{Enc}\left(ek, X_{11}, X_{12}, \tilde{c}_{sn}'^{j}\right) \wedge$
    $\mathsf{T}.\mathsf{SVfy}_{\mathsf{all}}\left(X_1, X_5, X_{13}\right) = 1 \wedge$
    $\mathsf{T}.\mathsf{SVfy}\left(X_1, X_9, X_{14}\right) = 1 \wedge \mathsf{T}.\mathsf{SVfy}\left(X_3, X_{11}, X_{15}\right) = 1 \wedge$
    $\mathsf{E}.\mathsf{Verify}\left(ek, X_{16}, X_{17}, \tilde{c}_{tag}^{j}\right) \wedge \mathsf{E}.\mathsf{Enc}\left(ek, X_{18}, X_{19}, \tilde{c}_{tag}'^{j}\right) \wedge$
    $\mathsf{T}.\mathsf{TVfy}\left(X_1, X_5, X_9, X_{16}, X_{20}\right) = 1 \wedge$
    $\mathsf{T}.\mathsf{TVfy}\left(X_1, X_7, X_{11}, X_{18}, X_{21}\right) = 1,$
    $c_{sn}^{0}, c_{\nu_{sn}}^{0}, \dots, c_{sn}^{k}, c_{\nu_{sn}}^{k}, c_{tag}^{1}, c_{\nu_{tag}}^{1}, \dots, c_{tag}^{k}, c_{\nu_{tag}}^{k},$
    $c_{pk}^{(j-1)}, c_{cert}^{(j-1)}, c_{pk}^{j}, c_{pk}'^{j}, c_{sn}^{(j-1)}, c_{\nu_{sn}}^{(j-1)}, c_{sn}'^{(j-1)}, c_{\nu_{sn}}'^{(j-1)}, c_{sn}^{j}, c_{\nu_{sn}}^{j}, c_{sn}'^{j}, c_{\nu_{sn}}'^{j},$
    $c_{sn\text{-}pf}^{(j-1)}, c_{sn\text{-}pf}^{j}, c_{sn\text{-}pf}'^{j}, c_{tag}^{j}, c_{\nu_{tag}}^{j}, c_{tag}'^{j}, c_{\nu_{tag}}'^{j}, c_{t\text{-}pf}^{j}, c_{t\text{-}pf}'^{j},$
    $\bigwedge_{k=0}^{i} \pi_{sn,\mathrm{eq}}^{k} \bigwedge_{k=1}^{i} \pi_{tag,\mathrm{eq}}^{k} \wedge \pi_{cert}^{(j-1)} \wedge \pi_{sn,\mathrm{eq}}^{(j-1)} \wedge \pi_{sn,\mathrm{eq}}'^{(j-1)} \wedge \pi_{sn,\mathrm{eq}}^{j} \wedge \pi_{sn,\mathrm{eq}}'^{j} \wedge$
    $\pi_{sn,\mathrm{valid}}^{(j-1)} \wedge \pi_{sn,\mathrm{valid}}^{j} \wedge \pi_{sn,\mathrm{valid}}'^{j} \wedge \pi_{tag,\mathrm{eq}}^{j} \wedge \pi_{tag,\mathrm{eq}}'^{j} \wedge \pi_{tag,\mathrm{valid}}^{j} \wedge \pi_{tag,\mathrm{valid}}'^{j}\Big)$
```

where Y_j is the variable in the equations representing the purported values in the j-th commitment, and the X_i's are the last 21 variables.

We define the following two events:

$E_{\text{com},2}$: $\mathcal{B}_4$ breaks the soundness of C, and

$E_{\text{same-nonce}}$: the same nonce is picked twice by the bank during two different calls to `BDepo`.

Proposition 19. $E_{\text{unforg}} \setminus E_{\text{counterfeit}} \subset E_{\text{same}} \cup E_{\text{com},2} \cup E'_{\text{unforg}}$.

Suppose that we are in $E_{\text{unforg}} \setminus (E_{\text{com},2} \cup E_{\text{counterfeit}} \cup E_{\text{same}})$. Because the coin has been accepted, the proofs are correct (as they are verified in the Spend protocol, during a call to `BDepo`). We are thus in a case where the extracted commitment will verify the equations. Let

$$\big(sn^0, \dots, sn^i, tag^1, \dots, tag^i, pk_{tag}^{(j-1)}, cert^{(j-1)}, pk_{tag}^j, pk_{tag}'^j, sn^{(j-1)}, \nu_{sn}^{(j-1)}, sn'^{(j-1)}, \nu_{sn}'^{(j-1)}, sn^j, \nu_{sn}^j, sn'^j, \nu_{sn}'^j, sn\text{-}pf^{(j-1)}, sn\text{-}pf^j, sn\text{-}pf'^j, tag^j, \nu_{tag}^j, tag^{j\prime}, \nu_{tag}^{j\prime}, t\text{-}pf^j, t\text{-}pf'^j\big)$$

be what the challenger of the soundness game extracts from the commitments output by $\mathcal{B}_4$. Since we are not in $E_{\text{com},2}$ we have that $\mathcal{B}_4$ loses the game: for all $k \in \{1, \dots, i\}$:

$$\mathsf{E}'.\mathsf{Verify}\left(ek', sn^0, \nu_{sn}^0, \tilde{c}_{sn}^0\right) = \mathsf{E}.\mathsf{Verify}\left(ek, sn^k, \nu_{sn}^k, \tilde{c}_{sn}^k\right) = 1,$$

and for all $k \in \{1, \dots, i\}$:

$$\mathsf{E}.\mathsf{Verify}\left(ek, tag^k, \nu_{sn}^k, \tilde{c}_{tag}^k\right) = 1.$$

By correctness of E and E', we deduce that $E_{\text{Decrypt-fails}}$ will not happen. Being in $E_{\text{unforg}} \setminus E_{\text{counterfeit}}$ means that $\mathsf{CheckDS}$ detected that the first SN-component of c is the same as that of another coin (here c'). Note that the last sn of a deposited coin is generated (with the key sk_{T}) and encrypted by the bank itself. Now because we are not in E_{same}, we have that $\mathsf{CheckDS}$ will find some j, such that $\vec{sn}_j \neq \vec{sn'}_j$.

By construction, $\mathsf{E}.\mathsf{Dec}\left(dk, \tilde{c}_{sn}^j\right) = \mathsf{E}.\mathsf{Dec}\left(dk, \tilde{c}_{sn}'^j\right)$, which by correctness of E (and because we are not in $E_{\text{com, }2}$) means $sn^{(j-1)} = sn'^{(j-1)}$. Since

$$\begin{aligned}
1 &= \mathsf{T}.\mathsf{SVfy}\big(pk_{\mathsf{T}}^j, sn^j, sn\text{-}pf^j\big) \\
&= \mathsf{T}.\mathsf{SVfy}\big(pk_{\mathsf{T}}'^j, sn'^j, sn\text{-}pf'^j\big) \\
&= \mathsf{T}.\mathsf{TVfy}\big(pk_{\mathsf{T}}^{(j-1)}, sn^{(j-1)}, sn^j, tag^j, t\text{-}pf^j\big) \\
&= \mathsf{T}.\mathsf{TVfy}\big(pk_{\mathsf{T}}'^{(j-1)}, sn^{(j-1)}, sn'^j, tag'^j, t\text{-}pf'^j\big) \\
&= \mathsf{T}.\mathsf{SVfy}_{\text{all}}\big(pk_{\mathsf{T}}^{(j-1)}, sn^{(j-1)}, sn\text{-}pf^{(j-1)}\big),
\end{aligned}$$

and, because T is SN-identifiable, we get that $pk_{tag}^j = pk_{tag}'^j$.

Moreover, since T is two-extractable, we deduce that if $pk_{tag}^j \in \mathcal{UL}$, and that E_{DDSfails}, $E_{\text{incorrect}}$ and $E_{\text{not-register}}$ will not happen.

We have proved that $\big(E_{\text{unforg}} \setminus E_{\text{counterfeit}} \cup E_{\text{same}} \cup E_{\text{com, 2}}\big) \implies pk^{j}_{\text{tag}} \notin \mathcal{UL}$. Finally note that if $pk^{j}_{\text{tag}} \notin \mathcal{UL}$, and if $E_{\text{com, 2}} \cup E_{\text{same}} \cup E_{\text{counterfeit}}$ does not happen, then $\mathcal{B}_2$ will win the unforgeability game against S'. This yields:

$$E_{\text{unforg}} \setminus (E_{\text{com,2}} \cup E_{\text{counterfeit}}) \subset E_{\text{same}} \cup E'_{\text{sig}}. \qquad \square$$

Now suppose we are in E_{same}. By correctness of E, we deduce that the serial numbers were also identical before their encryption. Then by $\mathcal{N}$-injectivity, we have that the nonces picked during the deposits were the same, and we are therefore in $E_{\text{same-nonce}}$. Thus $E_{\text{same}} \subset E_{\text{same-nonce}}$. From this we deduce

$$E_{\text{unforg}} \subset E_{\text{com,2}} \cup E_{\text{same-nonce}} \cup E'_{\text{sig}} \cup E_{\text{com,2}} \cup E_{\text{sig}}.$$

By considering the probabilities, we finally conclude that

$$\epsilon \leq \epsilon_{\text{h,1}} + \epsilon_{\text{h,2}} + \epsilon_{\text{sig}} + \epsilon'_{\text{sig}} + \frac{d^2}{N}. \qquad \square$$

A.2 Exculpability

Theorem 20. *Suppose there is an adversary $\mathcal{A}$ against exculpability (Def. 3) of our scheme with advantage ϵ that makes at most u calls to the oracle* `URegist`*. Then there exist adversaries $\mathcal{B}_1$ against tag-exculpability with advantage ϵ_{tag}, and $\mathcal{B}_2$ against mode-hiding of C with advantage $\epsilon_{\text{m-ind}}$ such that*

$$\epsilon \leq u\,\epsilon_{tag} + \epsilon_{\text{m-ind}}.$$

We start with recalling the tag-exculpability game:

```
Experiment $\mathbf{Expt}^{\texttt{tag-exculpability}}_{\mathsf{T},\mathcal{B}}(Gr)$:
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
    $(sk_{\mathsf{T}}, pk_{\mathsf{T}}) \leftarrow \mathsf{T.KeyGen}(1^{\lambda})$
    $\mathcal{L} := \emptyset$
    $\Pi' \leftarrow \mathcal{B}^{O_1(sk_{\mathsf{T}}), O_2(sk_{\mathsf{T}}, \cdot)}(pk_{\mathsf{T}})$
    Return $\mathsf{T.VfyGuilt}(pk_{\mathsf{T}}, \Pi')$
```

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

We construct the following adversary against tag-exculpability of T.

```
Adversary $\mathcal{B}_1^{O_1(sk_{\mathsf{T}}), O_2(sk_{\mathsf{T}}, \cdot)}(par_{\mathsf{T}}, pk_{\mathsf{T}})$:
    Obtain $Gr$ from $par_{\mathsf{T}}$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
    $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$
    $par_{\mathsf{S}'} \leftarrow \mathsf{S}'\mathsf{.Setup}(Gr)$
    $par \leftarrow \big(1^{\lambda}, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck\big)$
```

```
$pk_{\mathcal{B}} \leftarrow \mathcal{A}\,(par)$
$u^* \xleftarrow{\$} \{1,\ldots,u\}$
$(i^*, \Pi^*) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy},\texttt{UWith},\texttt{Rcv},\texttt{Spd},\texttt{S\&R},\texttt{UDepo}}\,(par, pk_{\mathcal{B}})$
In the $u^*$-th call of $\texttt{URegist}$, use $pk_{\mathsf{T}}$ (instead of running $\mathsf{T.KeyGen}$)
If the adversary queries $\texttt{Spy}\,(u^*)$, abort
If the adversary queries $\texttt{UWith}, \texttt{Rcv}, \texttt{Spd}, \texttt{S\&R}, \texttt{UDepo}$ on $u^*$,
    use $O_1$ and $O_2$, and $td$ (since $sk_{\mathsf{T}}$ is unknown)
If $O_2$ fails, abort the entire procedure
Output $\Pi^*$
```

The game is perfectly simulated from $\mathcal{A}$'s point of view, except when it calls $\texttt{Spy}(u^*)$, or makes that user double-spend, or if it detects that we are in hiding-mode (which happens with probability at most $\epsilon_{\text{m-ind}}$). Let E_{ex} and E_{tag} be the events that $\mathcal{A}$ wins and that $\mathcal{B}_1$ wins, respectively. Suppose that we are in E_{ex}. This means that $\mathcal{A}$ forges a proof against one of the registered users (and does not spy on her). The probability that this user is u^* is at least $\frac{1}{u}$. In this case, we have:

- $\mathcal{A}$ did not spy on u^* or make her double-spend (as in both cases we would not be in E_{ex}).
- $\mathsf{VfyGuilt}(pk_{\mathsf{T}}, \Pi^*) = 1$ (because we are in E_{ex}); thus $\mathsf{T.VfyGuilt}(sk_{\mathsf{T}}, \Pi^*) = 1$.

We thus deduce that

$$\Pr[E_{\text{ex}}] \leq u \Pr[E_{\text{tag}}].$$

□

A.3 Coin anonymity

Theorem 21. *Suppose there is an $\mathcal{A}$ against* **coin anonymity** *($\texttt{c-an}$) of our scheme with advantage ϵ and let k be an upper-bound on the number of users transferring the challenge coins. Then there exist adversaries against mode-indistinguishability of C and tag-anonymity of T with advantages $\epsilon_{\text{m-ind}}$ and $\epsilon_{\text{t-an}}$, resp., such that*

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

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

We then argue that, by tag anonymity of T, the adversary cannot link a user to a pair (sn, tag), even when it knows the users' secret keys. We define a sequence of $k + 1$ hybrid games (as k transfers involve $k + 1$ users); going through the user vector output by the adversary, we can switch, one by one, all users from the first two the second vector. Each switch can be detected by the adversary with probability at most $2\epsilon_{\text{t-an}}$.

A technical difficulty occurs during the first swap: We would like to switch the two initial serial numbers of c_0 and c_1, but this seems problematic, as during the first withdraw (of c_0), the challenger does not yet know i_1 (and possibly this user has not even been defined yet), and thus the initial serial number of c_1. But fortunately, we note (in Proposition 25) that in hiding mode of the proof system, we do not need to compute the initial serial numbers during the withdraws! This is because we only send to the adversary (playing the bank) committed elements and proofs that reveal no information. We can therefore compute theses serial numbers *after* these withdraws, and switch them at this later moment.

Full proof. We recall $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0}$:

```
$\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0}(\lambda)$:
    $par \leftarrow \mathsf{ParamGen}(1^{\lambda})$;  $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
    $i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Run $\texttt{UWith}(i_0)$ with $\mathcal{A}$
    $i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Run $\texttt{UWith}(i_1)$ with $\mathcal{A}$
    $(\vec{i^{(0)}}, \vec{i^{(1)}}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Let $k := |\vec{i^{(0)}}|$;  if $k \neq |\vec{i^{(1)}}|$, abort the entire procedure
    Then repeat the following step for $j = 1, \ldots, k$:
        Run $\texttt{S\&R}(2j-1, (\vec{i^{(0)}})_j)$; Run $\texttt{S\&R}(2j, (\vec{i^{(1)}})_j)$
    Run $\texttt{Spd}(2k+1+b)$ with $\mathcal{A}$
    Run $\texttt{Spd}(2k+2-b)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}$ ; return $b^*$
```

In the game $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\text{hiding}}$, we will change the commitment key. If the adversary detects this, it breaks the mode-indistinguishability of C. Thus the distribution of the experiment will not change except with probability $\epsilon_{\text{m-ind}}$ (Property 22).

```
Experiment $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\text{hiding}}(\lambda)$:
    $\boxed{Gr \leftarrow \mathsf{GrGen}(1^{\lambda})}$
    $\boxed{par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)}$
    $\boxed{par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)}$
    $\boxed{par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)}$
    $\boxed{(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)}$
    $par \leftarrow \boxed{(1^{\lambda}, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)}$
    $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
    $i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Run $\texttt{UWith}(i_0)$ with $\mathcal{A}$
    $i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Run $\texttt{UWith}(i_1)$ with $\mathcal{A}$
    $(\overrightarrow{i^{(0)}}, \overrightarrow{i^{(1)}}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Let $k := |\overrightarrow{i^{(0)}}|$;  if $k \neq |\overrightarrow{i^{(1)}}|$, abort the entire procedure
    Then repeat the following for $j = 1, \ldots, k$:
        Run $\texttt{S\&R}(2j-1, (\overrightarrow{i^{(0)}})_j)$; Run $\texttt{S\&R}(2j, (\overrightarrow{i^{(1)}})_j)$
    Run $\texttt{Spd}(2k+1+b)$ with $\mathcal{A}$
    Run $\texttt{Spd}(2k+2-b)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}$ ; return $b^*$
```

Proposition 22. $\mathrm{Expt}^{\texttt{c-an}}_{\mathcal{A},0}(\lambda)$ *and* $\mathrm{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{hiding}}(\lambda)$ *are* $\epsilon_{\text{m-ind}}$*-statistically close.*

Note that td is never used in $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{hiding}}(\lambda)$. Therefore, the game can be simulated using a mode-indistinguishability challenge (Gr, ck). If ck has been generated by $\mathsf{C.Setup}$, this simulates $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,}(\lambda)$; if ck has been generated by $\mathsf{C.SmSetup}$, this simulates $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{hiding}}$. This experiment can therefore be seen as a *mode-distinguisher*. □

Let $\mathsf{C.SmPrv}_{\mathrm{sn}}, \mathsf{C.SmPrv}_{\mathrm{sn,init}}, \mathsf{C.SmPrv}_{\mathrm{tag}}, \mathsf{C.E.SmPrv}_{\mathrm{enc}}$ denote analogs of algorithms $\mathsf{C.Prv}_{\mathrm{sn}}$, $\mathsf{C.Prv}_{\mathrm{sn,init}}$, $\mathsf{C.Prv}_{\mathrm{tag}}$, $\mathsf{C.E.Prv}_{\mathrm{enc}}$, except that every $\mathsf{C.Prv}$ is substituted by $\mathsf{C.SmPrv}$ and every $\mathsf{C.Cm}$ by $\mathsf{C.ZCm}$.

Now each time the challenger is using an oracle, it uses $\mathsf{C.ZCm}$ instead of $\mathsf{C.Cm}$, $\mathsf{C.SmPrv}$ instead of $\mathsf{C.Prv}$, $\mathsf{C.SmPrv}_{\mathrm{enc}}$ instead of $\mathsf{C.Prv}_{\mathrm{enc}}$, etc.

Let $\texttt{S\&R}_{\mathrm{ZK}}, \texttt{UWith}_{\mathrm{ZK}}, \texttt{Spd}_{\mathrm{ZK}}$ denote these modified oracles.

```
Experiment $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{ZK}}(\lambda)$:
    $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
    $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$
    $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
    $par \leftarrow (1^\lambda, Gr, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
    $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
    $i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Run $\boxed{\texttt{UWith}_{\mathrm{ZK}}(i_0)}$ with $\mathcal{A}$
    $i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Run $\boxed{\texttt{UWith}_{\mathrm{ZK}}(i_1)}$ with $\mathcal{A}$
    $(\overrightarrow{i^{(0)}}, \overrightarrow{i^{(1)}}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    Let $k := |\overrightarrow{i^{(0)}}|$;  if $k \neq |\overrightarrow{i^{(1)}}|$, abort the entire procedure
    Then repeat the following step for $j = 1, \ldots, k$:
        Run $\boxed{\texttt{S\&R}_{\mathrm{ZK}}}$ $(2j-1, (\overrightarrow{i^{(0)}})_j)$; Run $\boxed{\texttt{S\&R}_{\mathrm{ZK}}}$ $(2j, (\overrightarrow{i^{(1)}})_j)$
    Run $\boxed{\texttt{Spd}_{\mathrm{ZK}}}(2k+1+b)$ with $\mathcal{A}$
    Run $\boxed{\texttt{Spd}_{\mathrm{ZK}}}(2k+2-b)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}$ ; return $b^*$
```

Because we are in the hiding mode, the following follows directly from *perfect zero-knowledge in hiding mode*:

Proposition 23. $\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{hiding}}(\lambda)$ *and* $\mathrm{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{ZK}}(\lambda)$ *are equivalently distributed.*

We now consider the following part of $\mathrm{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{ZK}}(\lambda)$) in more detail:

```
$i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
Run $\texttt{UWith}_{\mathrm{ZK}}$ $(i_0)$ with $\mathcal{A}$
$i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
Run $\texttt{UWith}_{\mathrm{ZK}}(i_1)$ with $\mathcal{A}$
```

We would like to swap the serial numbers of i_0 and i_1 by using tag-anonymity. The issue here is that in the first call to $\texttt{UWith}_{\mathrm{ZK}}$, we do not know i_1 yet (because it is only chosen in a second round). Fortunately, at this step we only sent $\mathcal{A}$ data that is unrelated to this serial number, since we are using ZCm. Thus, at the end of this part, we can compute the ciphertexts of both initial coins.

We can decompose this part of the game as follows:

```
$i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
$n^{(0)} \xleftarrow{\$} \mathcal{N}$; $\rho_{sn}^{(0)}, \rho_{cert}^{(0)}, \rho_{pk}^{(0)}, \rho_{M}^{(0)} \xleftarrow{\$} \mathcal{R}$
    $(sn^{(0)}, M_{sn}^{(0)}) \leftarrow \mathsf{T.SGen}_{\mathrm{init}}(sk_{i_0}, n^{(0)})$
    $c_{cert}^{(0)}, c_{sn}^{(0)}, c_{pk}^{(0)}, c_{M}^{(0)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{cert}^{(0)}, \rho_{sn}^{(0)}, \rho_{pk}^{(0)}, \rho_{M}^{(0)})$
    $\pi_{cert}^{(0)} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, \rho_{pk}^{(0)}, \rho_{cert}^{(0)})$
    $\pi_{sn}^{(0)} \leftarrow \mathsf{C.SmPrv}_{sn,\mathrm{init}}(td, \rho_{pk}^{(0)}, \rho_{sn}^{(0)}, \rho_{M}^{(0)})$
Send $(c_{pk}^{(0)}, c_{cert}^{(0)}, \pi_{cert}^{(0)}, c_{sn}^{(0)}, c_{M}^{(0)}, \pi_{sn}^{(0)})$ to $\mathcal{A}$
Receive $(c_{\sigma}^{(0)}, \pi_{\sigma}^{(0)})$ from $\mathcal{A}$
If $\mathsf{C.Verify}\big(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_{M}^{(0)}, c_{\sigma}^{(0)}, \pi_{\sigma}\big) = 0$, then return $\perp$
    $\nu_{sn}^{(0)} \xleftarrow{\$} \mathcal{R}$
    $\tilde{c}_{sn}^{(0)} \leftarrow \mathsf{E.Enc}(ek, sn^{(0)}, \nu_{sn}^{(0)})$
    $\tilde{\pi}_{sn}^{(0)} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{sn}^{(0)}, \tilde{c}_{sn}^{(0)})$
Pick $\rho^{(\vec{0})\prime}$ long enough to compute:
$c_1^{(0)} = \big(\mathsf{Rand}((c_{pk}^{(0)}, c_{cert}^{(0)}, \pi_{cert}^{(0)}, c_{sn}^{(0)}, \pi_{sn}^{(0)}, c_{M}^{(0)}, c_{\sigma}^{(0)}, \pi_{\sigma}^{(0)}, \tilde{c}_{sn}^{(0)}, \tilde{\pi}_{sn}^{(0)}), \rho^{(\vec{0})\prime}),$
                            $n^{(0)}, sn^{(0)}, \rho_{sn}^{(0)} + (\rho^{(\vec{0})\prime})_{sn}, \rho_{pk}^{(0)} + (\rho^{(\vec{0})\prime})_{pk}\big)$
$\mathcal{CL} \leftarrow [(i_0, c_1^{(0)}, 0, \mathcal{A})]$
$i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
$n^{(1)} \xleftarrow{\$} \mathcal{N}$; $\rho_{sn}^{(1)}, \rho_{cert}^{(1)}, \rho_{pk}^{(1)}, \rho_{M}^{(1)} \xleftarrow{\$} \mathcal{R}$;
    $(sn^{(1)}, M_{sn}^{(1)}) \leftarrow \mathsf{T.SGen}_{\mathrm{init}}(sk_{i_1}, n^{(1)})$
    $c_{cert}^{(1)}, c_{sn}^{(1)}, c_{pk}^{(1)}, c_{M}^{(1)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{cert}^{(1)}, \rho_{sn}^{(1)}, \rho_{pk}^{(1)}, \rho_{M}^{(1)})$
    $\pi_{cert}^{(1)} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, \rho_{pk}^{(1)}, \rho_{cert}^{(1)})$
    $\pi_{sn}^{(1)} \leftarrow \mathsf{C.SmPrv}_{sn,\mathrm{init}}(td, \rho_{pk}^{(1)}, \rho_{sn}^{(1)}, \rho_{M}^{(1)})$
Send $(c_{pk}^{(1)}, c_{cert}^{(1)}, \pi_{cert}^{(1)}, c_{sn}^{(1)}, c_{M}^{(1)}, \pi_{sn}^{(1)})$ to $\mathcal{A}$
Receive $(c_{\sigma}^{(1)}, \pi_{\sigma}^{(1)})$ from $\mathcal{A}$
If $\mathsf{C.Verify}\big(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_{M}^{(1)}, c_{\sigma}^{(1)}, \pi_{\sigma}^{(1)}\big) = 0$, then return $\perp$
    $\nu_{sn}^{(1)} \xleftarrow{\$} \mathcal{R}$
```

```
        $\tilde{c}_{sn}^{(1)} \leftarrow \mathsf{E.Enc}(ek, sn^{(1)}, \nu_{sn}^{(1)})$
        $\tilde{\pi}_{sn}^{(1)} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{sn}^{(1)}, \tilde{c}_{sn}^{(1)})$
    Pick $\rho^{\overrightarrow{(1)}\prime}$ long enough to compute the following:
    $c_1^{(1)} = \big(\mathsf{Rand}((c_{pk}^{(1)}, c_{cert}^{(1)}, \pi_{cert}^{(1)}, c_{sn}^{(1)}, \pi_{sn}^{(1)}, c_M^{(1)}, c_\sigma^{(1)}, \pi_\sigma^{(1)}, \tilde{c}_{sn}^{(1)}, \tilde{\pi}_{sn}^{(1)}), \rho^{\overrightarrow{(1)}\prime}),$
                $n^{(1)}, sn^{(1)}, \rho_{sn}^{(1)} + (\rho^{\overrightarrow{(1)}\prime})_{sn}, \rho_{pk}^{(1)} + (\rho^{\overrightarrow{(1)}\prime})_{pk}\big)$
    $\mathcal{CL}[2] \leftarrow (i_1, c_1^{(1)}, 0, \mathcal{A})$
```

We can do the *sn*-computations and the encryptions at the end of this part (because they are not related to data sent to $\mathcal{A}$). We can therefore replace the previous instructions by the following algorithm `DoubleUWith`:

```
DoubleUWith$^{\mathcal{A}}$:
    $i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    $\rho_{sn}^{(0)}, \rho_{cert}^{(0)}, \rho_{pk}^{(0)}, \rho_M^{(0)} \xleftarrow{\$} \mathcal{R}$; Compute:
        $c_{cert}^{(0)}, c_{sn}^{(0)}, c_{pk}^{(0)}, c_M^{(0)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{cert}^{(0)}, \rho_{sn}^{(0)}, \rho_{pk}^{(0)}, \rho_M^{(0)})$
        $\pi_{cert}^{(0)} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S'.Verify}(vk', \cdot, \cdot) = 1, \rho_{pk}^{(0)}, \rho_{cert}^{(0)})$
        $\pi_{sn}^{(0)} \leftarrow \mathsf{C.SmPrv}_{sn,\mathrm{init}}(td, \rho_{pk}^{(0)}, \rho_{sn}^{(0)}, \rho_M^{(0)})$
    Send $\big(c_{pk}^{(0)}, c_{cert}^{(0)}, \pi_{cert}^{(0)}, c_{sn}^{(0)}, c_M^{(0)}, \pi_{sn}^{(0)}\big)$ to $\mathcal{A}$
    Receive $(c_\sigma^{(0)}, \pi_\sigma^{(0)})$ from $\mathcal{A}$
    If $\mathsf{C.Verify}(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_M^{(0)}, c_\sigma^{(0)}, \pi_\sigma) = 0$ then output $\perp$
    $i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
    $\rho_{sn}^{(1)}, \rho_{cert}^{(1)}, \rho_{pk}^{(1)}, \rho_M^{(1)} \xleftarrow{\$} \mathcal{R}$; Compute:
        $c_{pk}^{(1)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{pk}^{(1)})$ ; $c_{cert}^{(1)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{cert}^{(1)})$
        $c_{sn}^{(1)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{sn}^{(1)})$ ; $c_M^{(1)} \leftarrow \mathsf{C.ZCm}(ck, \rho_M^{(1)})$
        $\pi_{cert}^{(1)} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S'.Verify}(vk', \cdot, \cdot) = 1, \rho_{pk}^{(1)}, \rho_{cert}^{(1)})$
        $\pi_{sn}^{(1)} \leftarrow \mathsf{C.SmPrv}_{sn,\mathrm{init}}(td, \rho_{pk}^{(1)}, \rho_{sn}^{(1)}, \rho_M^{(1)})$
    Send $\big(c_{pk}^{(1)}, c_{cert}^{(1)}, \pi_{cert}^{(1)}, c_{sn}^{(1)}, c_M^{(1)}, \pi_{sn}^{(1)}\big)$ to $\mathcal{A}$
    Receive $(c_\sigma^{(1)}, \pi_\sigma^{(1)})$ from $\mathcal{A}$
    If $\mathsf{C.Verify}(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_M^{(1)}, c_\sigma^{(1)}, \pi_\sigma^{(1)}) = 0$ then output $\perp$
    $n^{(0)}, n^{(1)} \xleftarrow{\$} \mathcal{N}$; $(sn^{(0)}, M_{sn}^{(0)}) \leftarrow \mathsf{T.SGen}_{\mathrm{init}}(sk_{i_0}, n^{(0)})$
    $(sn^{(1)}, M_{sn}^{(1)}) \leftarrow \mathsf{T.SGen}_{\mathrm{init}}(sk_{i_1}, n^{(1)})$
    $\nu_{sn}^{(0)}, \nu_{sn}^{(1)} \xleftarrow{\$} \mathcal{R}$
    $\tilde{c}_{sn}^{(0)} \leftarrow \mathsf{E.Enc}(ek, sn^{(0)}, \nu_{sn}^{(0)})$; $\tilde{c}_{sn}^{(1)} \leftarrow \mathsf{E.Enc}(ek, sn^{(1)}, \nu_{sn}^{(1)})$
    $\tilde{\pi}_{sn}^{(0)} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{sn}^{(0)}, \tilde{c}_{sn}^{(0)})$
    $\tilde{\pi}_{sn}^{(1)} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{sn}^{(1)}, \tilde{c}_{sn}^{(1)})$
    Pick uniformly at random $\rho^{\overrightarrow{(0)}\prime}, \rho^{\overrightarrow{(1)}\prime}$ long enough to compute:
    $c_1^{(0)} = \big(\mathsf{Rand}((c_{pk}^{(0)}, c_{cert}^{(0)}, \pi_{cert}^{(0)}, c_{sn}^{(0)}, \pi_{sn}^{(0)}, c_M^{(0)}, c_\sigma^{(0)}, \pi_\sigma^{(0)}, \tilde{c}_{sn}^{(0)}, \tilde{\pi}_{sn}^{(0)}), \rho^{\overrightarrow{(0)}\prime}),$
                $n^{(0)}, sn^{(0)}, \rho_{sn}^{(0)} + (\rho^{\overrightarrow{(0)}\prime})_{sn}, \rho_{pk}^{(0)} + (\rho^{\overrightarrow{(0)}\prime})_{pk}\big)$
    $c_1^{(1)} = \big(\mathsf{Rand}((c_{pk}^{(1)}, c_{cert}^{(1)}, \pi_{cert}^{(1)}, c_{sn}^{(1)}, \pi_{sn}^{(1)}, c_M^{(1)}, c_\sigma^{(1)}, \pi_\sigma^{(1)}, \tilde{c}_{sn}^{(1)}, \tilde{\pi}_{sn}^{(1)}), \rho^{\overrightarrow{(1)}\prime}),$
                $n^{(1)}, sn^{(1)}, \rho_{sn}^{(1)} + (\rho^{\overrightarrow{(1)}\prime})_{sn}, \rho_{pk}^{(1)} + (\rho^{\overrightarrow{(1)}\prime})_{pk}\big)$
        $\mathcal{CL}[1] \leftarrow (i_0, c_1^{(0)}, 0, \mathcal{A})$ ; $\mathcal{CL}[2] \leftarrow (i_1, c_1^{(1)}, 0, \mathcal{A})$
        Return $(i_0, i_1)$
```

To express these instruction changes, we define the following game.

```
Experiment $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{ZKV2}}(\lambda)$:
        $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
        $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
        $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$
        $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$
        $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
        $par \leftarrow (1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
        $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
        $\boxed{(i_0, i_1) \leftarrow \texttt{DoubleUWith}^{\mathcal{A}}}$
        $(\vec{i^{(0)}}, \vec{i^{(1)}}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
        Let $k := |\vec{i^{(0)}}|$;  if $k \neq |\vec{i^{(1)}}|$, abort the entire procedure
        Then repeat the following for $j = 1, \ldots, k$:
                Run $\texttt{S\&R}_{\mathrm{ZK}}$ $(2j-1, (\vec{i^{(0)}})_j, )$; Run $\texttt{S\&R}_{\mathrm{ZK}}$ $(2j, (\vec{i^{(1)}})_j)$
        Run $\texttt{Spd}_{\mathrm{ZK}}$ $(2k+1+b)$ with $\mathcal{A}$
        Run $\texttt{Spd}_{\mathrm{ZK}}$ $(2k+2-b)$ with $\mathcal{A}$
        $b^* \leftarrow \mathcal{A}$ ; return $b^*$
```

Since this change is transparent for the adversary, we get the following:

Proposition 24. $\mathrm{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{ZK}}(\lambda)$ *and* $\mathrm{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\mathrm{ZKV2}}(\lambda)$ *are equally distributed.*

Next we have to swap the serial numbers. We define two new procedures:

```
$\texttt{DoubleUWith}^{\mathcal{A}}_{\mathrm{rev}}$:
        $i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
        $\rho^{(0)}_{sn}, \rho^{(0)}_{cert}, \rho^{(0)}_{pk}, \rho^{(0)}_{M} \xleftarrow{\$} \mathcal{R}$; Compute:
                $c^{(0)}_{cert} \leftarrow \mathsf{C.ZCm}(ck, \rho^{(0)}_{cert})$
                $c^{(0)}_{sn} \leftarrow \mathsf{C.ZCm}(ck, \rho^{(0)}_{sn})$; $c^{(0)}_{pk} \leftarrow \mathsf{C.ZCm}(ck, \rho^{(0)}_{pk})$
                $c^{(0)}_{M} \leftarrow \mathsf{C.ZCm}(ck, \rho^{(0)}_{M})$
                $\pi^{(0)}_{cert} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, \rho^{(0)}_{pk}, \rho^{(0)}_{cert})$
                $\pi^{(0)}_{sn} \leftarrow \mathsf{C.SmPrv}_{sn,\mathrm{init}}(td, \rho^{(0)}_{pk}, \rho^{(0)}_{sn}, \rho^{(0)}_{M})$
        Send $(c^{(0)}_{pk}, c^{(0)}_{cert}, \pi^{(0)}_{cert}, c^{(0)}_{sn}, c^{(0)}_{M}, \pi^{(0)}_{sn})$ to $\mathcal{A}$
        Receive $(c^{(0)}_{\sigma}, \pi^{(0)}_{\sigma})$ from $\mathcal{A}$
        If $\mathsf{C.Verify}\big(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c^{(0)}_{M}, c^{(0)}_{\sigma}, \pi_{\sigma}\big) = 0$ then output $\perp$
        $i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
        $\rho^{(1)}_{sn}, \rho^{(1)}_{cert}, \rho^{(1)}_{pk}, \rho^{(1)}_{M} \xleftarrow{\$} \mathcal{R}$; Compute:
                $c^{(1)}_{cert} \leftarrow \mathsf{C.ZCm}(ck, \rho^{(1)}_{cert})$
                $c^{(1)}_{sn} \leftarrow \mathsf{C.ZCm}(ck, \rho^{(1)}_{sn})$
        $c_{pk}^{(1)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{pk}^{(1)})$
        $c_{M}^{(1)} \leftarrow \mathsf{C.ZCm}(ck, \rho_{M}^{(1)})$
        $\pi_{cert}^{(1)} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, \rho_{pk}^{(1)}, \rho_{cert}^{(1)})$
        $\pi_{sn}^{(1)} \leftarrow \mathsf{C.SmPrv}_{sn,\mathrm{init}}(td, \rho_{pk}^{(1)}, \rho_{sn}^{(1)}, \rho_{M}^{(1)})$
    Send $(c_{pk}^{(1)}, c_{cert}^{(1)}, \pi_{cert}^{(1)}, c_{sn}^{(1)}, c_{M}^{(1)}, \pi_{sn}^{(1)})$ to $\mathcal{A}$
    Receive $(c_{\sigma}^{(1)}, \pi_{\sigma}^{(1)})$ from $\mathcal{A}$
    If $\mathsf{C.Verify}\big(ck, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, c_{M}^{(1)}, c_{\sigma}^{(1)}, \pi_{\sigma}^{(1)}\big) = 0$ then output $\perp$
    $n^{(0)}, n^{(1)} \xleftarrow{\$} \mathcal{N}$; $(sn^{(0)}, M_{sn}^{(0)}) \leftarrow \mathsf{T.SGen}_{\mathrm{init}}(\boxed{sk_{i_1}}, n^{(0)})$
    $(sn^{(1)}, M_{sn}^{(1)}) \leftarrow \mathsf{T.SGen}_{\mathrm{init}}(\boxed{sk_{i_0}}, n^{(1)})$
    $\nu_{sn}^{(1)}, \nu_{sn}^{(0)} \xleftarrow{\$} \mathcal{R}$
    $\tilde{c}_{sn}^{(0)} \leftarrow \mathsf{E.Enc}(ek, sn^{(0)}, \nu_{sn}^{(0)})$; $\tilde{c}_{sn}^{(1)} \leftarrow \mathsf{E.Enc}(ek, sn^{(0)}, \nu_{sn}^{(1)})$
    $\tilde{\pi}_{sn}^{(1)} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{sn}^{(1)}, \tilde{c}_{sn}^{(1)})$
    $\tilde{\pi}_{sn}^{(0)} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{sn}^{(0)}, \tilde{c}_{sn}^{(0)})$
    Pick uniformly at random $\rho^{(\vec{0})'}, \rho^{(\vec{1})'}$ long enough to compute:
    $c_1^{(0)} = \big(\mathsf{Rand}\big((c_{pk}^{(0)}, c_{cert}^{(0)}, \pi_{cert}^{(0)}, c_{sn}^{(0)}, \pi_{sn}^{(0)}, c_{M}^{(0)}, c_{\sigma}^{(0)}, \pi_{\sigma}^{(0)}, \tilde{c}_{sn}^{(0)}, \tilde{\pi}_{sn}^{(0)}), \rho^{(\vec{0})'}\big),$
                                  $n^{(0)}, sn^{(0)}, \rho_{sn}^{(0)} + (\rho^{(\vec{0})'})_{sn}, \rho_{pk}^{(0)} + (\rho^{(\vec{0})'})_{pk}\big)$
    $c_1^{(1)} = \big(\mathsf{Rand}\big((c_{pk}^{(1)}, c_{cert}^{(1)}, \pi_{cert}^{(1)}, c_{sn}^{(1)}, \pi_{sn}^{(1)}, c_{M}^{(1)}, c_{\sigma}^{(1)}, \pi_{\sigma}^{(1)}, \tilde{c}_{sn}^{(1)}, \tilde{\pi}_{sn}^{(1)}\big), \rho^{(\vec{1})'}\big),$
                                  $n^{(1)}, sn^{(1)}, \rho_{sn}^{(1)} + (\rho^{(\vec{1})'})_{sn}, \rho_{pk}^{(1)} + (\rho^{(\vec{1})'})_{pk}\big)$
    $\mathcal{CL}[1] \leftarrow (1, i_0, c_1^{(0)}, 0, \mathcal{A})$
    $\mathcal{CL}[2] \leftarrow (i_1, c_1^{(1)}, 0, \mathcal{A})$


$\mathsf{S\&R}_{\mathrm{ZK,inv}}(j, i, sk_1, sk_2)$:
    $c \leftarrow \mathcal{CL}[j].c$
    $n' \xleftarrow{\$} \mathcal{N}$ ; $\rho'_{sn}, \rho'_{cert}, \rho'_{pk}, \rho'_{sn\text{-}pf}, \nu'_{sn} \xleftarrow{\$} \mathcal{R}$; Compute:
        $(sn', sn\text{-}pf') \leftarrow \mathsf{T.SGen}(par_{\mathsf{T}}, \boxed{sk_2}, n')$
        $c'_{cert}, c'_{pk}, c'_{sn}, c'_{sn\text{-}pf} \leftarrow \mathsf{C.ZCm}(ck, \rho'_{cert}, \rho'_{pk}, \rho'_{sn}, \rho'_{sn\text{-}pf})$
        $\tilde{c}'_{sn} \leftarrow \mathsf{E.Enc}(ek, sn', \nu'_{sn})$
        $\pi'_{cert} \leftarrow \mathsf{C.SmPrv}\big(td, \mathsf{S.Verify}(vk', \cdot, \cdot) = 1, \rho'_{\mathrm{vk}}, \rho'_{pk}, \rho'_{cert}\big)$
        $\pi'_{sn} \leftarrow \mathsf{C.SmPrv}_{sn}(td, pk'_{tag}, sn', sn\text{-}pf, \rho'_{pk}, \rho'_{sn}, \rho'_{sn\text{-}pf})$
        $\tilde{\pi}'_{sn} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho'_{sn}, \tilde{c}'_{sn})$
    Parse $c$ as $\big(c^0, (c^j = (c_{pk}^j, c_{cert}^j, \pi_{cert}^j, c_{sn}^j, \pi_{sn}^j, c_{tag}^j, \pi_{tag}^j, \tilde{c}_{sn}^j, \tilde{c}_{tag}^j, \tilde{\pi}_{sn}^j, \tilde{\pi}_{tag}^j))_{j=1}^{i},$
                                                        $n, sn, \rho_{sn}, \rho_{pk}\big)$
    $\rho_{tag}, \nu_{tag}, \rho_{t\text{-}pf} \xleftarrow{\$} \mathcal{R}$
    $(tag, t\text{-}pf) \leftarrow \mathsf{T.Gen}(par_{\mathsf{T}}, \boxed{sk_1}, n, sn')$
    $c_{tag} \leftarrow \mathsf{C.ZCm}(ck, \rho_{tag})$
    $c_{\tilde{tag}} \leftarrow \mathsf{E.Enc}(ek, tag, \nu_{tag})$
    $\pi_{tag} \leftarrow \mathsf{C.SmPrv}_{tag}(td, pk_{tag}, sn, sn', tag, t\text{-}pf, \rho_{pk}, \rho_{sn}, \rho'_{sn}, \rho_{tag}, \rho_{t\text{-}pf})$
    $\tilde{\pi}_{tag} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{tag}, \tilde{c}_{tag})$
    Check $\mathsf{VER}_{\mathrm{init}}(c^0) \wedge \bigwedge_{j=1}^{i} \mathsf{VER}_{\mathrm{std}}(c^{j-1}, c^j) \wedge$
    $\mathsf{T.Verify}(ck, c^i_{pk}, c'_{sn}, c_{tag}, \pi_{tag}) \wedge \mathsf{C.Verify}_{\mathrm{enc}}(ck, ek, c_{tag}, \tilde{c}_{tag}, \tilde{\pi}_{tag})$,
        if any of them rejects then output $\perp$
    Else choose a sufficiently long vector of randomness $\vec{\rho''}$ to compute:
    $c'' \leftarrow \mathsf{Rand}\Big((c^0, (c^j)^i_{j=1}, c'_{pk}, c'_{cert}, \pi'_{cert}, c'_{sn}, \pi'_{sn}, c_{tag}, \pi_{tag}, \tilde{c}'_{sn}, \tilde{\pi}'_{sn}, \tilde{c}'_{tag}, \tilde{\pi}'_{tag}), \vec{\rho''}\Big)$
    $c_{\mathrm{new}} := \big(c'', n', sn', \rho'_{sn} + (\vec{\rho''})_{sn'}, \rho'_{pk} + (\vec{\rho''})_{pk'}\big)$
    $\mathcal{CL}[|\mathcal{CL}| + 1] := (i, c_{\mathrm{new}}, 0, j)$
```

We define a new game for all $l \in \{0, \ldots, k-1\}$:

```
Experiment $\mathbf{Expt}^{\mathsf{c\text{-}an}}_{\mathcal{A},0,\mathrm{ZKV2},l}(\lambda)$:
    $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
    $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$
    $par_{\mathsf{S}'} \leftarrow \mathsf{S}'\mathsf{.Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
    $par \leftarrow (1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
    $pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
    $\boxed{(i_0, i_1) \leftarrow \mathtt{DoubleUWith}^{\mathcal{A}}_{\mathrm{rev}}}$
    $(\vec{i^{(0)}}, \vec{i^{(1)}}) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy}}$
    Let $k := |\vec{i^{(0)}}|$;  if $k \neq |\vec{i^{(1)}}|$, abort the entire procedure
    $\boxed{\text{Consider } i_0 \text{ as } (\vec{i^{(0)}})_0, \text{ and } i_1 \text{ as } (\vec{i^{(1)}})_0}$
    $\boxed{\text{For all } b, j : sk^{(b)}_j \leftarrow \mathcal{UL}[(\vec{i^{(b)}})_j].sk}$
    Repeat the following for $j = 1, \ldots, l$:
        $\boxed{\text{Run } \mathtt{S\&R}_{\mathrm{ZK,inv}}\ (2j-1, (\vec{i^{(0)}})_j, sk^{(1)}_{j-1}, sk^{(1)}_j)}$
        $\boxed{\text{Run } \mathtt{S\&R}_{\mathrm{ZK,inv}}\ (2j, (\vec{i^{(1)}})_j, sk^{(0)}_{j-1}, sk^{(0)}_j)}$
    $\boxed{\text{Run } \mathtt{S\&R}_{\mathrm{ZK,inv}}\ (2l+1, (\vec{i^{(0)}})_{l+1}, sk^{(1)}_l, sk^{(0)}_{l+1})}$
    $\boxed{\text{Run } \mathtt{S\&R}_{\mathrm{ZK,inv}}\ (2l+2, (\vec{i^{(1)}})_{l+1}, sk^{(0)}_l, sk^{(1)}_{l+1})}$
    Repeat the following for $j = l+2, \ldots, k$:
        Run $\mathtt{S\&R}_{\mathrm{ZK}}$ $(2j-1, (\vec{i^{(0)}})_j)$; Run $\mathtt{S\&R}_{\mathrm{ZK}}$ $(2j, (\vec{i^{(1)}})_j)$
    Run $\mathtt{Spd}_{\mathrm{ZK}}$ $(2k+1+b)$ with $\mathcal{A}$
    Run $\mathtt{Spd}_{\mathrm{ZK}}$ $(2k+2-b)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}$ ; return $b^*$
```

Proposition 25. $\mathrm{Expt}^{\mathsf{c\text{-}an}}_{\mathcal{A},0,\mathrm{ZKV2}}(\lambda)$ *and* $\mathrm{Expt}^{\mathsf{c\text{-}an}}_{\mathcal{A},0,\mathrm{ZKV2},0}(\lambda)$ *are* $2\epsilon_{\mathrm{t\text{-}an}}$*-statistically close.*

We receive a challenge par_{T} in the `tag-anon` game for T (and not the tag exculpability game, in contrast to the proof of Theorem 20), and we use par_{T} as parameter for the tags instead of generating it in $\mathrm{Expt}^{\mathsf{c\text{-}an}}_{\mathcal{A},0,\mathrm{ZKV2}}$.

In `DoubleUWith`, we send to the `tag-anon`-challenger the secret keys of i_0 and i_1, and we use O_1 to generate the serial number of i_0 in `DoubleUWith` and $O_2(0)$ to generate the corresponding tag in the first $\mathtt{S\&R}_{\mathrm{ZK}}$[4].

If the challenger was in mode 0, this will not change the experiment. But if the challenger was in mode 1, it will replace i_0 by i_1. Let $\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2},-1}$ denote the game corresponding to this swap and $\varDelta$ be the statistical distance.

We have just proved that

$$\varDelta\big(\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2}}(\lambda), \mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2},-1}(\lambda)\big) \leq \epsilon_{\text{t-an}}.$$

Analogously, we replace i_1 by i_0, i.e., we show that

$$\varDelta\big(\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2},-1}(\lambda), \mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2},0}(\lambda)\big) \leq \epsilon_{\text{t-an}},$$

and therefore $\varDelta\big(\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2}}(\lambda), \mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2},0}(\lambda)\big) \leq 2\epsilon_{\text{t-an}}$. □

The proof is completely analogous for the following property, which lets us swap multiple games.

Proposition 26. *For all* $l \in \{0,\dots,k-2\}$*, we have that* $\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2},l}(\lambda)$ *and* $\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2},l+1}(\lambda)$ *are* $2\epsilon_{\text{t-an}}$*-statistically close.*

Finally, we define a last oracle to swap the last keys (and the corresponding game):

```
$\mathsf{Spd}_{\mathrm{ZK,inv}}(k, sk_1)$:
    Receive $(sn', \rho'_{sn})$ from $\mathcal{A}$
    $c \leftarrow \mathcal{CL}[k].c$
    Parse $c$ as $\big(c^0, (c^j = (c^j_{pk}, c^j_{cert}, \pi^j_{cert}, c^j_{sn}, \pi^j_{sn}, c^j_{tag}, \pi^j_{tag}, \tilde{c}^j_{sn}, \tilde{c}^j_{tag}, \tilde{\pi}^j_{sn}, \tilde{\pi}^j_{tag}))^i_{j=1},$
        $n, sn, \rho_{sn}, \rho_{pk}\big)$
    $\rho_{tag}, \nu_{tag}, \rho_{t\text{-}pf} \xleftarrow{\$} \mathcal{R}$
    $(tag, t\text{-}pf) \leftarrow \mathsf{T.Gen}(par_{\mathsf{T}}, \boxed{sk_1}, n, sn')$
    $c_{tag} \leftarrow \mathsf{C.ZCm}(ck, \rho_{tag})$
    $\tilde{c}_{tag} \leftarrow \mathsf{E.Enc}(ek, tag, \nu_{tag})$
    $\pi_{tag} \leftarrow \mathsf{C.SmPrv}_{tag}(td, pk_{tag}, sn, sn', tag, t\text{-}pf, \rho_{pk}, \rho_{sn}, \rho'_{sn}, \rho_{tag}, \rho_{t\text{-}pf})$
    $\tilde{\pi}_{tag} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{tag}, \tilde{c}_{tag})$
    Send $\big(c^0, (c^j)^i_{j=1}, c_{tag}, \pi_{tag}, \tilde{c}_{tag}, \tilde{\pi}_{tag}\big)$ to $\mathcal{A}$

Experiment $\mathbf{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2,k}}(\lambda)$:
    $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$
    $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$
    $par_{\mathsf{S'}} \leftarrow \mathsf{S'.Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
```

[4] We use the oracle only in these step; for the other serial number and tag generations, we use the secret keys (which we have generated) like in $\mathrm{Expt}^{\text{c-an}}_{\mathcal{A},0,\mathrm{ZKV2}}$.

```
$par \leftarrow (1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
$pk_{\mathcal{B}} \leftarrow \mathcal{A}(par)$
$(i_0, i_1) \leftarrow \texttt{DoubleUWith}^{\mathcal{A}}_{\text{rev}}$
$(\vec{i^{(0)}}, \vec{i^{(1)}}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
Let $k := |\vec{i^{(0)}}|$;  if $k \neq |\vec{i^{(1)}}|$, abort the entire procedure
Consider $i_0$ as $(\vec{i^{(0)}})_0$, and $i_1$ as $(\vec{i^{(1)}})_0$
For all  $b, j : sk^{(b)}_j \leftarrow \mathcal{UL}[(\vec{i^{(b)}})_j].sk$
Then repeat the following for $j = 1, \ldots, \boxed{\text{k}}$:
    Run $\texttt{S\&R}_{\text{ZK,inv}}\ (2j-1, (\vec{i^{(0)}})_j, \mathcal{UL}[(\vec{i^{(1)}})_{j-1}].sk, \mathcal{UL}[(\vec{i^{(1)}})_j].sk)$
    Run $\texttt{S\&R}_{\text{ZK,inv}}\ (2j, (\vec{i^{(1)}})_j, \mathcal{UL}[(\vec{i^{(0)}})_{j-1}].sk, \mathcal{UL}[(\vec{i^{(0)}})_j].sk)$
$\boxed{\text{Run } \texttt{Spd}_{\text{ZK,inv}}\ (2k+1+b, \mathcal{UL}[(\vec{i^{(1)}})_k].sk) \text{ with } \mathcal{A}}$
$\boxed{\text{Run } \texttt{Spd}_{\text{ZK,inv}}\ (2k+2-b, \mathcal{UL}[(\vec{i^{(0)}})_k].sk) \text{ with } \mathcal{A}}$
$b^* \leftarrow \mathcal{A}$ ; return $b^*$
```

Analogously to previous two propositions, we get:

Proposition 27. $\text{Expt}^{\text{c-an}}_{\mathcal{A},0,\text{ZKV2},k-1}(\lambda)$ *and* $\text{Expt}^{\text{c-an}}_{\mathcal{A},0,\text{ZKV2},k}(\lambda)$ *are* $2\epsilon_{\text{t-an}}$*-statistically close.*

By noting that two randomized commitments of the same type in the hiding-mode have the (exact) same distribution, we get that $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0,\text{ZKV2},k}(\lambda)$ is equally distributed as $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},1,\text{ZK}}(\lambda)$.

From a similar reasoning, we get that $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},1,\text{ZK}}(\lambda)$ is $\epsilon_{\text{m-ind}}$ statistically close to $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},1}(\lambda)$. Finally, we deduce that $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},1}(\lambda)$ is $2(\epsilon_{\text{ZK}}+(k+1)\epsilon_{\text{t-an}})$-statistically-close to $\mathbf{Expt}^{\texttt{c-an}}_{\mathcal{A},0}(\lambda)$. □

Note that $\epsilon_{\text{t-an}}$ is the advantage against tag-anonymity of an adversary that is making just one call to O_1 and one to O_2.

A.4 Coin transparency

Theorem 28. *Let $\mathcal{A}$ be an adversary against* **coin-transparency** *(*`c-tr`*) of our scheme with advantage ϵ, and let ℓ be the size of the challenge coins, and k be an upper-bound on the number of users transferring the challenge coins. Then there exist adversaries against mode-indistinguishability of C, tag-anonymity of T, IACR-security of E and RCCA-security of E' with advantages $\epsilon_{\text{m-ind}}$, $\epsilon_{\text{t-an}}$, ϵ_{iacr} and ϵ_{rcca}, resp., such that*

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

The proof proceeds via an hybrid argument. We first recall game $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0}$.

```
Experiment $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0}(\lambda)$:
    $par \leftarrow \mathsf{ParamGen}(1^\lambda)$;  $(sk_{\mathcal{B}}, pk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}(par)$
```

```
$\mathcal{DCL}' := \emptyset$; $ctr \leftarrow 0$
$i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo}',\texttt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
Run $\texttt{Rcv}(i_0)$ with $\mathcal{A}$
Let $c_0$ be the coin received
$x_0 \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_0)$
If $x_0 = \perp$ then $ctr \leftarrow ctr + 1$
$\mathcal{DCL}' \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \emptyset, c_0)$
$i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo}',\texttt{Spy}}$
Run $\texttt{Rcv}(i_1)$ with $\mathcal{A}$
Let $c_1$ be the coin received
$x_1 \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_1)$
If $x_1 = \perp$ then $ctr \leftarrow ctr + 1$
If $\texttt{comp}(c_0, c_1) \neq 1$ then return 0
$x_2 \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}', c_1)$
If $x_2 \neq \perp$ then $\mathcal{DCL}' \leftarrow x_2$
$(\vec{i}^{(0)}, \vec{i}^{(1)}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo}',\texttt{Spy}}$
Let $k := |\vec{i}^{(0)}|$;  If $k \neq |\vec{i}^{(1)}|$ then return 0
If $k \neq 0$ then run $\texttt{S\&R}(1, (\vec{i}^{(0)})_1)$
For $j = 2, \ldots, k$:
        Run $\texttt{S\&R}(j + 1, (\vec{i}^{(0)})_j)$
Run $\texttt{Spd}(k + 2)$ with $\mathcal{A}$
$b^* \leftarrow \mathcal{A}^{\texttt{BDepo}'}$ ; return $b^*$     // instead of $\mathsf{CheckDS}^*$, $\texttt{BDepo}'$
                                  uses $\mathsf{CheckDS}'(\cdot,\cdot,\cdot,\cdot,\mathcal{DCL}')$ defined as follows:
```

```
$\mathsf{CheckDS}'(sk_{\mathcal{CK}}, \mathcal{UL}, \mathcal{DCL}, c, \mathcal{DCL}')$:
    $x \leftarrow \mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}', c)$
    If $x = \perp$:
        $ctr \leftarrow ctr + 1$
        If $ctr > 1$ then return 0
    Return $\mathsf{CheckDS}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}, c)$
```

Note that we are only interested in detecting double spending, and not tracing the cheater (because $\mathsf{CheckDS}$ always run on an empty user list, which in our instantiation implies that it will never accuse someone and will output $\perp$ when it detects a double-spending). We can therefore simplify $\mathsf{CheckDS}$ as follows (and the distribution of the output of the experiment will be unchanged):

```
$\mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \mathcal{UL}, \mathcal{DCL}, c)$:
    $sn \leftarrow \mathsf{E}'.\mathsf{Dec}(dk_{\text{init}}, \tilde{c}^0_{sn})$
    If $sn \in \mathcal{DCL}$ then return $\perp$
    Else $\mathcal{DCL} := \mathcal{DCL} \cup \{sn\}$; return $\mathcal{DCL}$
```

Let $\mathsf{CheckDS}'_{\text{simple}}$ and $\texttt{BDepo}'_{\text{simple}}$ be similar variants of $\mathsf{CheckDS}'$ and $\texttt{BDepo}'$, respectively, that use $\mathsf{CheckDS}_{\text{simple}}$ instead of $\mathsf{CheckDS}$. The beginning of the proof will be very similar to the one of coin-anonymity.

Note that we choose to only keep the *initial* serial numbers in $\mathcal{DCL}$, since in this game we only check if there is double-spending or not (in particular, we do not send any proof of culpability). Thus only the first serial-number component of a coin matters, and it will not change in the following game.

Using the same arguments as in Sect. A.3, we get the following.

Proposition 29. $\mathrm{Expt}^{\text{c-tr}}_{\mathcal{A},0}(\lambda)$ *and* $\mathrm{Expt}^{\text{c-tr}}_{\mathcal{A},0,ZK}(\lambda)$ *are* $(\epsilon_{\text{m-ind}}+t\epsilon_{\text{t-an}})$ *statistically close.*

Experiment $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\mathrm{ZK}}(\lambda)$:

```
    $\boxed{Gr \leftarrow \mathsf{GrGen}(1^\lambda)}$
    $\boxed{par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)}$
    $\boxed{par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)}$
    $\boxed{par_{\mathsf{S}'} \leftarrow \mathsf{S}'\mathsf{.Setup}(Gr)}$
    $\boxed{(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)}$
    $par \leftarrow \boxed{(1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)}$
    $(pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}$
    $\mathcal{DCL}' := \emptyset;\ ctr \leftarrow 0$
    $i_0 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo}'_{\text{simple}},\texttt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
    Run $\boxed{\texttt{Rcv}_{\mathrm{ZK}}}(i_0)$ with $\mathcal{A}$ ; let $c_0$ be the received coin
    $x_0 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_0)$
    If $x_0 = \perp$ then $ctr \leftarrow ctr + 1$
    $\mathcal{DCL}' \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \emptyset, c_0)$
    $i_1 \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo}'_{\text{simple}},\texttt{Spy}}$
    Run $\boxed{\texttt{Rcv}_{\mathrm{ZK}}}(i_1)$ with $\mathcal{A}$ ; let $c_1$ be the received coin
    $x_1 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_1)$
    If $x_1 = \perp$ then $ctr \leftarrow ctr + 1$
    If $\texttt{comp}(c_0, c_1) \neq 1$ then return 0
    $x_2 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{DCL}', c_1)$
    If $x_2 \neq \perp$ then $\mathcal{DCL}' \leftarrow x_2$
    $(\vec{i}^{(0)}, \vec{i}^{(1)}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo}'_{\text{simple}},\texttt{Spy}}$
    Let $k$ be the size of $\vec{i}^{(0)}$
    If $k$ is different of the size of $\vec{i}^{(1)}$: return 0
    If $k \neq 0$, then run $\boxed{\texttt{S\&R}_{\mathrm{ZK}}}(1, (\vec{i}^{\boxed{(1)}})_1)$
    Then repeat the following for $j = 3, \ldots, (k+1)$:
        Run $\boxed{\texttt{S\&R}_{\mathrm{ZK}}}(j, (\vec{i}^{\boxed{(1)}})_{j-1})$
    Run $\boxed{\texttt{Spd}_{\mathrm{ZK}}}(k+1)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}^{\texttt{BDepo}'_{\text{simple}}}$ ; return $b^*$        // $\texttt{BDepo}'_{\text{simple}}$ *uses* $\mathsf{CheckDS}'_{\text{simple}}$
```

Now we can leverage zero-knowledge and randomizability to partially "remelt (all the commits and proofs in) c_0. The strategy is the following: Using `Extract`, defined below, we "break" c_0 and c_1 to extract all the relevant information (serial numbers, tags and nonce). Using `Remelt` we remelt both coins, after switching the following content:

- In $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{iacr}}$, we switch all the tags, and serial numbers (except the first serial number).
- In $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{rcca1}}$, we switch the first serial numbers of both coins.
- In $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{final}}$, we switch the nonces.

Thus we define the two followings procedures:

```
$\texttt{Extract}(sk_{\mathcal{CK}}, c)$:
    Parse $c$ as:
    $\big(c^0, \big(c^k = (c^k_{pk}, c^k_{cert}, \pi^k_{cert}, c^k_{sn}, \pi^k_{sn}, c^k_{tag}, \pi^k_{tag}, \tilde{c}^k_{sn}, \tilde{c}^k_{tag}, \tilde{\pi}^k_{sn}, \tilde{\pi}^k_{tag})\big)^{\ell}_{k=1},$
                                                    $n, sn, \rho_{sn}, \rho_{pk}\big)$
    $sn_0 = \mathsf{E}'.\mathsf{Dec}(dk_{\text{init}}, \tilde{c}^0_{sn})$
    Return $\big(sn_0, n, (\tilde{c}^1_{sn}, \ldots, \tilde{c}^{\ell}_{sn}), (\tilde{c}^1_{tag}, \ldots, \tilde{c}^{\ell}_{tag})\big)$
```

```
$\texttt{Remelt}(td, \tilde{c}_{sn}, n, (\tilde{c}^1_{sn}, \ldots, \tilde{c}^{\ell}_{sn}), (\tilde{c}^1_{tag}, \ldots, \tilde{c}^{\ell}_{tag}))$:
    $\rho^0_{sn}, \rho^0_{cert}, \rho^0_{pk}, \rho^0_M, \rho^0_\sigma \xleftarrow{\$} \mathcal{R}$
    $c_{sn}, c_{cert}, c_{pk}, c_M, c_\sigma \leftarrow \mathsf{C.ZCm}(ck, \rho^0_{sn}, \rho^0_{cert}, \rho^0_{pk}, \rho^0_M, \rho^0_\sigma)$
    $\pi_{cert} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, \rho^0_{pk}, \rho^0_{cert})$
    $\pi_\sigma \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S.Verify}(vk, \cdot, \cdot) = 1, \rho^0_M, \rho^0_\sigma)$
    $\pi_{sn} \leftarrow \mathsf{C.SmPrv}_{sn,\text{init}}(td, \rho^0_{pk}, \rho^0_{sn}, \rho^0_M)$
    $\tilde{\pi}_{sn} \leftarrow \mathsf{C.SmPrv}'_{\text{enc}}(td, ek', \rho^0_{sn}, \tilde{c}^0_{sn})$
    $c^0 \leftarrow (c_{pk}, c_{cert}, \pi_{cert}, c_{sn}, \pi_{sn}, c_M, c_\sigma, \pi_\sigma, \tilde{c}_{sn}, \tilde{\pi}_{sn})$
    For $k \in \{1, \ldots, \ell\}$:
        $\rho^k_{sn}, \rho^k_{cert}, \rho^k_{pk}, \rho^k_{tag}, \xleftarrow{\$} \mathcal{R}$
        $c^k_{sn}, c^k_{tag}, c^k_{pk}, c^k_M, c^k_\sigma \leftarrow \mathsf{C.ZCm}(ck, \rho^k_{sn}, \rho^k_{tag}, \rho^k_{pk}, \rho^k_M, \rho^k_\sigma)$
        $\pi^k_{cert} \leftarrow \mathsf{C.SmPrv}(td, \mathsf{S}'.\mathsf{Verify}(vk', \cdot, \cdot) = 1, \rho^k_{pk}, \rho^k_{cert})$
        $\pi^k_{sn} \leftarrow \mathsf{C.SmPrv}_{sn}(td, \rho^k_{pk}, \rho^k_{sn})$
        $\tilde{\pi}^k_{sn} \leftarrow \mathsf{C.SmPrv}_{\text{enc}}(td, ek, \rho^k_{sn}, \tilde{c}^k_{sn})$
        $\pi^k_{tag} \leftarrow \mathsf{C.SmPrv}_{sn}(td, \rho^k_{pk}, \rho^{k-1}_{sn}, \rho^k_{sn}, \rho^k_{tag})$
        $\tilde{\pi}^k_{tag} \leftarrow \mathsf{C.SmPrv}_{\text{enc}}(td, ek, \rho^k_{tag}, \tilde{c}^k_{tag})$
        $c^k \leftarrow (c^k_{pk}, c^k_{cert}, \pi^k_{cert}, c^k_{sn}, \pi^k_{sn}, \tilde{c}^k_{sn}, \tilde{\pi}^k_{sn}, \tilde{c}^k_{sn}, \pi^k_{tag}, \tilde{\pi}^k_{tag})$
    Return $\big((c^k)^{\ell}_{k=0}, n, sn, \rho_{sn}, \rho_{pk}\big)$
```

By the zero-knowledge property of C, the outputs of $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{ZK}}(\lambda)$ and $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{remelt}}(\lambda)$ will follow perfectly the same distribution:

Proposition 30. $\text{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{remelt}}(\lambda)$ *and* $\text{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{ZK}}(\lambda)$ *are equally distributed.*

```
Experiment $\mathbf{Expt}^{\texttt{c-tr}}_{\mathcal{A},0,\text{remelt}}(\lambda)$:
    $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$ ; $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$ ; $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
    $par \leftarrow (1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
    $(pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}()$
    $\mathcal{DCL}' := \emptyset$; $ctr \leftarrow 0$
    $i_0 \leftarrow \mathcal{A}^{\mathtt{URegist}, \mathtt{BDepo}'_{\text{simple}}, \mathtt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
    Run $\mathtt{Rcv}_{\text{ZK}}(i_0)$ with $\mathcal{A}$ ; let $c_0$ be the received coin
    $x_0 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_0)$
    If $x_0 = \perp$ then $ctr \leftarrow ctr + 1$
    $i_1 \leftarrow \mathcal{A}^{\mathtt{URegist}, \mathtt{BDepo}', \mathtt{Spy}}$
    Run $\mathtt{Rcv}_{\text{ZK}}(i_1)$ with $\mathcal{A}$ ; let $c_1$ be the received coin
    $x_1 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_1)$
    If $x_1 = \perp$ then $ctr \leftarrow ctr + 1$
    If $\mathtt{comp}(c_0, c_1) \neq 1$ then abort
    $\boxed{sn_0, n^{(0)}, \vec{\tilde{c}}_{sn,0}, \vec{\tilde{c}}_{tag,0} \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_0)}$
    $\boxed{sn_1, n^{(1)}, \vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1} \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_1)}$
    $\boxed{\nu \xleftarrow{\$} \mathcal{R}}$
    $\boxed{\tilde{c}_{sn} \leftarrow \mathsf{E}'.\mathsf{Enc}(ek', sn_0, \nu)}$
    $\boxed{c'_0 \leftarrow \mathtt{Remelt}(td, \tilde{c}_{sn}, n^{(0)}, \vec{\tilde{c}}_{sn,0}, \vec{\tilde{c}}_{tag,0})}$
    $\boxed{\mathcal{DCL}' := \{sn_0, sn_1\}}$
    $(\vec{i}^{(0)}, \vec{i}^{(1)}) \leftarrow \mathcal{A}^{\mathtt{URegist}, \mathtt{BDepo}'_{\text{simple}}, \mathtt{Spy}}$
    Let $k$ be the size of $\vec{i}^{(0)}$
    If $k$ is different of the size of $\vec{i}^{(1)}$: return 0
    $\boxed{\mathcal{CL}[1].c \leftarrow c'_0}$
    Run $\mathtt{S\&R}_{\text{ZK}}(1, (\vec{i}^{(1)})_1)$
    Then repeat the following for $j = 3, \ldots, (k+1)$:
        Run $\mathtt{S\&R}_{\text{ZK}}(j, (\vec{i}^{(1)})_{j-1})$
    Run $\mathtt{Spd}_{\text{ZK}}(k+1)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}^{\mathtt{BDepo}'_{\text{simple}}}$ ; return $b^*$
```

The serial numbers and tags are encrypted. We can change the ciphertexts encrypted with E in `Extract`; each switch will affect the distribution of the output of the overall experiment with probability at most ϵ_{iacr}. We deduce the following:

Proposition 31. *$\text{Expt}^{\text{c-tr}}_{\mathcal{A},0,\text{remelt}}(\lambda)$ and $\text{Expt}^{\text{c-tr}}_{\mathcal{A},0,\text{iacr}}(\lambda)$ are $2\,\ell$-statistically close.*

```
Experiment $\mathbf{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},0,\text{iacr}}(\lambda)$:
    $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$ ; $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$ ; $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
    $par \leftarrow (1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
    $(pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}$
    $\mathcal{DCL}' := \emptyset$; $ctr \leftarrow 0$
    $i_0 \leftarrow \mathcal{A}^{\mathtt{URegist}, \mathtt{BDepo}', \mathtt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
    Run $\mathtt{Rcv}(i_0)$ with $\mathcal{A}$ ; let $c_0$ be the received coin
    $x_0 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_0)$
    If $x_0 = \bot$ then $ctr \leftarrow ctr + 1$
    $i_1 \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}$
    Run $\mathtt{Rcv}(i_1)$ with $\mathcal{A}$ ; let $c_1$ be the received coin
    $x_1 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_1)$
    If $x_1 = \bot$ then $ctr \leftarrow ctr + 1$
    If $\mathtt{comp}(c_0, c_1) \neq 1$ abort the entire procedure
    $sn_0, n^{(0)}, \vec{\tilde{c}}_{sn,0}, \vec{\tilde{c}}_{tag,0} \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_0)$
    $sn_1, n^{(1)}, \vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1} \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_1)$
    $\nu \xleftarrow{\$} \mathcal{R}$
    $\tilde{c}_{sn} \leftarrow \mathsf{E}'.\mathsf{Enc}(ek', sn_0, \nu)$
    $c_0' \leftarrow \mathtt{Remelt}(td, \tilde{c}_{sn}, n^{(0)}, \boxed{\vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1}})$
    $\mathcal{DCL}' := \{sn_0, sn_1\}$
    $(\vec{i}^{(0)}, \vec{i}^{(1)}) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}'_{\text{simple}},\mathtt{Spy}}$
    Let $k$ be the size of $\vec{i}^{(0)}$
    If $k$ is different of the size of $\vec{i}^{(1)}$ abort
    $\mathcal{CL}[1].c \leftarrow c_0'$
    Run $\mathtt{S\&R}_{\text{ZK}}(1, (\vec{i}^{(1)})_1)$
    Then repeat the following step for $j = 3, \ldots, (k+1)$:
        Run $\mathtt{S\&R}_{\text{ZK}}(j, (\vec{i}^{(1)})_{j-1})$
    Run $\mathtt{Spd}_{\text{ZK}}(k+1)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}^{\mathtt{BDepo}'_{\text{simple}}}$ ; return $b^*$
```

For the next step, we will rely on RCCA-security of E'.

```
Experiment $\mathbf{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},0,\text{rcca}}(Gr, ek')$:
    $par_{\mathsf{T}} \leftarrow \mathsf{T}.\mathsf{Setup}(Gr)$ ; $par_{\mathsf{S}} \leftarrow \mathsf{S}.\mathsf{Setup}(Gr)$ ; $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C}.\mathsf{SmSetup}(Gr)$
    $par \leftarrow (1^{\lambda}, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
    $\boxed{(vk, sk) \leftarrow \mathsf{S}.\mathsf{KeyGen}}$
    $\boxed{(vk', sk') \leftarrow \mathsf{S}.\mathsf{KeyGen}}$
    $\boxed{(ek, dk) \leftarrow \mathsf{E}.\mathsf{KeyGen}}$
    $\boxed{pk_{\mathcal{B}} := (ek', ek, vk, vk')}$
    $\boxed{sk_{\mathcal{W}} := (sk, sk')}$
    $\mathcal{DCL}' := \emptyset$; $ctr \leftarrow 0$
    $\boxed{\text{(Use the } \mathsf{Dec} \text{ oracle call in all the } \mathtt{BDepo}'_{\text{simple}} \text{ call from } \mathcal{A};)}$
    $i_0 \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
    Run $\mathtt{Rcv}(i_0)$ with $\mathcal{A}$ ; et $c_0$ be the received coin
    $x_0 \leftarrow \mathsf{CheckDS}_{\text{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_0)$
    If $x_0 = \bot$ then $ctr \leftarrow ctr + 1$
    $i_1 \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}$
```

```
Run $\mathtt{Rcv}(i_1)$ with $\mathcal{A}$ ; let $c_1$ be the received coin
$x_1 \leftarrow \mathsf{CheckDS}_{\mathrm{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_1)$
If $x_1 = \perp$ then $ctr \leftarrow ctr + 1$
If $\mathtt{comp}(c_0, c_1) \neq 1$ abort the entire procedure
$sn_0, n^{(0)}, \vec{sn}_0, \vec{tag}_0 \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_0)$
$sn_1, n^{(1)}, \vec{sn}_1, \vec{tag}_1 \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_1)$
$\mathcal{DCL}' := \{sn_0, sn_1\}$
$(\vec{i}^{(0)}, \vec{i}^{(1)}) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}'_{\mathrm{simple}},\mathtt{Spy}}$
Let $k$ be the size of $\vec{i}^{(0)}$
If $k$ is different of the size of $\vec{i}^{(1)}$ abort
$\boxed{\text{Send } sn_0, sn_1 \text{ as challenge for the rcca-security game and receive } \tilde{c}}$
$\boxed{c'_0 \leftarrow \mathtt{Remelt}(td, \tilde{c}, n^{(0)}, \vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1})}$
$\boxed{\text{(insert the challenge } \tilde{c} \text{ in this step as } \tilde{c}^0_{sn})}$
$\mathcal{CL}[1].c \leftarrow c'_0$
Run $\mathtt{S\&R}_{\mathrm{ZK}}(1, (\vec{i}^{(1)})_1)$
Then repeat the following step for $j = 3, \ldots, (k+1)$:
        Run $\mathtt{S\&R}_{\mathrm{ZK}}(j, (\vec{i}^{(1)})_{j-1})$
Run $\mathtt{Spd}_{\mathrm{ZK}}(k+1)$ with $\mathcal{A}$
$b^* \leftarrow \mathcal{A}^{\mathtt{BDepo}'_{\mathrm{rcca}}}$ ; return $b^*$

  $\mathsf{CheckDS}'_{\mathrm{rcca}}(sk_{\mathcal{CK}}, \mathcal{UL}, \mathcal{DCL}, c, )$:
        $t \leftarrow \mathsf{E.GDec}(dk_{\mathrm{init}}, \tilde{c}^0_{sn})$
        If $t =$"replay" and $ctr > 0$ then abort the entire procedure
        Else if $t =$"replay" then $ctr \leftarrow ctr + 1$
        Else if $t \in \mathcal{DCL}$ then return $\perp$
        Else add $sn$ to $\mathcal{DCL}$, and return $\mathcal{DCL}$
```

If the challenger of the RCCA security game encrypts sn_0, the resulting experiment will be $\mathbf{Expt}^{\mathtt{c-tr}}_{\mathcal{A},0,\mathrm{iacr}}(\lambda)$; otherwise it will be $\mathbf{Expt}^{\mathtt{c-tr}}_{\mathcal{A},0,\mathrm{rcca1}}(\lambda)$. We thus deduce:

Proposition 32. $\mathrm{Expt}^{\mathtt{c-tr}}_{\mathcal{A},0,\mathrm{iacr}}(\lambda)$ *and* $\mathrm{Expt}^{\mathtt{c-tr}}_{\mathcal{A},0,\mathrm{rcca1}}(\lambda)$ *are* ϵ_{rcca}*-statistically close.*

```
Experiment $\mathbf{Expt}^{\mathtt{c-tr}}_{\mathcal{A},0,\mathrm{rcca1}}(\lambda)$:
    $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$ ; $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$ ; $par_{\mathsf{S'}} \leftarrow \mathsf{S'.Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
    $par \leftarrow (1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S'}}, par_{\mathsf{T}}, ck)$
    $(pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}$
    $\mathcal{DCL}' := \emptyset$; $ctr \leftarrow 0$
    $i_0 \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
    Run $\mathtt{Rcv}(i_0)$ with $\mathcal{A}$ ; let $c_0$ be the received coin
    $x_0 \leftarrow \mathsf{CheckDS}_{\mathrm{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_0)$
    If $x_0 = \perp$ then $ctr \leftarrow ctr + 1$
    $i_1 \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}$
    Run $\texttt{Rcv}(i_1)$ with $\mathcal{A}$ ; let $c_1$ be the received coin
    $x_1 \leftarrow \mathsf{CheckDS}_{\mathrm{simple}}(sk_{\mathcal{CK}}, \emptyset, \mathcal{CL}, c_1)$
    If $x_1 = \perp$ then $ctr \leftarrow ctr + 1$
    If $\texttt{comp}(c_0, c_1) \neq 1$ abort the entire procedure
    $sn_0, n^{(0)}, \vec{\tilde{c}}_{sn,0}, \vec{\tilde{c}}_{tag,0} \leftarrow \texttt{Extract}(sk_{\mathcal{CK}}, c_0)$
    $sn_1, n^{(1)}, \vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1} \leftarrow \texttt{Extract}(sk_{\mathcal{CK}}, c_1)$
    $\nu \xleftarrow{\$} \mathcal{R}$
    $\tilde{c}_{sn} \leftarrow \mathsf{E}'.\mathsf{Enc}(ek', \boxed{sn_1}, \nu)$
    $c'_0 \leftarrow \texttt{Remelt}(td, \tilde{c}_{sn}, n^{(0)}, \vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1})$
    $\mathcal{DCL}' := \{sn_0, sn_1\}$
    $(\vec{i}^{(0)}, \vec{i}^{(1)}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo}'_{\mathrm{simple}},\texttt{Spy}}$
    Let $k$ be the size of $\vec{i}^{(0)}$
    If $k$ is different of the size of $\vec{i}^{(1)}$: return 0
    $\mathcal{CL}[1].c \leftarrow c'_0$
    Run $\texttt{S\&R}_{\mathrm{ZK}}(1, (\vec{i}^{(1)})_1)$
    Then repeat the following step for $j = 3, \ldots, (k+1)$:
        Run $\texttt{S\&R}_{\mathrm{ZK}}(j, (\vec{i}^{(1)})_{j-1})$
    Run $\texttt{Spd}_{\mathrm{ZK}}(k+1)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}^{\texttt{BDepo}'_{\mathrm{simple}}}$ ; return $b^*$
```

We define:

```
$\texttt{S\&R}_{\mathrm{ZK},tag}(j, i, n^{(0)}, sk^{(0)})$:
    $c := \mathcal{CL}[j].c$
    $u := \mathcal{CL}[j].owner$
    $n' \xleftarrow{\$} \mathcal{N}$ ; $\rho'_{sn}, \rho'_{cert}, \rho'_{pk}, \rho'_{sn\text{-}pf}, \nu'_{sn} \xleftarrow{\$} \mathcal{R}$; Compute:
        $(sn', sn\text{-}pf') \leftarrow \mathsf{T.SGen}(par_{\mathsf{T}}, \mathcal{UL}[i].sk, n')$
        $c'_{cert}, c'_{pk}, c'_{sn}, c'_{sn\text{-}pf} \leftarrow \mathsf{C.ZCm}(ck, \rho'_{cert}, \rho'_{pk}, \rho'_{sn}, \rho'_{sn\text{-}pf})$
        $\tilde{c}'_{sn} \leftarrow \mathsf{E.Enc}(ek, sn', \nu'_{sn})$
        $\pi'_{cert} \leftarrow \mathsf{C.SmPrv}\big(td, \mathsf{S.Verify}(vk', \cdot, \cdot) = 1, \rho'_{vk}, \rho'_{pk}, \rho'_{cert}\big)$
        $\pi'_{sn} \leftarrow \mathsf{C.SmPrv}_{sn}(td, pk'_{tag}, sn', sn\text{-}pf, \rho'_{pk}, \rho'_{sn}, \rho'_{sn\text{-}pf})$
        $\tilde{\pi}'_{sn} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho'_{sn}, \tilde{c}'_{sn})$
    Decompose $c$ as
    $\big(c^0, (c^j = (c^j_{pk}, c^j_{cert}, \pi^j_{cert}, c^j_{sn}, \pi^j_{sn}, c^j_{tag}, \pi^j_{tag}, \tilde{c}^j_{sn}, \tilde{c}^j_{tag}, \tilde{\pi}^j_{sn}, \tilde{\pi}^j_{tag}))^i_{j=1},$
        $n, sn, \rho_{sn}, \rho_{pk}\big)$
    $\rho_{tag}, \nu_{tag}, \rho_{t\text{-}pf} \xleftarrow{\$} \mathcal{R}$
    $(tag, t\text{-}pf) \leftarrow \mathsf{T.TGen}(par_{\mathsf{T}}, \mathcal{UL}[u].sk, n, sn')$
    $\boxed{(tag^{(0)}, P^{(0)}_{tag}) \leftarrow \mathsf{T.TGen}(par_{\mathsf{T}}, sk^{(0)}, n^{(0)}, sn')}$
    $c_{tag} \leftarrow \mathsf{C.ZCm}(ck, \rho_{tag})$
    $\tilde{c}_{tag} \leftarrow \boxed{\mathsf{E.Enc}(ek, tag^{(0)}, \nu_{tag})}$
    $\pi_{tag} \leftarrow \mathsf{C.SmPrv}_{tag}(td, pk_{tag}, sn, sn', tag, t\text{-}pf, \rho_{pk}, \rho_{sn}, \rho'_{sn}, \rho_{tag}, \rho_{t\text{-}pf})$
    $\tilde{\pi}_{tag} \leftarrow \mathsf{C.SmPrv}_{\mathrm{enc}}(td, ek, \rho_{tag}, \tilde{c}_{tag})$
    Compute $\mathsf{VER}_{\text{init}}(c^0) \wedge \bigwedge_{j=1}^{i} \mathsf{VER}_{\text{std}}(c^{j-1}, c^j) \wedge$
        $\mathsf{T.TVfy}(ck, c^i_{pk}, c'_{sn}, c_{tag}, \pi_{tag}) \wedge \mathsf{C.Verify}_{\text{enc}}(ck, ek, c_{tag}, \tilde{c}_{tag}, \tilde{\pi}_{tag})$
    Pick uniformly at random a vector of randomness $\vec{\rho''}$
    $c'' \leftarrow$
    $\mathsf{Rand}((c^0, (c^j)^i_{j=1}, c'_{pk}, c'_{cert}, \pi'_{cert}, c'_{sn}, \pi'_{sn}, c_{tag}, \pi_{tag}, \tilde{c}'_{sn}, \tilde{\pi}'_{sn}, \tilde{c}'_{tag}, \tilde{\pi}'_{tag}), \vec{\rho''})$
    $c_{new} := (c'', n', sn', \rho'_{sn} + (\vec{\rho''})_{sn'}, \rho'_{pk} + (\vec{\rho''})_{pk'})$
    $\mathcal{CL}[|\mathcal{CL}| + 1] := (i, c_{new}, 0, j)$
```

We substituted one ciphertext for another in the previous algorithm and get:

Proposition 33. *$\boldsymbol{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},0,rcca1}$ and $\boldsymbol{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},0,final}$ are ϵ_{iacr} statistically close.*

```
Experiment $\mathbf{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},0,\text{final}}(\lambda)$:
    $Gr \leftarrow \mathsf{GrGen}(1^\lambda)$
    $par_{\mathsf{T}} \leftarrow \mathsf{T.Setup}(Gr)$ ; $par_{\mathsf{S}} \leftarrow \mathsf{S.Setup}(Gr)$ ; $par_{\mathsf{S}'} \leftarrow \mathsf{S}'.\mathsf{Setup}(Gr)$
    $(ck, td) \leftarrow \mathsf{C.SmSetup}(Gr)$
    $par \leftarrow (1^\lambda, par_{\mathsf{S}}, par_{\mathsf{S}'}, par_{\mathsf{T}}, ck)$
    $(pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{BKeyGen}$
    $\mathcal{DCL}' := \emptyset$; $ctr \leftarrow 0$
    $i_0 \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}(par, pk_{\mathcal{B}}, sk_{\mathcal{W}}, sk_{\mathcal{D}})$
    Run $\mathtt{Rcv}(i_0)$ with $\mathcal{A}$ ; let $c_0$ be the received coin
    $i_1 \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}',\mathtt{Spy}}$
    Run $\mathtt{Rcv}(i_1)$ with $\mathcal{A}$ ; let $c_1$ be the received coin
    If $\mathtt{comp}(c_0, c_1) \neq 1$ abort the entire procedure
    $sn_0, n^{(0)}, \vec{\tilde{c}}_{sn,0}, \vec{\tilde{c}}_{tag,0} \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_0)$
    $sn_1, n^{(1)}, \vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1} \leftarrow \mathtt{Extract}(sk_{\mathcal{CK}}, c_1)$
    $\nu \xleftarrow{\$} \mathcal{R}$
    $\tilde{c}_{sn} \leftarrow \mathsf{E}'.\mathsf{Enc}(ek', sn_1, \nu)$
    $c'_0 \leftarrow \mathtt{Remelt}(td, \tilde{c}_{sn}, n^{(0)}, \vec{\tilde{c}}_{sn,1}, \vec{\tilde{c}}_{tag,1})$
    $\mathcal{DCL}' := \{sn_0, sn_1\}$
    $(\vec{i}^{(0)}, \vec{i}^{(1)}) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{BDepo}'_{\text{simple}},\mathtt{Spy}}$
    Let $k$ be the size of $\vec{i}^{(0)}$
    If $k$ is different of the size of $\vec{i}^{(1)}$: return 0
    $\mathcal{CL}[1].c \leftarrow c'_0$
    Run $\boxed{\mathtt{S\&R}_{\text{ZK},tag}(1, (\vec{i}^{(1)})_1, n^{(1)}, sk)}$
    Then repeat the following step for $j = 3, \ldots, (k + 1)$:
        Run $\mathtt{S\&R}_{\text{ZK}}(j, (\vec{i}^{(1)})_{j-1})$
    Run $\mathtt{Spd}_{\text{ZK}}(k + 1)$ with $\mathcal{A}$
    $b^* \leftarrow \mathcal{A}^{\mathtt{BDepo}'_{\text{simple}}}$ ; return $b^*$
```

We note that

$$\mathbf{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},0,\text{final}}(\lambda) = \mathbf{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},1,\text{ZK}}(\lambda).$$

It is $\epsilon_{\text{m-ind}}$-close to $\mathbf{Expt}^{\mathtt{c\text{-}tr}}_{\mathcal{A},1}(\lambda)$. By combining this last remark with Propositions 29, 30, 31, 32 and 33, this proves the theorem. □

B Instantiation

B.1 Instantiation and proofs of the double spending tag scheme

We will reuse the scheme introduced in [BCFK15], which we recall here.

```
$\mathsf{T.Setup}\,(Gr)$:
    – Parse $Gr$ as $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e, g_1, \hat{g})$
    – $g_2, h_1, h_2 \xleftarrow{\$} \mathbb{G}$
    – Return $(g_1 = g, g_2, h_1, h_2)$
  We define $\mathcal{M} = \{(g_1^m, \hat{g}^m) \in \mathbb{G} \times \hat{\mathbb{G}}\}_{m \in \mathbb{Z}_p}$
$\mathsf{T.KeyGen}\,(par_{\mathsf{T}})$:
    – $sk \xleftarrow{\$} \mathbb{Z}_p$
    – Return $\left(sk_{\mathsf{T}} := sk, pk_{\mathsf{T}} := \hat{g}^{sk}\right)$
$\mathsf{T.SGen}_{\mathrm{init}}\,(par_{\mathsf{T}}, sk_{\mathsf{T}}, n)$:
    – $M \leftarrow g_1^n$ ; $N \leftarrow g_2^{n+sk_{\mathsf{T}}}$
    – $M_{sn}^{(1)} = (g_1^n, \hat{g}^n)$ ; $M_{sn}^{(2)} = (g_1^{sk_{\mathsf{T}}}, \hat{g}^{sk_{\mathsf{T}}})$
    – Return $\left(sn = (M, N), M_{sn} = (M_{sn}^{(1)}, M_{sn}^{(2)})\right)$
$\mathsf{T.SGen}\,(par_{\mathsf{T}}, sk_{\mathsf{T}}, n)$:
    – $M \leftarrow g_1^n$ ; $N \leftarrow g_2^{n+sk_{\mathsf{T}}}$
    – $sn\text{-}pf = \hat{g}^n$
    – Return $(sn = (M, N), sn\text{-}pf)$
$\mathsf{T.TGen}\,(par_{\mathsf{T}}, sk, n, sn = (M, N))$:
    – $M_0 \leftarrow g_1^n$
    – $tag := \left(M^{sk} h_1^n, N^{sk} h_2^n\right)$
    – $t\text{-}pf \leftarrow \hat{g}^n$
    – Return $(tag := (A, B), t\text{-}pf)$.
$\mathsf{T.Detect}\,(sn, sn', tag, tag', \mathcal{L})$:
    – Parse $sn$ as $(M, N)$ ; parse $sn'$ as $(M', N')$
    – Parse $tag$ as $(A, B)$ ; parse $tag'$ as $(A', B')$
    – $A'' := \frac{A}{A'}$ ; $B'' := \frac{B}{B'}$
    – $M'' := \frac{M}{M'}$ ; $N'' := \frac{N}{N'}$
    – If $A'' = 0_{G_1}$ then:
            $A'' := B''$ ; $M'' := N''$
    – Search $pk_{\mathsf{T}}$ in $\mathcal{L}$ such that $e(A'', \hat{g}) = e(M'', pk)$
    – Return $(pk_{\mathsf{T}}, (A'', M''))$
$\mathsf{T.VfyGuilt}\,(pk, \pi)$:
    – Parse $\pi$ as $(A, N)$;
    – Return $\left(e(A, \hat{g}) = e(N, pk) \wedge A \neq 0_{G_1}\right)$.
$\mathsf{T.SVfy}_{\mathrm{init}}(par_{\mathsf{T}}, pk_{\mathsf{T}}, sn, M_{sn})$:
    – Parse $sn$ as $(M, N)$
    – Parse $M_{sn}$ as $\left((M_1, \hat{M}_1), (M_2, \hat{M}_2)\right)$
    – Return $\big(e(M, \hat{g})e(g_1^{-1}, \hat{M}_1) = 1_{G_T} \wedge e(M, \hat{g})e(g_2^{-1}, \hat{M}_2)e(g_2^{-1}, pk_{\mathsf{T}}) =$
      $1_{G_T} \wedge \hat{M}_2 = pk_{\mathsf{T}} \wedge e(M_1, \hat{g}) = e(g_1, \hat{M}_1) \wedge e(M_2, \hat{g}) = e(g_1, \hat{M}_2)\big)$
```

```
$\mathsf{T.SVfy}(par_\mathsf{T}, pk_\mathsf{T}, sn, sn\text{-}pf)$:
    – Parse $sn$ as $(M, N)$
    – Return $\big(e(M, \hat{g})e(g_1^{-1}, sn\text{-}pf) = 1_{G_T} \wedge$
                                  $e(N, \hat{g})e(g_2^{-1}, sn\text{-}pf)e(g_2^{-1}, pk_\mathsf{T}) = 1_{G_T}\big)$
$\mathsf{T.TVfy}(par_\mathsf{T}, pk, sn, sn', tag, t\text{-}pf)$:
    – Parse $sn$ as $(M, N)$
    – Parse $tag$ as $(A, B)$
    – Parse $sn'$ as $(M', N')$
    – Return $\big(e(M, \hat{g})e(g_1^{-1}, t\text{-}pf) = 1_{G_T} \wedge$
                                  $e(A, \hat{g}^{-1})e(M', pk)e(h_1, t\text{-}pf) = 1_{G_T} \wedge$
                                  $e(B, \hat{g}^{-1})e(N', pk)e(h_2, t\text{-}pf) = 1_{G_T}\big)$
```

Proofs

Theorem 34. *This above scheme is extractable, bootable, SN-verifiable, tag-verifiable and $\mathcal{N}$ injective.*

These properties are all straightforward to show and we therefore omit the proof.

Proposition 35. *The above scheme is SN-collision-resistant.*

Let $(pk, sn\text{-}pf)$ and $(pk', sn\text{-}pf')$ such that for some sn:

$$\mathsf{T.SVfy}(par_\mathsf{T}, pk, sn, sn\text{-}pf) = \mathsf{T.SVfy}(par, pk', sn, sn\text{-}pf') = 1.$$

We parse sn as (M, N), and deduce the followings equations:

$$e(M, \hat{g}) = e(g_1, sn\text{-}pf) = e(g_1, sn\text{-}pf'),$$

from which we get $sn\text{-}pf = sn\text{-}pf'$. Then we can deduce

$$e(M, \hat{g})e(g_2^{-1}, sn\text{-}pf)e(g_2^{-1}, pk) = 0_{G_T} = e(M, \hat{g})e(g_2^{-1}, sn\text{-}pf)e(g_2^{-1}, pk')$$

and thus $e(g_2^{-1}, pk) = e(g_2^{-1}, pk')$, which finally yields $pk = pk'$. The reasoning is analogous for $\mathsf{T.SVfy}_{\mathrm{init}}$. □

To prove the two other results, we will use the following lemma.

```
$\mathbf{Expt}^{Tuple}_{\mathcal{A},b}(\lambda)$:
    $((\mathbb{G}, g_1, q), (\hat{\mathbb{G}}, \hat{g}, q), e) \leftarrow \mathsf{GrGen}(1^\lambda)$
    $g_2, h_1, h_2 \xleftarrow{\$} \mathbb{G}$
    $b' \leftarrow \mathcal{A}^{O'_b}$
    Return $b = b'$

$O'_0$:
    $n \xleftarrow{\$} \mathbb{Z}_q$
    Return $(g_1^n, g_2^n, h_1^n, h_2^n)$
$O'_1$:
    $n_1, n_2, n_3, n_4 \xleftarrow{\$} \mathbb{Z}_q$
    Return $(g_1^{n_1}, g_2^{n_2}, h_1^{n_3}, h_2^{n_4})$
```

Lemma 36. *For any adversary $\mathcal{A}$ against the game Tuple defined above with advantage ϵ, there exists an adversary $\mathcal{B}$ against DDH in $\mathbb{G}$ with advantage ϵ_{DDH} such that $\frac{\epsilon}{3K} \leq \epsilon_{\mathrm{DDH}}$, with K the number of oracles calls to O'_b.*

We define the following oracles:

```
$O'_{0.3}$:
    $n, n_2 \xleftarrow{\$} \mathbb{Z}_q$
    Return $(g_1^n, g_2^{n_2}, h_1^n, h_2^n)$

$O'_{0.7}$:
    $n, n_2, n_3 \xleftarrow{\$} \mathbb{Z}_q$
    Return $(g_1^n, g_2^{n_2}, h_1^{n_3}, h_2^n)$
```

Viewing the tuples (g_2, g_1^n, g_2^n), (h_1, g_1^n, h_1^n) and (h_2, g_1^n, h_2^n) as DDH challenge tuples, we get taht the adversary cannot distinguish O'_0 from $O'_{0.3}$ nor $O'_{0.3}$ from $O'_{0.7}$, or $O'_{0.7}$ from O'_1, with probability more than ϵ_{DDH} each respectively. (Note that we can compute all other elements, since we know the discrete logarithms of the elements which are not part of the respective DDH-tuple; and we can answer all other oracle calls honestly). This proves the lemma. □

Corollary 37. *No adversary can break* `tag-anonymity` *with an advantage better than* $6K\epsilon_{\mathrm{DDH}}$*, where* K *is the number of calls to* O_1*.*

```
$\mathbf{Expt}_{\mathcal{A},b}^{\texttt{Perfect-tag-anonymity}}((\mathbb{G}, g_1, q), (\hat{\mathbb{G}}, \hat{g}, q), e)$:
$par_{\mathsf{T}} \leftarrow ((\mathbb{G}, g_1, q), (\hat{\mathbb{G}}, \hat{g}, q), e)$
$(sk_0, sk_1) \leftarrow \mathcal{A}(par_{\mathsf{T}})$
$k := 0$
$b^* \leftarrow \mathcal{A}^{O_1^{\text{perfect}}(sk_b), O_2^{\text{perfect}}(sk_b, \cdot, \cdot)}(par_{\mathsf{T}}, sk_0, sk_1)$
Return $(b = b^*)$

$O_1^{\text{perfect}}(sk)$:
$(g_3, g_4, g_5, g_6) \leftarrow O'_0$
$T[k] := (g_5, g_6)$
Return $\left(g_3, g_1^{sk} \cdot g_4\right)$

$O_2^{\text{perfect}}(sk, sn', t)$:
If $t > k$ abort entire game
$(g_5, g_6) \leftarrow T[t]$
$(N, M) \leftarrow sn'$
Return $(N g_5, M g_6)$
```

Lemma 36 implies that the adversary cannot, except with probability $3K\epsilon_{\mathrm{DDH}}$, distinguish `Perfect-tag-anonymity` from `tag-anonymity`: if we replace O'_0 by O'_1, the game becomes exactly `tag-anonymity`. In the latter game, we can replace b by $(1 - b)$ without changing the distribution of the adversary's input. □

Theorem 38. *Let* $\mathcal{A}$ *be an adversary that wins the exculpability game with probability* ϵ *after* K *oracle calls to* O_1*, then there exist* $\mathcal{B}_1$ *against DDH in* $\mathbb{G}$ *with advantage* ϵ_{DDH} *and* $\mathcal{B}_2$ *against DDH in* $\hat{\mathbb{G}}$ *with advantage* $\hat{\epsilon}_{\mathrm{DDH}}$*, such that:*

$$\epsilon \leq 3K\epsilon_{\mathrm{DDH}} + \hat{\epsilon}_{\mathrm{DDH}}.$$

Using the same argument as in Corollary 37, we deduce, incurring a loss of $3K\epsilon_{\mathrm{DDH}}$, that we can consider that oracle calls do not yield any information to the adversary. After receiving a triple $(\hat{g}_1, \hat{g}_2, \hat{g}_3)$ in $\hat{\mathbb{G}}$, we send $\hat{g}_1$ as the public key. When we receive (N, A) such that

$$e(N, pk) = e(A, \hat{g})$$

with $A \neq 0_{\mathbb{G}_1}$, this means that $A = N^{\log_{g_1}(pk)}$ and we can check if $e(N, \hat{g}_3) = e(A, \hat{g}_2)$ to decide whether we received a DDH triple or not. □

Efficiency We summarize all the efficiency results as follows (where "m.s.w.u" means multiscalar with unkown):

$\lvert par_{\mathsf{T}}\rvert$	$3\lvert\mathbb{G}\rvert$
$\lvert sk_{\mathsf{T}}\rvert$	$\lvert\mathbb{Z}_p\rvert$
$\lvert pk_{\mathsf{T}}\rvert$	$\lvert\hat{\mathbb{G}}\rvert$
$\lvert sn\rvert = \lvert tag\rvert$	$2\lvert\mathbb{G}\rvert$
$\lvert sn\text{-}pf\rvert = \lvert t\text{-}pf\rvert$	$\lvert\hat{\mathbb{G}}\rvert$
π	$2\lvert\mathbb{G}\rvert$
Number of pairing equations in $\mathsf{T.SVfy}$	2 generic eq.
Number of pairing equations in $\mathsf{T.SVfy}_{\text{init}}$	4 generic, 1 m.s.w.u eq. in $\hat{\mathbb{G}}$
Number of pairing equations in $\mathsf{T.TVfy}$	3 generic eq.
$\lvert\pi_{sn}\rvert$	$8\lvert\mathbb{G}\rvert + 8\lvert\hat{\mathbb{G}}\rvert$
$\lvert\pi_{sn,\text{init}}\rvert$	$16\lvert\mathbb{G}\rvert + 16\lvert\hat{\mathbb{G}}\rvert + 2\lvert\mathbb{Z}_p\rvert$
$\lvert\pi_{tag}\rvert$	$12\lvert\mathbb{G}\rvert + 12\lvert\hat{\mathbb{G}}\rvert$
$\lvert\tilde{\pi}_{sn}\rvert$	$8\lvert\mathbb{G}\rvert + 10\lvert\hat{\mathbb{G}}\rvert$
$\lvert\tilde{\pi}_{tag}\rvert$	$12\lvert\mathbb{G}\rvert + 14\lvert\hat{\mathbb{G}}\rvert$

B.2 Instantiation of the encryption scheme E'

Construction overview. We roughly follow the framework proposed by Chase et al. [CKLM12]. The first part of the ciphertext is an encryption

$$\vec{C} = (c_0, c_1, \dots, c_{n+1}) = (f^\theta, g^\theta, \{h_i^\theta \cdot m_i\}_{i=1}^n)$$

of the message vector $\vec{m} = (m_1, \dots, m_n) \in \mathbb{G}_n$. As in [LPQ17], we use the same one-time linearly homomorphic structure-preserving signature scheme [LPJY13] $\mathsf{LHSPS} = (\mathsf{KeyGen}, \mathsf{Sign.Verify})$, for which let

$$\vec{v}_1 = (f, g, 1, \dots, 1), \vec{v}_2 = (1, 1, 1, g, h_1, \dots, h_n),$$

the signing key of the LHSPS is composed of two linearly homomorphic signatures of $\vec{v}_1$ and $\vec{v}_2$, that is, $sk = (\sigma_{\vec{v}_1}, \sigma_{\vec{v}_2})$. Using this signing key, anyone can generate the signature $\sigma_{\vec{m}}$ of any message $\vec{m} \in Span(\vec{v}_1, \vec{v}_2)$.

The second part of the ciphertext is a zero-knowledge proof for the language

$$\mathcal{L}_\vee = \big\{(\vec{C}, (b, \theta, \{m_i\}_{i=1}^n, \sigma_{\vec{v}})) \bigm| b \in \{0,1\} \vee \mathsf{LHSPS.Verify}(vk_{\mathsf{LHSPS}}, \sigma_{\vec{v}}, \vec{v}) = 1\big\}.$$

where $\vec{v} = (c_0^b, c_1^b, g^{1-b}, c_1^{(1-b)}, c_2^{(1-b)}, \dots, c_{n+1}^{1-b})$. Note that when $b = 1$, $\vec{v} \in Span(\vec{v}_1, \vec{v}_2)$ means that $\log_f(c_0) = \log_g(c_1)$, then the ciphertext is a valid ciphertext. Also note that signatures on $\vec{v}$ with $b = 1$ will only be generated in the security proof.

To enable re-randomization, we generate a signature $\sigma_{\vec{w}}$ on the vector $\vec{w} = (f^b, g^b, 1, g^{(1-b)\cdot\theta}, h_1^{(1-b)\cdot\theta}, \dots, h_n^{(1-b)\cdot\theta})$ and add a zero-knowledge proof of knowledge of the valid signature $\sigma_{\vec{w}}$. It is easy to see that with $\sigma_{\vec{w}}$, we can generate signatures for all re-randomization of the vector $\vec{v}$.

One-time linearly homomorphic structure-preserving signature. To construct the re-randomizable CCA encryption scheme, we need the one-time linearly homomorphic structure-preserving signature.

Definition 39 ((One-time) linearly homomorphic structure-preserving signature [LPJY13]). *A one-time linearly homomorphic structure-preserving signature is tuple of* 4 *algorithms* $\mathsf{LHSPS} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{SignDerive}, \mathsf{Verify})$ *with the following specifications:*

$\mathsf{Setup}\,(Gr, n)$ *is a probabilistic algorithm taking the group parameter* Gr *and an integer* n *denoting the dimension of the message to be signed. It outputs the public verification key* vk *and the signature key* sk.

$\mathsf{Sign}\,(sk, \vec{m})$ *is a deterministic algorithm that takes the signing key* sk *and the message* $\vec{m} \in \mathbb{G}^n$*, and outputs a signature* σ.

$\mathsf{SignDerive}\big(vk, \{(w_i, \sigma^{(i)})\}_{i=1}^{\ell}\big)$ *is a deterministic algorithm taking the verification key* vk *and* ℓ *pairs* $(w_i\sigma^{(i)})$ *where* $w_i \in \mathbb{Z}_p$ *and* $\sigma^{(i)}$ *is an LHSPS signature. It outputs a signature* σ *on the message* $\vec{m} = \prod_{i=1}^{\ell} \vec{m}_i^{w_i}$.

$\mathsf{Verify}\,(vk, \vec{m}, \sigma)$ *is a deterministic algorithm taking the verification key* vk*, the message vector* $\vec{m}$ *and a signature* σ*. It outputs* 1 *if the signature is valid,* 0 *otherwise.*

Definition 40 (One-time unforgeability). *A one-time linearly homomorphic SPS scheme* $\Sigma = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *is secure if no adversary has non-negligible advantage in the following game:*

1. *The adversary* $\mathcal{A}$ *outputs an integer* n*, sends it to the challenger* $\mathcal{C}$*. The challenger generates* $(vk, sk) \leftarrow \mathsf{Setup}(1^{\lambda}, n)$ *and sends the public verification key back to* $\mathcal{A}$.
2. *The adversary* $\mathcal{A}$ *has access to the signing oracle*
   - $\mathsf{Sign}(sk, \cdot)$*:* $\mathcal{A}$ *can request the challenger* $\mathcal{C}$ *to sign the message vectors* $\{\vec{m}_i\}_{i=1}^{Q_s}$ *where* Q_s *denotes the number of signing queries.*
3. $\mathcal{A}$ *outputs* $(\vec{m}^{\star}, \sigma_{\star})$*. The adversary wins if and only if* $\mathsf{Verify}(vk, \vec{m}^{\star}, \sigma_{\star}) = 1$ *and* $\vec{m}^{\star} \notin Span(\{\vec{m}_i\}_{i=1}^{Q_s})$.

We recall the following construction of the one-time linearly homomorphic structure-preserving signature scheme.

- $\mathsf{LHSPS.Setup}\,(Gr, n)$:
  1. Parse Gr as $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$.
  2. Chose $\hat{g}_z, \hat{g}_r \xleftarrow{\$} \hat{\mathbb{G}}$. For $i \in \{1, \ldots, n\}$, randomly chose χ_i, γ_i and compute $\hat{g}_i = \hat{g}_z^{\chi_i} \hat{g}_r^{\gamma_i}$.
  3. Output the verification key $pk = (\hat{g}_z, \hat{g}_r, \{\hat{g}_i\}^n) \in \hat{\mathbb{G}}^{n+2}$ and the signing key $sk = (\{\chi_i, \gamma_i\}_{i=1}^n)$.
- $\mathsf{LHSPS.Sign}\big(vk, sk, \vec{M}\big)$:
  1. Parse the verification key $vk = (\hat{g}_z, \hat{g}_r, \{\hat{g}_i\}^n) \in \hat{\mathbb{G}}^{n+2}$, the signing key $sk = (\{\chi_i, \gamma_i\}_{i=1}^n)$ and the message $\vec{M} = (M_1, \ldots, M_n) \in \mathbb{G}^n$.

2. Output the signature $\vec{\sigma} = (z, r) \in \mathbb{G}^2$ such that $z = \prod_{i=1}^{n} M_i^{\chi_i}$ and $r = \prod_{i=1}^{n} M_i^{\gamma_i}$.

– $\mathsf{LHSPS.SignDerive}\big(vk, (\vec{\sigma}, \{w_i, \vec{\sigma^{(i)}}\}_{i=1}^{\ell})\big)$:
  1. For all $i \in \{1, \ldots, \ell\}$, parse $\sigma^{(i)}$ as (z_i, r_i).
  2. Output the signature $\sigma = (\prod_{i=1}^{\ell} z_i^{w_i}, \prod_{i=1}^{\ell} r_i^{w_i})$.

– $\mathsf{LHSPS.Verify}(vk_{\mathsf{LHSPS}}, \sigma)$:
  1. Parse the signature as $\sigma = (z, r)$ and the message $\vec{M} = (M_1, \ldots, M_n)$.
  2. Return 1 iff $(M_1, \ldots, M_n) \neq (1_{\mathbb{G}}, \ldots, 1_{\mathbb{G}})$ and the following equation is verified.

$$e(z, \hat{g}_z) \cdot e(r, \hat{g}_r) = \prod_{i=1}^{n} e(M_i, \hat{g}_i).$$

Theorem 41 ([LPJY13, Theorem 1]). *The above construction of a one-time linearly homomorphic structure-preserving signature scheme is unforgeable if the SXDH assumption holds in the underlying group.*

The above scheme was proven to be unforgeable under the DP assumption, which is implied by the SXDH assumption. As in the remaining part of the construction of RCCA requires SXDH to hold, we state this theorem with SXDH assumption.

Replayable-CCA encryption scheme. An RCCA encryption scheme E consists of six PPT algorithms $\mathsf{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReRand}, \mathsf{Dec}, \mathsf{Verify}, \mathsf{AdptPrf})$. It should verify the following specifications:

– $\mathsf{E.KeyGen}\,(Gr)$: a randomized algorithm which takes as input the group description and outputs an encryption public key pk and a corresponding decryption key dk.
– $\mathsf{E.Enc}\,(pk, m, \nu)$: a randomized encryption algorithm which takes as input a public encryption key pk, a plaintext (from a plaintext space), some randomness and outputs a ciphertext.
– $\mathsf{E.ReRand}\,(pk, c, \nu)$: a randomized algorithm which takes as input a public key, a ciphertext and some randomness,. and outputs another ciphertext.
– $\mathsf{E.Dec}\,(dk, c)$: a deterministic decryption algorithm which takes a decryption key and a ciphertext, and outputs either a plaintext or an error indicator $\perp$.
– $\mathsf{E.Verify}\,(pk, m, \rho, c)$: a deterministic algorithm which takes as input a public key, a message, some randomness, and a ciphertext and outputs a bit.
– $\mathsf{E.AdptPrf}\,(ck, pk, c_M, c, (\pi, c_\nu)\,, \nu')$ a randomized algorithm which takes as input a commitment key, an encryption public key, a commitment, an equality proof (i.e a Groth-Sahai proof and a commitment), a ciphertext, a proof, some randomness, and outputs an equality proof.

We give the following explicit construction of the RCCA scheme supporting encryption of vectors of group elements.

$\mathsf{E.KeyGen}\,(Gr)$:

1. Parse Gr as $(\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$.
2. Choose two random group elements $f, g \xleftarrow{\$} \mathbb{G}^2$.
3. Choose random exponents $\{\alpha_i\}_{i=1}^n \xleftarrow{\$} \mathbb{Z}_p$ and compute $\{h_i\}_{i=1}^n = g^{\alpha_i}$.
4. Generate the Groth-Sahai crs $c\vec{r}s_{GS}$ by choosing random $\vec{u}_1, \vec{u_2} \xleftarrow{\$} \mathbb{G}^2$ and $\hat{\vec{u}}_1, \hat{\vec{u}}_2 \xleftarrow{\$} \hat{\mathbb{G}}^2$.
5. Define two vectors $\vec{v}_1, \vec{v}_2$ such that

$$\vec{v}_1 = (f, g, 1, 1, \dots, 1) \in \mathbb{G}^{n+2} \qquad \vec{v}_2 = (1, 1, 1, h_1, \dots, h_n) \in \mathbb{G}^{n+2},$$

   then generate two LHSPS signatures $\sigma_{\vec{v}_1}$ and $\sigma_{\vec{v}_2}$ which will be used to proof that a vector is in the $Span(\vec{v}_1, \vec{v}_2)$. together with the signing key $t\vec{k}$.
6. Output the decryption key $dk = \alpha$ and the public key

$$pk = (f, g, \{h^{\alpha_i}\}_{i=1}^n, c\vec{r}s_{GS}, \sigma_{\mathsf{LHSPS}} = (\sigma_{\vec{v}_1}, \sigma_{\vec{v}_2})).$$

   Notice that the LHSPS signing key $t\vec{k}$ will never be published by the key generation algorithm, it will only be used in the security proofs.

$\mathsf{E.Enc}\,(pk, m, \nu)$:

1. Randomly pick a number $\theta \in \mathbb{Z}_p$. Compute $\vec{C} = (c_0, c_1, \dots, c_{n+1}) = (f^\theta, g^\theta, M_1 \cdot h_1^\theta, \dots, M_n \cdot h_n^\theta)$.
2. Define the bit $b = 1$ and denote $G = g^b \in \mathbb{G}$ and $\hat{G} = \hat{g}^b \in \hat{\mathbb{G}}$.
3. Generate the Groth-Sahai proof π_b of

$$e(G, \hat{g}) = e(g, \hat{G})$$

4. For all $i \in \{1, \dots n{+}1\}$, compute $\Theta_i = c_i^b$. Compute also the Groth-Sahai π_Θ proof of the equations:

$$e(\Theta_i, \hat{g}) = e(c_i, \hat{G})$$

5. Define the vector $\vec{v} = (c_0^b, c_1^b, g^{1-b}, c_1^{1-b}, \dots, c_{n+1}^{1-b})$. Generate a LHSPS signature $\sigma_{\vec{v}}$ such that $\vec{v} \in Span(\vec{v}_1, \vec{v}_2)$.
6. Compute a Groth-Sahai proof $\pi_{\vec{v}}$ of the validity of the LHSPS signature $\sigma_{\vec{v}}$.
7. To enable the re-randomization, compute

$$(F, G, \{H_i\}_{i=1}^n) = (f^b, g^b, \{h_i^b\}_{i=1}^n)$$

   and Groth-Sahai proof π_{FGH} of them.
8. Define the vector $\vec{w} = (f^b, g^b, 1, h_1^{1-b}, \dots, h_n^{1-b})$. Compute a LHSPS signature $\sigma_{\vec{w}}$ of the fact that $\vec{w} \in Span(\vec{v}_1, \vec{v}_2)$.
9. Generate a Groth-Sahai proof $\pi_{\vec{w}}$ of the validity of LHSPS signature $\sigma_{\vec{w}}$.
10. Output the ciphertext $c = (\{c_i\}_{i=1}^n, \pi_b, \pi_\theta, \pi_{\vec{v}}, \pi_{FGH}, \pi_{\vec{w}})$.

$\mathsf{E.ReRand}\,(pk, c, \nu)$:

1. Parse $c = (\{c_i\}_{i=1}^{n+1}, \pi_b, \pi_\theta, \pi_{\vec{v}}, \pi_{FGH}, \pi_{\vec{w}})$.
2. Compute $c'_0 = c_0 \cdot f^\nu$, $c'_1 = c_1 \cdot g^\nu$ and for $i \in \{2, \dots, n+1\}$, compute $c'_i = c_i \cdot h^\nu_{i-1}$.
3. We update the proof π_b, π_θ using the commitment C_F, C_G, C_H in π_{FGH} to get π'_b, π'_θ.
4. We update the commitment of the LHSPS signature $C'_{\sigma_{\vec{v}}} = C_{\sigma_{\vec{v}}} \cdot C^\nu_{\sigma_{\vec{w}}}$ and the update the proof $\pi_{\vec{v}}$ accordingly to get $\pi'_{\vec{v}}$.
5. We re-randomize all the updated Groth-Sahai proofs
$$\pi'_b, \pi'_\theta, \pi'_{\vec{v}}, \pi_{FGH}, \pi_{\vec{w}}$$
to get the new proofs $\pi''_b, \pi''_\theta, \pi''_{\vec{v}}, \pi''_{FGH}, \pi''_{\vec{w}}$.
6. Output the new ciphertext $c' = (\{c'_i\}_{i=1}^{n}, \pi''_b, \pi''_\theta, \pi''_{\vec{v}}, \pi''_{FGH}, \pi''_{\vec{w}})$.

$\mathsf{E.Dec}\,(dk, c)$:
1. Parse c as $\sigma = (\{c_i\}_{i=1}^{n}, \pi_b, \pi_\theta, \pi_{\vec{v}}, \pi_{FGH}, \pi_{\vec{w}})$.
2. Check all proofs $(\pi_b, \pi_\theta, \pi_{\vec{v}}, \pi_{FGH}, \pi_{\vec{w}})$ are valid.
3. For $i \in \{1, \dots n\}$, compute $M_i = c_{i+1}/(c_1^{\alpha_i})$.
4. Output $\{M_i\}_{i=1}^{n}$.

$\mathsf{E.Verify}\,(pk, \vec{m}, \nu, c)$:
1. Parse c as $\sigma = (\{c_i\}_{i=1}^{n}, \pi_b, \pi_\theta, \pi_{\vec{v}}, \pi_{FGH}, \pi_{\vec{w}})$.
2. Verify that $\pi_b, \pi_\theta, \pi_{\vec{v}}, \pi_{FGH}, \pi_{\vec{w}}$ are all correct.
3. Verify the following pairing equations:
$$c_0 = g^\nu \qquad c_1 = f^\nu \qquad c_{i+1} = h^\nu \cdot m_i.$$
where $i \in \{1, \dots, n\}$.

$\mathsf{E.AdptPrf}\,(ck, pk, c_M, c, (\pi, c_\nu), \nu')$:
1. We just update the Groth-Sahai proof the new randomness ν' by multiplying $c'_\nu = c_\nu \cdot \hat{g}^\nu$.
2. As the equality proofs consists of the following pairing equations:
$$c_0 = g^\nu \qquad c_1 = f^\nu \qquad c_{i+1} = h^\nu \cdot m_i.$$
where $i \in \{1, \dots, n\}$.

Encryption key	$(10+n)\mathbb{G} + 4\hat{\mathbb{G}}$
Decryption key	$n\mathbb{Z}_p$
Ciphertext	$(6n+19)\mathbb{G} + (16+4n)\hat{\mathbb{G}}$
Verification equations	2 linear + n quadratic
Size of the equality proof	$(2+2n)\mathbb{G} + (2+4n)\hat{\mathbb{G}}$

Proof (of Theorem 16). The completeness and the correctness of the above RCCA encryption scheme are straightforward to verify. We will focusing on the Replayable-CCA property.

We proceed by the series of hybrid games $\mathsf{Game}_0, \dots, \mathsf{Game}_5$, we denote by Adv_i the advantage of the adversary $\mathcal{A}$ to win the game Game_i.

Game_0: We have Game_0 is identical to the original RCCA security game and thus by definition:

$$\mathsf{Adv}_0 = \mathsf{Adv}^{\mathtt{RCCA}}_{\mathcal{A}}(1^\lambda)$$

Game_1: In this game, we will modify the challenge ciphertext provided to the adversary in the RCCA security game. The new challenge ciphertext is:

$$c^\star = (\{c_i^\star\}_{i=1}^n, \pi_b^\star, \pi_\theta^\star, \pi_{\vec{v}}^\star, \pi_{FGH}^\star, \pi_{\vec{w}}^\star)$$

We only modify $\pi_{\vec{v}}^\star$ and $\pi_{\vec{w}}^\star$. Instead of generating these two proofs using the signing key $(\sigma_{\vec{v}_1}, \sigma_{\vec{v}_2})$ of the LHSPS, we will use the signing key td to directly compute the signatures of $\vec{v}^\star$ and $\vec{w}$, where $\vec{v}^\star = (1, 1, g, c_2, \ldots, c_{n+1})$. (Notice that the secret signing key is never used in the real game.)

As this change is only conceptional, the distribution of the challenge ciphertext is identical in Game_1 as in Game_0. We have $\mathsf{Adv}_1 = \mathsf{Adv}_0$

Game_2: In this game, we modify the $\vec{crs}$ of the Groth-Sahai proof system. We generate two random values $\xi, \zeta \xleftarrow{\$} \mathbb{Z}_p$, then compute $\vec{u}_1, \vec{u}_2, \hat{\vec{u}}_1, \hat{\vec{u}}_2$ such that $\vec{u}_1 = \vec{u}_2^{\xi}$ and $\hat{\vec{u}}_1 = \hat{\vec{u}}_2^{\zeta}$.

Notice that this is the perfect sound setting of the Groth-Sahai proof system. ξ and ζ can be used to extract the witness. Since the only difference between Game_1 and Game_2 is the change of $\vec{u}_1, \vec{u}_2, \hat{\vec{u}}_1, \hat{\vec{u}}_2$, the indistinguishability can be proven using the SXDH assumption. Thus, we have $\mathsf{Adv}_2 \leq \mathsf{Adv}_1 + 2 \cdot \mathsf{Adv}_{SXDH}$.

Game_3: In this game, we modify the decryption oracle. We will add a manual verification of the underlying LHSPS for the decryption queries. To do this, since the Groth-Sahai proof is settled in the soundness mode ($\vec{u}_1 = \vec{u}^{\xi}$ and $\hat{\vec{u}}_1 = \hat{\vec{u}}_2^{\zeta}$). We can use the trapdoors ξ, ζ to extract the witness in the commitments of the Groth-Sahai proof. We extract $\vec{v}$ and $\sigma_{\vec{v}} = (z, r)$ from the proof $\pi_{\vec{v}}$. We use the signing key td of the linearly homomorphic structure-preserving signature $\sigma_{\vec{v}}^\dagger = (z^\dagger, r^\dagger)$ to generate a signature $\sigma_{\vec{v}}^\dagger$ of the vector $\vec{v}$. The challenger will reject the decryption query if $\sigma_{\vec{v}}^\dagger \neq \sigma_{\vec{v}}$.

We can see that, if an adversary can distinguish Game_3 from Game_2 then he can forge a valid signature of the underlying LHSPS. Since the unforgeability of the LHSPS is based on the SXDH problem, we have $\mathsf{Adv}_3 \leq \mathsf{Adv}_2 + \mathsf{Adv}_{DP}(1^\lambda)$.

Game_4: We will modify all the decryption oracles (both pre-challenge and post-challenge ones) to avoid the use of $\log_g(h_i) = \alpha_i$. After making these changes, we can modify the generation of h_i to $h_i = f^{x_i} g^{y_i}$.

Pre-challenge decryption queries: We use the trapdoor of the Groth-Sahai proof to extract the witness of the proof, if we have $b = 0$ then we directly reject the proof.

Post-challenge decryption queries: We also use the trapdoor of the Groth-Sahai proof to extract the witness of the proof, if $b = 0$ and the ciphertext is not rejected by the rule of Game_3, the challenger outputs Replay. Additionally, both in pre-challenge and post-challenge decryption queries.

Since we don't have α_i anymore, we decrypt the ciphertext by computing $M_i = c_{i+1}/(c_0^{x_i} \cdot c_1^{y^i})$.
We now analyse the change of the decryption oracles:

Pre-challenge: It is easy to see that in case of $b = 0$, the challenger only issued two LHSPS signatures of $\vec{v}_1$ and $\vec{v}_2$. And the vector $\vec{v}$ is clearly not in the span of $Span(\vec{v}_1, \vec{v}_2)$. So the adversary is statistically impossible to forge a correct signature.

Post-challenge: Note that the Groth-Sahai proof is in the perfect soundness setting of the Groth-Sahai proof, thus the challenger $\mathcal{C}$ can use the trapdoor to extract all the witness used in the proof. We will now separate two case:

- If $g^b = 1$, we have $\vec{v} = (c_0, c_1, 1, 1, \ldots, 1)$. But $\vec{c}$ is not rejected in the Game_3, with a overwhelming probability, we will have $\vec{v} \in Span(\vec{v}_1)$. Thus we have $M_i = c_{i+1}/(c_0^x \cdot c_1^y)$.
- If $g^b = 0$, we have $\vec{v} = (1, 1, g, c_2, \ldots, c_{n+1})$. As the third element is g, $\vec{v} = \vec{v} \cdot \vec{v}_2^{\theta} \cdot \vec{w}^{\rho}$. This means that $\vec{v}$ is a randomization of $\vec{v}^{\star}$, thus we can answer *Replay* to the adversary.

Game_5: We modify the distribution of the challenge ciphertext. Instead of choosing them as an encryption of $\vec{M}_0$ or $\vec{M}_1$. We Choose them all random elements. By the self-rerandomizability of the DDH assumption in $\mathbb{G}$, the game 5 is indistinguishable from the game 4.

During the Game_5, as the challenge ciphertext is only random group elements, the adversary cannot have more advantage than a random guess.

B.3 Instantiation of the encryption scheme E

Let $Gr = (p, \mathbb{G}, g)$.

$\mathsf{E.KeyGen}()$:
- $(dk_1, dk_2) \xleftarrow{\$} \mathbb{Z}_p^2$
- Return $((g^{dk_1}, g^{dk_2}), (dk_1, dk_2))$

$\mathsf{E.Enc}((D_1, D_2), (M_1, M_2), \nu)$:
- Return $(g^{\nu}, M_1 \cdot D_1^{\nu}, M_2 \cdot D_2^{\nu})$

$\mathsf{E.ReRand}((D_1, D_2), (C_0, C_1, C_2), \nu)$:
- Return $(C_0 \cdot g^{\nu}, C_1 \cdot D_1^{\nu}, C_2 \cdot D_2^{\nu})$

$\mathsf{E.Dec}((dk_1, dk_2), (C_0, C_1, C_2))$:
- Return $(C_0 \cdot C_1^{dk_1}, C_2 \cdot C_0^{dk_2})$

$\mathsf{E.Verify}((D_1, D_2), (M_1, M_2), \nu, (C_0, C_1, C_2))$:
- Return $(g^{\nu}, M_1 \cdot D_1^{\nu}, M_2 \cdot D_2^{\nu}) = (C_0, C_1, C_2)$

$\mathsf{E.AdptPrf}(ck, ek, (com_{M_1}, com_{M_2}), c, \tilde{\pi} = (\pi, com_{\nu}), \nu')$:
- Analog to B.2

Proposition 42. *If there exists an adversary $\mathcal{A}$ that breaks the IACR property of the scheme with advantage ϵ_{IACR}, then there exists an adversary $\mathcal{B}$ that breaks SXDH with advantage ϵ_{SXDH}, with*

$$\epsilon_{\text{IACR}} \leq 4\, \epsilon_{\text{SXDH}}.$$

We define the following experiments:

```
$\mathbf{Expt}^{\texttt{IACR}}_{\mathcal{A},b}((\mathbb{G},g,p))$:
    $((P_1,P_2),(dk_1,dk_2)) \leftarrow \mathsf{KeyGen}(Gr)$
    $((C_0^{(0)},C_1^{(0)},C_2^{(0)}),(C_0^{(1)},C_1^{(1)},C_2^{(1)})) \leftarrow \mathcal{A}((P_1,P_2))$
    $\nu \xleftarrow{\$} \mathbb{Z}_p$
    $(C_0,C_1,C_2) \leftarrow (C_0^{(b)} \cdot g^{\nu}, C_1^{(b)} \cdot P_1^{\nu}, C_2^{(b)} \cdot P_2^{\nu})$
    $b' \leftarrow \mathcal{A}(C_0,C_1,C_2)$
    Return $b'$
$\mathbf{Expt}^{\texttt{IACR}V2}_{\mathcal{A},b}((\mathbb{G},g,p))$:
    $((P_1,P_2),(dk_1,dk_2)) \leftarrow \mathsf{KeyGen}(Gr)$
    $((C_0^{(0)},C_1^{(0)},C_2^{(0)}),(C_0^{(1)},C_1^{(1)},C_2^{(1)})) \leftarrow \mathcal{A}((P_1,P_2))$
    $\nu,\nu_2 \xleftarrow{\$} \mathbb{Z}_p$
    $(C_0,C_1,C_2) \leftarrow (C_0^{(b)} \cdot g^{\nu}, C_1^{(b)} \cdot P_1^{\nu_2}, C_2^{(b)} \cdot P_2^{\nu})$
    $b' \leftarrow \mathcal{A}(C_0,C_1,C_2)$
    Return $b'$
$\mathbf{Expt}^{\texttt{IACR}V3}_{\mathcal{A},b}((\mathbb{G},g,p))$:
    $((P_1,P_2),(dk_1,dk_2)) \leftarrow \mathsf{KeyGen}(Gr)$
    $((C_0^{(0)},C_1^{(0)},C_2^{(0)}),(C_0^{(1)},C_1^{(1)},C_2^{(1)})) \leftarrow \mathcal{A}((P_1,P_2))$
    $\nu,\nu_2,\nu_3 \xleftarrow{\$} \mathbb{Z}_p$
    $(C_0,C_1,C_2) \leftarrow (C_0^{(b)} \cdot g^{\nu}, C_1^{(b)} \cdot P_1^{\nu_2}, C_2^{(b)} \cdot P_2^{\nu_3})$
    $b' \leftarrow \mathcal{A}(C_0,C_1,C_2)$
    Return $b'$
```

By noticing that $|\Pr(\mathbf{Expt}^{\texttt{IACR}}_{\mathcal{A},b}((\mathbb{G},g,p)) = 1) - \Pr(\mathbf{Expt}^{\texttt{IACR}V2}_{\mathcal{A},b}((\mathbb{G},g,p)) = 1)|$ and $|\Pr(\mathbf{Expt}^{\texttt{IACR}V2}_{\mathcal{A},b}((\mathbb{G},g,p)) = 1) - \Pr(\mathbf{Expt}^{\texttt{IACR}V3}_{\mathcal{A},b}((\mathbb{G},g,p)) = 1|$ are less or equal to ϵ_{SXDH}, and because $\mathbf{Expt}^{\texttt{IACR}V3}_{\mathcal{A},0}((\mathbb{G},g,p))$ and $\mathbf{Expt}^{\texttt{IACR}V3}_{\mathcal{A},1}((\mathbb{G},g,p))$ are distributed equally, we deduce $\epsilon_{\mathrm{IACR}} \leq 4\,\epsilon_{\mathrm{SXDH}}$.

C Efficiency analysis

We summarize the efficiency of the the building blocks C, S, S' and E in Tables 1, 2 and 3, where "m-s" stands for "multi-scalar".

D Computational assumptions

Definition 43 (SXDH). *The Symmetric External Diffie-Hellman Assumption states that given (g^r, g^s, g^t) for random $r, s \in \mathbb{Z}_p$, it is hard to decide whether $t = rs$ or t is random; moreover, given $(\hat{g}^{r'}, \hat{g}^{s'}, \hat{g}^{t'})$ for random $r', s' \in \mathbb{Z}_p$, it is hard to decide whether $t' = r's'$ or t' is random.*

The Asymetric Double Hidden Strong Diffie Hellman (ADHSDH) assumption and the Asymetric Weak Flexible Computational Diffie Hellman (AWFCDH) assumption have been introduced in [AFG+10].

Table 1. Sizes of components of the commit-and-prove scheme C

$\lvert ck\rvert$	$3\lvert\mathbb{G}\rvert + 3\lvert\hat{\mathbb{G}}\rvert$
$\lvert\mathsf{Cm}(g_1)\rvert$	$2\lvert\mathbb{G}\rvert$
$\lvert\mathsf{Cm}(\hat{g})\rvert$	$2\lvert\hat{\mathbb{G}}\rvert$
$\lvert\mathsf{Cm}(1_{\mathbb{Z}_p})\rvert$	$2\lvert\hat{\mathbb{G}}\rvert$
Homogeneous pairing product equation with variables in $\mathbb{G}$	$2\lvert\hat{\mathbb{G}}\rvert$
Homogeneous pairing product equation with variables in $\hat{\mathbb{G}}$	$2\lvert\mathbb{G}\rvert$
General homogeneous pairing product equation	$4\lvert\mathbb{G}\rvert + 4\lvert\hat{\mathbb{G}}\rvert$
M-s equation in $\mathbb{G}$ with variables in $\mathbb{Z}_p$	$\lvert\mathbb{G}\rvert$
Homogeneous m-s equation in $\hat{\mathbb{G}}$ with variables in $\hat{\mathbb{G}}$	$2\lvert\mathbb{Z}_p\rvert$
General m-s equation in $\mathbb{G}$	$2\lvert\mathbb{G}\rvert + 4\lvert\hat{\mathbb{G}}\rvert$

Table 2. Characteristics of the signature schemes S and S' for message spaces $\mathcal{M}' = \hat{\mathbb{G}}$ and $\mathcal{M} = \{(g^m, \hat{g}^m) \mid m \in \mathbb{Z}_p\}^2$, resp.

Signature scheme	S [Fuc11]	S' [AGHO11]
$\lvert par_S\rvert$	$3\lvert\mathbb{G}\rvert$	0
$\lvert sk\rvert$	$\lvert\mathbb{Z}_p\rvert$	$3\lvert\mathbb{Z}_p\rvert$
$\lvert vk\rvert$	$\lvert\mathbb{G}\rvert + \lvert\hat{\mathbb{G}}\rvert$	$3\lvert\hat{\mathbb{G}}\rvert$
$\lvert\sigma\rvert$	$13\lvert\mathbb{G}\rvert + 9\lvert\hat{\mathbb{G}}\rvert$	$2\lvert\mathbb{G}\rvert + \lvert\hat{\mathbb{G}}\rvert$
Nb of pairing eqs in $S.\mathsf{Verify}$	12 general equations	1 linear in $\hat{\mathbb{G}}$, 1 general
$\lvert\pi_\sigma\rvert$	$48\lvert\mathbb{G}\rvert + 48\lvert\hat{\mathbb{G}}\rvert$	$6\lvert\mathbb{G}\rvert + 4\lvert\hat{\mathbb{G}}\rvert$

Table 3. Characteristics of the ElGamal encryption E' with message space $\mathbb{G}^2$

$\lvert sk\rvert$	$2\lvert\mathbb{Z}_p\rvert$
$\lvert pk\rvert$	$2\lvert\mathbb{G}\rvert$
$\lvert c\rvert$	$3\lvert\mathbb{G}\rvert$
$\lvert\nu\rvert$	$\lvert\mathbb{Z}_p\rvert$
Nb ms eqs in $E.\mathsf{Verify}$	2 general equations 1 linear with unkown in $\mathbb{Z}_p$
$\lvert\tilde{\pi}_{\mathrm{eq}}\rvert$	$5\lvert\mathbb{G}\rvert + 10\lvert\hat{\mathbb{G}}\rvert$

Definition 44 (q-ADHSDH). *Given* $(g, f, k, x = g^\xi, \hat{g}) \xleftarrow{\$} \mathbb{G}^4 \times \hat{\mathbb{G}}$ *and* $\hat{y} = \hat{g}^\xi$ *and* $\big(a_i = (kg^{\omega_i})^{\frac{1}{\xi+\gamma_i}}, c_i = f^{\gamma_i}, v_i = g^{\omega_i}, \hat{d}_i = \hat{g}^{\gamma_i}, \hat{w}_i = \hat{g}^{\omega_i}\big)_{i=1}^{q}$, *for* $\gamma_i, \omega_i \xleftarrow{\$} \mathbb{Z}_p$, *it is hard to output a new tuple* $(a, c, v, \hat{d}, \hat{w}) \in \mathbb{G}^3 \times \hat{\mathbb{G}}^2$ *of this form, i.e., a tuple that satisfies*

$$e(a, \hat{y}\hat{d}) = e(kv, \hat{g}) \ \wedge \ e(c, \hat{g}) = e(f, \hat{d}) \ \wedge \ e(v, \hat{g}) = e(g, \hat{w}).$$

Definition 45 (AWFCDH). *Given random generators* $(g, a = g^\alpha, \hat{g}) \xleftarrow{\$} (\mathbb{G}^*)^2 \times \hat{\mathbb{G}}$, *it is hard to output* $(g^\nu, g^{\nu\alpha}, \hat{g}^\nu, \hat{g}^{\nu\alpha})$, *i.e., a tuple* $(r, m, \hat{s}, \hat{n})$ *that satisfies:*

$$e(a, \hat{s}) = e(m, \hat{g}) \ \wedge \ e(m, \hat{g}) = e(g, \hat{n}) \ \wedge \ e(r, \hat{g}) = e(g, \hat{s}).$$