Achieving Optimal Anonymity in Transferable E-cash with a Judge[⋆]

Olivier Blazy[1], Sébastien Canard[2], Georg Fuchsbauer[3], Aline Gouget[4], Hervé Sibert[5], and Jacques Traoré[2]

[1] École Normale Supérieure – CNRS – INRIA, Paris, France
[2] Orange Labs – Applied Crypto Group, Caen, France
[3] University of Bristol – Dept. Computer Science, UK
[4] Gemalto – Security Lab, Meudon, France
[5] ST-Ericsson, Le Mans, France

Abstract. Electronic cash (e-cash) refers to money exchanged electronically. The main features of traditional cash are usually considered desirable also in the context of e-cash. One such property is *off-line transferability*, meaning the recipient of a coin in a transaction can transfer it in a later payment transaction to a third person without contacting a central authority. Among security properties, the anonymity of the payer in such transactions has been widely studied. This paper proposes the first efficient and secure transferable e-cash scheme with the strongest achievable anonymity properties, introduced by Canard and Gouget. In particular, it should not be possible for adversaries who receive a coin to decide whether they have owned that coin before. Our proposal is based on two recent cryptographic primitives: the proof system by Groth and Sahai, whose randomizability enables strong anonymity, and the commuting signatures by Fuchsbauer, which allow one to sign values that are only given as encryptions.

Keywords. Transferable e-cash, anonymity, Groth-Sahai proofs, commuting signatures.

1 Introduction

While electronic cash has long been one of the most challenging problems in cryptography, its use in practice remains rare. Indeed, despite the numerous benefits it may provide, e-cash still has many significant disadvantages. These include susceptibility to fraud, failure of technology and possible surveillance of individuals. With the recent emergence of new communication means and the availability of many applications for smart phones, the interest of the cryptographic community in electronic money has returned. Recent technologies provide the foundations for novel and desirable features such as, among others, the

[⋆] This work has been supported by the French Agence Nationale de la Recherche under the PACE 07 TCOM Project, the European Commission through the ICT Program under Contract ICT-2007-216676 ECRYPT II and EPSRC Grant EP/H043454/1. Work done while the third author was at École normale supérieure, Paris, France.

transferability of digital money. The desired security properties for e-cash are today well-known and for transferable e-cash systems anonymity is a particularly delicate issue.

Anonymity properties in transferable e-cash. The traditional properties of anonymous electronic cash are called *weak anonymity* and *strong anonymity*. The former means that it is infeasible for an attacker to identify the spender or the recipient in a transaction, and the latter states that it is infeasible for an attacker to decide whether two transactions are done by the same user or not. In [4] Canard and Gouget give a complete taxonomy of anonymity properties for transferable e-cash systems. They observe that in the transferability setting the attacker may recognize a coin that he has already observed during previous transfers. Thus, in addition to the two above traditional properties, they introduce *full anonymity* (FA), which means that an attacker is not able to recognize a coin he has already observed during a transaction between two honest users ("*observe then receive*"). They also introduce *perfect anonymity* (PA), defined as an attacker's inability to decide whether he has already owned a coin he is receiving.

Chaum and Pedersen [6] showed that a payer with unlimited computing power can always recognize his own money if he sees it later being spent; thus, the PA property cannot be achieved against unbounded adversaries. But even when his power is limited, an adversary impersonating the bank can still win the anonymity game, as shown in [4]. Perfect anonymity can therefore not be achieved by a transferable e-cash scheme. Due to this impossibility result, Canard and Gouget [4] introduce two additional anonymity notions called PA_1 and PA_2. In order to break PA_1, the adversary is given a coin and must decide whether he has already (passively) seen it in a past transaction ("*spend then observe*"). For PA_2, the bank is trusted and the adversary should not be able to decide whether or not he has already owned a coin he is receiving ("*spend then receive*"). It is shown in [4] that both properties PA_1 and PA_2 are satisfiable and that a transferable e-cash scheme should satisfy full anonymity, PA_1 and PA_2 in order to achieve"optimal" anonymity guarantees. In this paper we maintain these anonymity notions but slightly modify the used terminologies to improve readability.

Related work. Many transferable e-cash schemes have been proposed, but most of them only provide weak [10, 11] or strong anonymity [13, 6, 5, 3]. A generic construction of a transferable e-cash system with FA and PA_1 security from a one satisfying strongly anonymity is shown in [4]. PA_2 remains thus the property that is hardest to achieve.

The first proposal of a transferable e-cash scheme satisfying PA_2 is a theoretical scheme in [4] that cannot be implemented effectively. This is due to its use of complex meta-proofs [12] which allow the blinding of previous transfers of a coin, even w.r.t. a previous owner of that coin.

Subsequently, Fuchsbauer et al. [8] proposed the first practical PA_2-secure scheme. However, their scheme has the important drawbacks that (i) each user has to keep in memory the data associated to all past transactions to prove her innocence in case of a fraud and (ii) the anonymity of all subsequent owners of a double-spent coin must be revoked in order to trace the defrauder, which constitutes a serous breach of anonymity.

In conclusion, the remaining open problem is an *efficient* transferable e-cash scheme that satisfies all anonymity properties including PA_2.

Our contribution. In this paper, we propose such a scheme. More precisely, we describe a new transferable e-cash scheme based on the work on randomization of Groth-Sahai proofs [9, 2] and on the recent primitive of commuting signatures [7] based on them. This yields a new way to efficiently blind previous transfers of a coin and permits to achieve the PA_2 property, without requiring the users to store anything. We moreover believe that the use of Groth-Sahai proofs and commuting signatures in concrete cryptographic applications is technically interesting.

There is a lot of concern regarding anonymity for electronic cash with respect to illegal activities, such as money laundering or financing of terrorism. A possible compromise between user privacy and the prevention of its abuse is to provide the opportunity to appeal to a judge either in case of double-spending or in a court case. In our proposal we introduce a trusted authority called *judge*, which retrieves the identity of the defrauder after detection of a double-spending (while detection can be performed locally by the bank). Although we do not consider this explicitly, the judge could additionally trace coins and users, as required for *fair e-cash* [14]. We argue that the use of Groth-Sahai proofs—which, besides not relying on the random-oracle heuristic and being efficient, are the only randomizable proofs known to date—requires a common reference string (CRS). Therefore, instead of assuming the existence of a trusted CRS "in the sky", we entrust the judge with its setup and let him use the contained trapdoor constructively rather than "forgetting" it.

The paper is now organized as follows. In Section 2 we present the procedures constituting a transferable e-cash scheme with a judge, and we detail its security properties in Section 3. In Section 4 we give the main cryptographic tools used to instantiate our scheme, which we describe in Section 5.

2 Definitions for Transferable E-cash with Judge

In this section, we first describe the algorithms for transferable e-cash, involving a bank $\mathcal{B}$, users $\mathcal{U}$ and a judge $\mathcal{J}$. We extend the model given in [4] to include the judge authority. Moreover, in accordance with [4], the bank $\mathcal{B}$ may be divided into two entities: $\mathcal{W}$ for the withdrawal phase and $\mathcal{D}$ for the deposit phase.

2.1 Algorithms

For simplicity and contrary to [4], we represent a coin simply as a value c, while its *identifier* Id is the value that the bank retrieves during a deposit to check for double-spending. Formally, a transferable e-cash system with judge, denoted Π, is composed of the following procedures, where λ is a security parameter.

- $\mathsf{ParamGen}(1^\lambda)$ is a probabilistic algorithm that outputs the parameters of the system par. In the following, we assume that par contains λ and that it is a default input of all the other algorithms.
- $\mathsf{BKeyGen}()$, $\mathsf{JKeyGen}()$ and $\mathsf{UKeyGen}()$ are probabilistic algorithms executed respectively by $\mathcal{B}$, $\mathcal{J}$ or $\mathcal{U}$, that output a key pair. When $\mathsf{BKeyGen}()$ is executed by $\mathcal{B}$, the output is $(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B})$. The secret key $\mathsf{sk}_\mathcal{B}$ may be divided into two parts: $\mathsf{sk}_\mathcal{W}$ for the withdrawal phase and $\mathsf{sk}_\mathcal{D}$ for the deposit phase. Consequently, we define separate algorithms $\mathsf{WKeyGen}()$ and $\mathsf{DKeyGen}()$ for the bank's key generation. The output of $\mathsf{JKeyGen}()$ is a keypair $(\mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J})$ for the judge, and $\mathsf{UKeyGen}()$ outputs $(\mathsf{sk}_\mathcal{U}, \mathsf{pk}_\mathcal{U})$.
  As a convention, we assume that each secret key contains the corresponding public key.
- $\mathsf{Withdraw}(\mathcal{W}[\mathsf{sk}_\mathcal{W}, \mathsf{pk}_\mathcal{U}], \mathcal{U}[\mathsf{sk}_\mathcal{U}, \mathsf{pk}_\mathcal{B}])$ is an interactive protocol where $\mathcal{U}$ withdraws one transferable coin from $\mathcal{B}$. At the end, $\mathcal{U}$ either gets a coin c and outputs ok, or it outputs $\perp$. The output of $\mathcal{B}$ is either its view $\mathcal{V}_\mathcal{B}^{\mathsf{W}}$ of the protocol (including $pk_\mathcal{U}$), or $\perp$ in case of error.
- $\mathsf{Spend}(\mathcal{U}_1[c, \mathsf{sk}_{\mathcal{U}_1}, \mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{J}], \mathcal{U}_2[\mathsf{sk}_{\mathcal{U}_2}, \mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{J}])$ is an interactive protocol in which $\mathcal{U}_1$ spends/transfers the coin c to $\mathcal{U}_2$. At the end, $\mathcal{U}_2$ outputs either a coin c' or $\perp$, and $\mathcal{U}_1$ either tags the coin c as spent and outputs ok, or outputs $\perp$.
- $\mathsf{Deposit}(\mathcal{U}[c, \mathsf{sk}_\mathcal{U}, \mathsf{pk}_\mathcal{B}], \mathcal{D}[\mathsf{sk}_\mathcal{D}, \mathsf{pk}_\mathcal{U}, \mathcal{L}])$ is an interactive protocol where $\mathcal{U}$ deposits a coin c at the bank $\mathcal{B}$. If c is not consistent, then $\mathcal{B}$ outputs $\perp_1$. Else, $\mathcal{B}$ computes the identifier Id of the deposited coin. If $\mathcal{L}$, the list of spent coins, contains an entry (Id, c'), for some c', then $\mathcal{B}$ outputs $(\perp_2, Id, c, c')$. Else, $\mathcal{B}$ adds (Id, c) to its list $\mathcal{L}$, credits $\mathcal{U}$'s account, and returns $\mathcal{L}$. $\mathcal{U}$'s output is ok or $\perp$.
- $\mathsf{Identify}(Id, c, c', \mathsf{sk}_\mathcal{J})$ is a deterministic algorithm executed by the judge $\mathcal{J}$ that outputs a key $\mathsf{pk}_\mathcal{U}$ and a proof τ_G. If the users who had submitted c and c' are not malicious, then τ_G is a proof that $\mathsf{pk}_\mathcal{U}$ is the registered key of a user that double-spent a coin. If $Id = 0$, this signifies that the judge cannot conclude.
- $\mathsf{VerifyGuilt}(\mathsf{pk}_\mathcal{U}, \tau_G)$ is a deterministic algorithm that can be executed by anyone. It outputs 1 if τ_G is correct and 0 otherwise.

The main differences between these algorithms and those described in [4] is the additional key generation algorithm $\mathsf{JKeyGen}()$ and the modification of the procedure to identify a defrauder in case of a double-spending detection.

2.2 Global Variables and Oracles

Before formalizing the security properties, we first define the adversary's means of interaction with his challenger in the security experiments of a transferable e-cash system: we introduce global variables (in accordance with to [4]) and oracles[1].

Global variables. The set of public (resp. secret) user keys is denoted by $\mathcal{PK} = \{(i, \mathsf{pk}_i) : i \in \mathbb{N}\}$ (resp. $\mathcal{SK} = \{(i, \mathsf{sk}_i) : i \in \mathbb{N}\}$ with $sk_i = \perp$ if user i is corrupted). The set of views by the bank of the withdrawals done by the adversary is denoted by $\mathcal{SC}$ (for supplied coins) and the set of all coins owned by the oracles is denoted by $\mathcal{OC}$ (for obtained coins). The set of deposited electronic cash (corresponding to $\mathcal{L}$) is denoted by $\mathcal{DC}$ (for deposited coins). In addition, we define the set of users who have received a coin from the adversary, denoted by $\mathcal{RU}$; and the set of users who have spent a coin to the adversary, denoted by $\mathcal{SU}$. These modifications should improve the understanding of the original description of oracles provided in [4].

Creation and corruption of users. The oracle $\mathsf{Create}(i)$ executes $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{UKeyGen}()$, defines $\mathcal{PK}[i] = \mathsf{pk}_i$ and $\mathcal{SK}[i] = \mathsf{sk}_i$, and outputs pk_i. The oracle $\mathsf{Corrupt}(i, \mathsf{pk}_i)$ defines $\mathcal{PK}[i] = \mathsf{pk}_i$ and $\mathcal{SK}[i] = \perp$, and outputs ok. If the adversary calls $\mathsf{Corrupt}(i, \perp)$ then the oracle outputs $\mathcal{SK}[i]$ and then sets $\mathcal{SK}[i] = \perp$. In all cases, the coins belonging to user i, stored in $\mathcal{OC}$, are also given to $\mathcal{A}$.

Withdrawal protocol. We define three oracles relating to withdrawal.

- The oracle $\mathsf{BWith}()$ plays the bank side of a $\mathsf{Withdraw}$ protocol. It updates $\mathcal{SC}$ by adding $\mathcal{V}_{\mathcal{B}}^{\mathsf{W}}$ with bit 1 to flag it as a corrupted coin.
- The oracle $\mathsf{UWith}(i)$ plays the user i in a $\mathsf{Withdraw}$ protocol. It updates $\mathcal{OC}$ by adding the value (i, j, c) with flag 1, where j is the first empty entry of $\mathcal{OC}$ (independently of the user i to which it belongs).
- The oracle $\mathsf{With}(i)$ simulates a complete $\mathsf{Withdraw}$ protocol, playing the role of both $\mathcal{B}$ and user i, updates $\mathcal{OC}$ as for $\mathsf{UWith}(i)$ and updates $\mathcal{SC}$ by adding $\mathcal{V}_{\mathcal{B}}^{\mathsf{W}}$ both with flag 0. It outputs the communications between $\mathcal{B}$ and $\mathcal{U}$.

Spending protocol. Here we take into account that during a Spend protocol the adversary can play the role of the payer, the receiver, or can only be a passive observer. This will be relevant for the anonymity experiments in Section 3.4.

- The oracle $\mathsf{Rcv}(i)$ allows $\mathcal{A}$ to spend a coin to user i. The oracle plays the role of $\mathcal{U}_2$ with the secret key of user i in the Spend protocol. It updates the set $\mathcal{OC}$ by adding a new entry (i, j, c) and adds i to the set $\mathcal{RU}$.
- The oracle $\mathsf{Spd}(i, j)$ enables $\mathcal{A}$ to receive either the coin j or a coin transferred from user i. Either i or j can be undetermined (equal to $\perp$). The owner i of the spent coin j is then added to $\mathcal{SU}$. The oracle plays the role of user $\mathcal{U}_1$ in the Spend protocol with the secret key of the owner i of the coin j in $\mathcal{OC}$. It uses the entry (i, j, c) of $\mathcal{OC}$ as the Spend protocol describes it. It finally updates this entry by changing the flag to 1.

[1] By convention, the name of an oracle corresponds to the action done by this oracle.

– The oracle $\mathsf{S\&R}$ (spend-and-receive) permits $\mathcal{A}$ to observe the spending of a coin j between users i_1 (in the role of $\mathcal{U}_1$) and i_2 (in the role of $\mathcal{U}_2$), who are both played by the oracle. It updates $\mathcal{OC}$ by adding (i_2, j', c) and by flagging the coin j as spent by i_1. It outputs all the (external) communications of the spending.

Deposit protocol. Depending on who the adversary impersonates there are several oracles for deposit.[2]

– The oracle $\mathsf{BDepo}()$ plays the role of the bank during a $\mathsf{Deposit}$ protocol and interacts with the adversary. The oracle gives the output of a $\mathsf{Deposit}$ procedure and updates the set $\mathcal{DC}$.
– The oracle $\mathsf{UDepo}(i, c)$ plays the role of the user i during a $\mathsf{Deposit}$ protocol for the coin c. The adversary is in this case the bank. If $c = \perp$ then the oracle randomly chooses one coin belonging to user i and deposits it.
– The oracle $\mathsf{Depo}(i, c)$ plays the role of both the bank and the user i in the $\mathsf{Deposit}$ protocol of the coin c. If $c = \perp$, then the oracle randomly chooses the coin to be deposited.
– The oracle $\mathsf{Idt}(Id, c, c')$ plays the role of the judge in the $\mathsf{Identify}$ procedure, with the same outputs.

A consequence of the result by Chaum and Pedersen [6], who showed that a transferred coin necessarily grows in size, is that an adversary may easily break anonymity by checking the number of times a given coin has been transferred. In the following, we say that two users i_0 and i_1 are *compatible*, and write $\mathsf{comp}(i_0, i_1) = 1$, if they both own at least one coin with the same size.

3 Security Properties

In this section, we define the security notions for an e-cash system with a judge, adapting those from Canard and Gouget [4]. In every security game the challenger first generates the parameters and the keys for the bank and the judge; we denote this by AllGen. The challenger then gives the adversary the keys corresponding to the parties he is allowed to impersonate.

3.1 Unforgeability

Unforgeability is a notion protecting the bank, meaning that no collection of users can ever spend more coins than they withdrew, even by corrupting the judge. Formally, we have the following definition based on the experiment given below.

[2] The main difference between these oracles and those described in [4] is the modification of the oracle $\mathsf{BDepo}()$ and the definition of the new oracle $\mathsf{Idt}(Id, c, c')$. In [4] there is a single oracle $\mathsf{CreditAccount}()$, which executes both $\mathsf{BDepo}()$ and $\mathsf{Ident}(Id, c, c')$. This modification is necessitated by the inclusion of the judge.

Definition 1 (Unforgeability). *Let Π be a transferable e-cash system with a judge. For an adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, we let $\boldsymbol{Succ}^{\mathsf{unfor}}_{\Pi,\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{\mathsf{unfor}}_{\Pi,\mathcal{A}}(\lambda) = 1]$. Π is said to be* unforgeable *if the function $\boldsymbol{Succ}^{\mathsf{unfor}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.*

```
$\underline{\mathbf{Exp}^{\mathsf{unfor}}_{\Pi,\mathcal{A}}(\lambda)}$
  – $(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J}) \leftarrow \mathsf{AllGen}(1^\lambda)$; $cont \leftarrow \mathsf{true}$; $st \leftarrow \emptyset$;
  – While $(cont = \mathsf{true})$ do {
      – $(cont, st)$
            $\leftarrow \mathcal{A}^{\mathsf{Create,Corrupt,BWith,With,Rcv,Spd,S\&R,BDepo,Depo}}(st, \mathsf{par}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{B})$;
        Let $q_W$ be the number of successful calls to BWith and With;
        let $q_D$ denote the number of successful calls to BDepo and Depo;
      – If $q_W < q_D$ then return 1; }
  – Return $\perp$.
```

3.2 Identification of Double-Spenders

This notion guarantees the bank that no collection of users, collaborating with the judge, can spend a coin twice (double-spend) without revealing one of their identities. Formally, we have the following experiment and definition.

```
$\underline{\mathbf{Exp}^{\mathsf{ident}}_{\Pi,\mathcal{A}}(\lambda)}$
  – $(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J}) \leftarrow \mathsf{AllGen}(1^\lambda)$; $cont \leftarrow \mathsf{true}$; $st \leftarrow \emptyset$;
  – While $(cont = \mathsf{true})$ do {
      – $st \leftarrow \mathcal{A}^{\mathsf{Create,Corrupt,BWith,With,Rcv,Spd,S\&R,BDepo,Depo,Idt}}(st, \mathsf{par}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{B})$;
      – If a call to BDepo outputs $(\perp_2, Id, c, c')$ then $cont \leftarrow \mathsf{false}$; }
  – $(i^*, \tau_G) \leftarrow \mathsf{Identify}(Id, c, c', \mathsf{sk}_\mathcal{J})$;
  – If $\mathsf{VerifyGuilt}(\mathsf{pk}_{i^*}, \tau_G) = 0$ or $i^* = 0$ then return 1;
  – Return $\perp$.
```

Definition 2 (Double-Spender Identification). *Let Π be a transferable e-cash system with a judge. For any adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, we let $\boldsymbol{Succ}^{\mathsf{ident}}_{\Pi,\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{\mathsf{ident}}_{\Pi,\mathcal{A}}(\lambda) = 1]$. Π* identifies double spenders *if the function $\boldsymbol{Succ}^{\mathsf{ident}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.*

3.3 Exculpability

This notion protects honest users in that the bank, even when colluding with a collection of malicious users and possibly the judge, cannot falsely accuse (with a proof) honest users of having double-spent a coin. Formally, we have the following experiment and definition.

```
$\underline{\mathbf{Exp}^{\mathsf{excul}}_{\Pi,\mathcal{A}}(\lambda)}$
  – $(\mathsf{par}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{J}}, \mathsf{pk}_{\mathcal{J}}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
  – $(Id^*, c_1^*, c_2^*, i^*, \tau^*)$
          $\leftarrow \mathcal{A}^{\mathsf{Create},\mathsf{Corrupt},\mathsf{UWith},\mathsf{Rcv},\mathsf{Spd},\mathsf{S\&R},\mathsf{UDepo},\mathsf{Idt}}(st, \mathsf{par}, \mathsf{sk}_{\mathcal{J}}, \mathsf{sk}_{\mathcal{B}})$;
  – If $\mathsf{VerifyGuilt}(\mathsf{pk}_{i^*}, \tau^*) = 1$ and $\mathsf{sk}_{i^*} \neq \perp$, return 1;
  – Return $\perp$.
```

Definition 3 (Exculpability). *Let Π be a transferable e-cash system with judge. For an adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, we let $\boldsymbol{Succ}^{\mathsf{excul}}_{\Pi,\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{\mathsf{excul}}_{\Pi,\mathcal{A}}(\lambda) = 1]$. Π is said to be* exculpable *if the function $\boldsymbol{Succ}^{\mathsf{excul}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.*

3.4 Anonymity Properties in Transferable E-cash

Regarding anonymity, Canard and Gouget [4] distinguish between five different notions: weak anonymity (WA), strong anonymity (SA), full anonymity (FA), and two types of restricted perfect anonymity (PA_1 and PA_2). They show that FA implies SA, which implies WA, and that FA, PA_1 and PA_2 are all incomparable. We say the anonymity for a transferable e-cash scheme is *optimal* when it satisfies the latter 3 properties. We work with the formal definitions of [4] but slightly modify the terminology[6].

- *Observe-then-Receive Full Anonymity* (OtR-FA, previously FA): the adversary, impersonating the bank, cannot link a coin he receives as "legitimate" user to a previously (passively) observed transfer between honest users.
- *Spend-then-Observe Full Anonymity* (StO-FA, previously PA_1): the adversary, impersonating the bank, cannot link a (passively) observed coin transferred between two honest users to a coin he has already owned as a "legitimate" user.
- *Spend-then-Receive Full Anonymity* (StR-FA, previously PA_2): when the bank is honest, the adversary cannot link two transactions involving the same coin, i.e. make the link between two coins he has received.

In the following, we say that a transferable e-cash scheme achieves *optimal anonymity* if it satisfies at the same time OtR-FA, StO-FA and StR-FA, which are incomparable, according to [4]. These anonymity notions are formally defined below, based on the corresponding experiments given in Figure 1.

Definition 4 (Anonymity Properties). *Let Π be a transferable e-cash system with judge and let $\mathsf{c} \in \{\mathsf{otr\text{-}fa}, \mathsf{sto\text{-}fa}, \mathsf{str\text{-}fa}\}$. For an adversary $\mathcal{A}$ and $\lambda \in \mathbb{N}$, we let $\boldsymbol{Adv}^{\mathsf{c}}_{\Pi,\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{\mathsf{c\text{-}1}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \Pr[\boldsymbol{Exp}^{\mathsf{c\text{-}0}}_{\Pi,\mathcal{A}}(\lambda) = 1]$. Π is said to be* Observe-then-Receive fully anonymous *(resp.* Spend-then-Observe fully anonymous*,* Spend-then-Receive fully anonymous*) if the function $\boldsymbol{Adv}^{\mathsf{otr\text{-}fa}}_{\Pi,\mathcal{A}}(\cdot)$ (resp. $\boldsymbol{Adv}^{\mathsf{sto\text{-}fa}}_{\Pi,\mathcal{A}}(\cdot)$, $\boldsymbol{Adv}^{\mathsf{str\text{-}fa}}_{\Pi,\mathcal{A}}(\cdot)$) is negligible for any polynomial-time adversary $\mathcal{A}$.*

[6] In particular, the notion of "perfect" anonymity in [4] is not based on the indistinguishability of distributions, which may be confusing as we only achieve computational security.

$\mathbf{Exp}_{\Pi,\mathcal{A}}^{\textsf{otr-fa-}b}(\lambda)$ // $b \in \{0,1\}$, $\mathcal{A} = (\mathcal{A}_{ch}, \mathcal{A}_c, \mathcal{A}_{gu})$

- $(\mathsf{par}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{J}}, \mathsf{pk}_{\mathcal{J}}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
- $(i_0^*, i_1^*, st) \leftarrow \mathcal{A}_{ch}^{\mathsf{Create,Corrupt,UWith,Rcv,Spd,S\&R,UDepo,Idt}}(\mathsf{par}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{J}})$;
- If $\mathsf{sk}_{i_0^*} = \perp \vee \mathsf{sk}_{i_1^*} = \perp \vee \mathsf{comp}(i_0^*, i_1^*) = 0 \vee i_0^* \in \mathcal{RU} \vee i_1^* \in \mathcal{RU}$ then return $\perp$;
- Choose j^* such that coin number j^* belongs to i_b^* and i_{1-b}^* owns a coin of equal size. Simulate $\mathsf{Spd}(i_b^*, j^*)$ to $\mathcal{A}_c$, which outputs st_c;
- $b^* \leftarrow \mathcal{A}_{gu}^{\mathsf{Create,Corrupt,UWith,Rcv,Spd,S\&R,UDepo,Idt}}(st_c)$;
- Return b^*.

$\mathbf{Exp}_{\Pi,\mathcal{A}}^{\textsf{sto-fa-}b}(\lambda)$ // $b \in \{0,1\}$, $\mathcal{A} = (\mathcal{A}_{ch}, \mathcal{A}_{gu})$

- $(\mathsf{par}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{J}}, \mathsf{pk}_{\mathcal{J}}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
- $(i_0^*, i_1^*, i_2^*, st) \leftarrow \mathcal{A}_{ch}^{\mathsf{Create,Corrupt,UWith,Rcv,Spd,S\&R,UDepo,Idt}}(\mathsf{par}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{J}})$;
- If $\mathsf{sk}_{i_0^*} = \perp \vee \mathsf{sk}_{i_1^*} = \perp \vee \mathsf{sk}_{i_2^*} = \perp \vee \mathsf{comp}(i_0^*, i_1^*) = 0$, return $\perp$;
- Choose j^* such that coin number j^* belongs to i_b^*, and i_{1-b}^* owns a coin of equal size; run $out \leftarrow \mathsf{S\&R}(j^*, i_b^*, i_2^*)$;
- $b^* \leftarrow \mathcal{A}_{gu}^{\mathsf{Create,Corrupt,UWith,Rcv,Spd,S\&R,UDepo,Idt}}(out, st_c)$;
- If an oracle call involved the coin used in $\mathsf{S\&R}$ then return $\perp$;
- Return b^*.

$\mathbf{Exp}_{\Pi,\mathcal{A}}^{\textsf{str-fa-}b}(\lambda)$ // $b \in \{0,1\}$, $\mathcal{A} = (\mathcal{A}_{ch}, \mathcal{A}_c, \mathcal{A}_{gu})$

- $(\mathsf{par}, \mathsf{sk}_{\mathcal{B}} = (\mathsf{sk}_{\mathcal{W}}, \mathsf{sk}_{\mathcal{D}}), \mathsf{pk}_{\mathcal{B}} = (\mathsf{pk}_{\mathcal{W}}, \mathsf{pk}_{\mathcal{D}}), \mathsf{sk}_{\mathcal{J}}, \mathsf{pk}_{\mathcal{J}}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
- (i_0^*, i_1^*, st)
  $\leftarrow \mathcal{A}_{ch}^{\mathsf{Create,Corrupt,UWith,Rcv,Spd,S\&R,Depo,Idt}}(\mathsf{par}, \mathsf{sk}_{\mathcal{W}}, \mathsf{pk}_{\mathcal{D}}, \mathsf{pk}_{\mathcal{J}})$;
- If $\mathsf{sk}_{i_0^*} = \perp \vee \mathsf{sk}_{i_1^*} = \perp \vee \mathsf{comp}(i_0^*, i_1^*) = 0$ then return $\perp$;
- Choose j^* such that coin number j^* belongs to i_b^*, and i_{1-b}^* owns a coin of equal size. Simulate $\mathsf{Spd}(i_b^*, j^*)$ to $\mathcal{A}_c$, which outputs st_c;
- $b^* \leftarrow \mathcal{A}_{gu}^{\mathsf{Create,Corrupt,UWith,Rcv,Spd,S\&R,Depo,Idt}}(st_c)$;
- If the oracle Depo is called on input either i_0^* or i_1^*, return $\perp$;
- Return b^*.

Fig. 1. Experiments for full-anonymity notions.

4 Cryptographic Tools

In this section we give the main tools we need to construct our new transferable e-cash system with judge. For each of them, we introduce the concept, give the underlying procedures and formally describe the main security characteristics.

4.1 Assumptions

Our construction will rely on two cryptographic assumptions: the *symmetric external Diffie-Hellman (DH) assumption* and the *asymmetric double hidden strong DH assumption*, a "q-type" assumption introduced in [1].

Definition 5 ((SXDH)). *Let $\mathbb{G}_1, \mathbb{G}_2$ be cyclic groups of prime order generated by g_1 and g_2, respectively, and let $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map. The SXDH assumption states that for $i = 1, 2$, given g_i, g_i^a, g_i^b, for random a, b, it is hard to distinguish g_i^{ab} from a random element from $\mathbb{G}_i$.*

Definition 6 (q-ADHSDH). *Given $(g, f, k, g^\xi, h, h^\xi) \in \mathbb{G}_1^4 \times \mathbb{G}_2^2$ and*

$$\left(a_i = (k \cdot g^{\nu_i})^{\frac{1}{\xi + \gamma_i}}, b_i = f^{\gamma_i}, v_i = g^{\nu_i}, d_i = h^{\gamma_i}, w_i = h^{\nu_i}\right)_{i=1}^{q-1}$$

for random $g, f, k \leftarrow \mathbb{G}_1$, $h \leftarrow \mathbb{G}_2$, $\xi, \gamma_i, \nu_i \leftarrow \mathbb{Z}_p$, it is hard to output a new such tuple $(a, b, v, d, w) \in \mathbb{G}_1^3 \times \mathbb{G}_2^2$, i.e., one that satisfies

$$e(a, h^\xi \cdot d) = e(k \cdot v, h) \qquad e(b, h) = e(f, d) \qquad e(v, h) = e(g, w)$$

4.2 Groth-Sahai Proofs

Groth and Sahai [9] proposed the first efficient non-interactive proof system for a large class of statements over bilinear groups in the standard model. Those proofs fit our purpose perfectly: their witness indistinguishability guarantees the anonymity of the users that withdraw, transfer and spend coins, and their randomizability provides unlinkability of transferred coins.

We use SXDH-based Groth-Sahai commitments and proofs in a pairing-friendly setting in order to commit to elements and prove relations satisfied by the associated plaintexts. The commitment key is: $\mathbf{u} \in \mathbb{G}_1^{2\times 2}$, $\mathbf{v} =\in \mathbb{G}_2^{2\times 2}$. Depending on whether the commitments should be perfectly binding or perfectly hiding (for simulations in security proofs), the initialization of the parameters will vary between: $\boldsymbol{u}_1 = (g_1, u)$ with $u = g_1^\mu$ and $\boldsymbol{u}_2 = \boldsymbol{u}_1^\nu$ with $\mu, \nu \xleftarrow{\$} \mathbb{Z}_p^*$ (which makes $\mathbf{u}$ a Diffie-Hellman tuple in $\mathbb{G}_1$) for the binding setting, and for the hiding setting $\boldsymbol{u}_2 = \boldsymbol{u}_1^\nu \odot (1, g_1)^{-1} = (g_1^\nu, g_1^{\mu\nu-1})$. Similarly, we define key pairs $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ in $\mathbb{G}_2^2$ with independent randomness.

Commitments to group elements. To commit to $X \in \mathbb{G}_1$ with random values $s_1, s_2 \in \mathbb{Z}_p$, we set $\mathcal{C}(X) = (1, X) \odot \mathbf{u}_1^{s_1} \odot \mathbf{u}_2^{s_2} = (u_{1,1}^{s_1} \cdot u_{2,1}^{s_2}, X \cdot u_{1,2}^{s_1} \cdot u_{2,2}^{s_2})$.

- Perfectly binding setting: We have $\mathcal{C}(X) = (g_1^a, X \cdot u^a)$, with $a = s_1 + \nu s_2$. A simulator that knows μ can extract X as this is an ElGamal encryption of X under (g_1, g_1^μ). The key μ is called the extraction key for such an *extractable commitment.*
- Perfectly hiding setting: We have $\mathcal{C}(X) = (g_1^a, X \cdot g_1^b \cdot u^a)$, for $a = s_1 + \nu s_2$ and $b = -s_2$, two independent random values. $\mathcal{C}(X)$ is thus an encryption of $X \cdot g_1^b$, for a random b, so it blinds X.

Analogously, one commits to elements from $\mathbb{G}_2$ by replacing $\mathbf{u}$ by $\mathbf{v}$ and g_1 by g_2 in the above.

Proofs. Under the SXDH assumption, the two initializations of the commitment key are indistinguishable. Groth and Sahai [9] show how to construct proofs that a set of committed values satisfies an equation of a certain type. A proof is in $\mathbb{G}_2^{2\times 2} \times \mathbb{G}_1^{2\times 2}$; it can be constructed using the committed values satisfying the equation and their randomness, and it is verified w.r.t. the commitments and the commitment key. If the key is set up as perfectly hiding then the proof does not reveal more than the fact that the values satisfy the equation.

Randomization. The commitments can easily be randomized. Given, e.g., a commitment $c \in \mathbb{G}_1^2$, one chooses two random values s_1', s_2' and computes the randomization as $c' = (c_1 \cdot u_{1,1}^{s_1'} \cdot u_{2,1}^{s_2'}, c_2 \cdot u_{1,2}^{s_1'} \cdot u_{2,2}^{s_2'})$. In [2] it is shown how to randomize and adapt a proof (π, θ) for a vector of commitments $(c_i)_i$ to their randomizations $(c_i')_i$.

4.3 Commuting Signatures

Commuting signatures and verifiable encryption [7] is a primitive combining a signature scheme (the automorphic signature from [1], whose messages are group elements) with Groth-Sahai (GS) proofs. This allows one to commit to a message, a verification key, or a corresponding signature (or arbitrary combinations of them), and prove that the committed values are valid (i.e. the signature is valid on the message under the key), via the GS methodology.

Commuting signatures provide several additional functionalities, of which we use the following two.

SigCom**:** This allows a signer, who is given a commitment $\mathbf{C}$ to a message, to make a commitment $\mathbf{c}_\Sigma$ to a signature (under his secret key) on that message (without knowing it though) and a proof that $\mathbf{c}_\Sigma$ contains a valid signature on the value committed in $\mathbf{C}$.

$\mathsf{AdC}_\mathcal{K}$**:** Given a commitment to a message, a commitment to a signature and a proof of validity w.r.t. to a verification key, this algorithm allows anyone to commit to that key and adapt the proof; more precisely, $\mathsf{AdC}_\mathcal{K}$ outputs a proof asserting that a commitment contains a valid signature on a committed message under a committed verification key.

Security states that the output of SigCom is the same as if the signer had known the message, signed it, made commitments to the message and the signature and had given a GS proof of validity w.r.t. his signature-verification key. Analogously, the output of $\mathsf{AdC}_\mathcal{K}$ is the same as a proof constructed for the known committed values.

In our e-cash scheme commuting signatures will enable users to produce signatures on values that are only available as commitments and make a proof of validity under their verification key, which is also given as a commitment.

5 A Tranferable E-cash System with Judge

Based on the cryptographic building blocks introduced in the last section, we are now in a position to describe our transferable e-cash system with judge. In our solution the withdrawer is anonymous towards the bank, a feature that previous schemes do not offer. This is motivated by the fact that our withdrawal is very similar to the spending protocol and it is easy to make the withdrawer non-anonymous, should one wish to. A possible application of our scheme is the anonymous purchase of tickets which can then be transferred to other users. Another scenario could be e-cash which can be purchased in exchange for actual cash.

5.1 Overview of our Solution

A coin is represented by a unique chain of nonces $n = n_0 \| n_1 \| n_2 \| \cdots$, where each n_i is randomly chosen by a consecutive owner of the coin. Thus, n_0 is chosen by the bank, n_1 by the withdrawer, n_2 by the one who receives this coin from the withdrawer, and so on.

A double-spending has occurred when two coins n and n' are deposited which both begin with $n_0 = n'_0$. Note that the minimum value i such that $n_i \neq n'_i$ corresponds to the transfer of the coin where it was double-spent. If we oblige every user to commit to her identity during a transfer and include this commitment in the coin then a judge holding an extraction key can trace the defrauder.

When a coin is transferred from $\mathcal{U}_i$ to $\mathcal{U}_{i+1}$, the spender $\mathcal{U}_i$ signs the following: (i) the nonce she chose when receiving the coin (during a spending or withdrawal), (ii) the nonce chosen by the receiver $\mathcal{U}_{i+1}$, and (iii) $\mathcal{U}_{i+1}$'s verification key. The latter binds this transfer to the next one, where $\mathcal{U}_{i+1}$ will use her signing key. In fact, since only $\mathcal{U}_{i+1}$ knows the secret key corresponding to the signed public key, she is the only one able to spend the coin.

However, to remain anonymous, $\mathcal{U}_{i+1}$ cannot let the spender know her verification key. This is where commuting signatures come into play: they allow the signer to make (a commitment to) a signature on the receiver's key, even when it is only given to the signer as a commitment. Since SigCom additionally outputs a proof, validity of the committed signature is publicly verifiable.

When a coin is spent, its entire history (i.e. committed nonces, keys, signatures and proofs of their validity from previous transfers) is transmitted. This will guarantee unforgeability, identification of double-spending and non-frameability, without requiring data to be stored by the user and provided later on demand to prove innocence (as was the case in the relaxed model in [8]). Every time a coin is transferred, its history (consisting of commitments and Groth-Sahai proofs) can be completely randomized. Thus, a previous owner of a coin cannot recognize it at a later moment; this is how our scheme achieves strong anonymity notions.

5.2 Key-Generation Algorithms

During the generation phase, the judge $\mathcal{J}$ generates two pairs of commitment/extraction keys, which will enable identification of double spenders. Similarly, the double-spending detector $\mathcal{D}$ also generates such a key pair.

We denote a commitment under $\mathcal{J}$'s keys by either c (first key) or $\tilde{c}$ (second key) and a commitment under $\mathcal{D}$'s key by d. Using their secret extraction keys, the judge and the detector can open commitments under their respective keys using $\mathsf{Open}_{\mathcal{J}}$ and $\mathsf{Open}_{\mathcal{D}}$.

The judge also generates a key pair for a commuting signature scheme; in the following, a signature on m from $\mathcal{J}$ is denoted $\mathsf{Sign}_{\mathcal{J}}(m)$. Moreover, the bank $\mathcal{B}$ and each user $\mathcal{U}$ generate key pairs $(\mathsf{bsk}, \mathsf{bpk})$ and $(\mathsf{usk}, \mathsf{upk})$ for the commuting signature scheme. When registering, a user $\mathcal{U}$ obtains from the judge $\mathcal{J}$ a signature on her verification key as membership certificate: $\mathsf{cert} = \mathsf{Sign}_{\mathcal{J}}(\mathsf{upk})$. In the following, we differentiate the users by different indices: $\mathcal{U}_1$, $\mathcal{U}_2$, etc.

5.3 Withdrawal Protocol

The withdrawal protocol involves a user $\mathcal{U}_1$ and the bank $\mathcal{B}$. In a nutshell, the bank $\mathcal{B}$ generates a random nonce n_0 and the user a random nonce n_1, which together will be the beginning of the serial number of the coin. The bank then signs these nonces and the user's public key upk_1, which will bind the user's identity to the coin and enable tracing in case of double spending.

However, to guarantee anonymity, rather than sending these values in the clear, the user sends *commitments* to them. She also adds a commitment to her certificate and a proof of validity, which convinces the bank that she is registered. This can be done since the certificate is an *automorphic* signature [1], for which GS proofs can be used to prove that a committed value is a signature on another committed value, in this case upk_1, valid under the judge's verification key.

The bank now has to construct a committed signature on the values n_0, n_1 and upk_1, which are only given in the form of commitments. This is where we take advantage of the functionality SigCom of the commuting-signature scheme introduced in Section 4.3: given commitments, a signer can produce a commitment to a signature on the values contained in them, together with a proof of validity of the signature.

All these commitments will be done w.r.t. the judge's commitment key. To enable the double-spending detector $\mathcal{D}$ to *detect* a double-spending (however without breaking the user's anonymity), we do the following: in addition to committing to the nonces w.r.t. the judge's key, the user and the bank make another commitment d_{n_i} to n_i under $\mathcal{D}$'s key. In order to show that this was done correctly, we require a proof that two commitments w.r.t. different keys contain the same value. This can be done by using two instances of Groth-Sahai proofs on top of each other, as was done in [8]. The outer layer is the one corresponding to c, which will enable us to simulate such a proof when the commitment key for c is set up as hiding, but the key for d is still binding.

We formalize the above in the following protocol:

$\underline{\mathcal{U}_1}$ picks at random a nonce n_1 and makes two extractable commitments (for $\mathcal{J}$ and $\mathcal{D}$) to n_1 denoted respectively by c_{n_1} and d_{n_1}, and a proof π_{n_1} that the two committed values are equal.

Moreover, $\mathcal{U}_1$ makes commitments c_{u_1}, $\tilde{c}_{u_1}$ and c_{c_1} to its public key upk_1 and its certificate cert_1, respectively, together with a proof π_{c_1} that the value in c_{c_1} is a valid signature on the value in c_{u_1}, i.e. $\mathsf{cert}_1 = \mathsf{Sign}_{\mathcal{J}}(\mathsf{upk}_1)$, and a proof $\tilde{\pi}_{u_1}$ that the committed values on c_{u_1} and $\tilde{c}_{u_1}$ are equal.

$\mathcal{U}_1$ sends the following values to the bank: $(c_{n_1}, c_{u_1}, c_{c_1}, \pi_{c_1})$.

$\underline{\mathcal{B}}$ after verifying π_{c_1} now also generates a random nonce n_0 and makes two commitments (for $\mathcal{J}$ and $\mathcal{D}$) to n_0 denoted by c_{n_0} and d_{n_0}, and a proof π_{n_0} that the two committed values are equal.

$\mathcal{B}$ produces a committed signature c_{s_1} on the values n_0, n_1 and upk_1 by running SigCom on c_{n_0}, c_{n_1} and c_{u_1}; this also outputs a proof π_{s_1} of validity of c_{s_1} w.r.t. c_{n_0}, c_{n_1} and c_{u_1} and the bank's verification key (which is public). The bank sends all these values, from which the user forms the coin

$$\mathsf{coin}_1 = (c_{n_0}, d_{n_0}, \pi_{n_0},\; c_{n_1}, d_{n_1}, \pi_{n_1},\; c_{u_1}, \tilde{c}_{u_1}, \tilde{\pi}_{u_1},\; c_{c_1}, \pi_{c_1},\; c_{s_1}, \pi_{s_1}) \ . \quad (1)$$

In the sequel, this coin will be randomized before being spent. The result of randomizing coin_1 is denoted $\mathsf{coin}_1^{(1)}$ and consists of randomizing all its components, i.e. commitments and proofs, as described in [2]. After randomization, we have thus $\mathsf{coin}_1^{(1)} = (c_{n_0}^{(1)}, d_{n_0}^{(1)}, \pi_{n_0}^{(1)}, c_{n_1}^{(1)}, d_{n_1}^{(1)}, \pi_{n_1}^{(1)}, c_{u_1}^{(1)}, \tilde{c}_{u_1}^{(1)}, \tilde{\pi}_{u_1}^{(1)}, c_{c_1}^{(1)}, \pi_{c_1}^{(1)}, c_{s_1}^{(1)}, \pi_{s_1}^{(1)})$.

5.4 Spending Protocol

This is a protocol between a user $\mathcal{U}_1$ holding a coin as in (1) and a user $\mathcal{U}_2$ playing the role of the receiver. The protocol is very similar to the withdrawal protocol, except for two points. First, $\mathcal{U}_1$ has to randomize the coin, which prevents a later linking of the coin. Note that, due to the contained proofs, the validity of a coin is publicly verifiable.

Second, while the bank's verification key is public, $\mathcal{U}_1$'s key must remain hidden. Thus, after $\mathcal{U}_1$ produces a commitment to a signature on the values n_1, n_2 (the nonce chosen by $\mathcal{U}_2$), and $\mathcal{U}_2$'s public key upk_2, and a proof that verifies w.r.t. her public key upk_1, $\mathcal{U}_1$ does the following: using the functionality $\mathsf{AdC}_{\mathcal{K}}$ (described in Section 4.3), she converts the proof into one asserting that the committed signature is valid under the value committed in $c_{u_1}^{(1)}$ (i.e. the randomization of the commitment to upk_1).

$\underline{\mathcal{U}_2}$ picks at random a nonce n_2 and commits to it as c_{n_2} and d_{n_2} (for $\mathcal{J}$ and $\mathcal{D}$) and makes a proof π_{n_2} of equality of the committed values.

$\mathcal{U}_2$ makes further commitments c_{u_2}, $\tilde{c}_{u_2}$ and c_{c_2} to her public key upk_2 and her certificate cert_2, together with a proof π_{c_2} that c_{c_2} contains a valid certificate and a proof $\tilde{\pi}_{u_2}$ that the committed values in c_{u_2} and $\tilde{c}_{u_2}$ are equal. She sends $(c_{n_2}, c_{u_2}, c_{c_2}, \pi_{c_2})$ to $\mathcal{U}_1$

$\underline{\mathcal{U}_1}$ checks the proof sent by $\mathcal{U}_2$ and randomizes coin_1 to $\mathsf{coin}_1^{(1)}$. $\mathcal{U}_1$ then produces a committed signature on the values committed in $c_{n_1}^{(1)}, c_{n_2}$ and c_{u_2} using

SigCom: this generates a commitment c_{s_2} to a signature on the values n_1, n_2 and upk_2, as well as a proof π'_{s_2} of validity of c_{s_2} on c_{n_1}, c_{n_2}, c_{u_2} w.r.t. upk_1. Running $\mathsf{AdC}_{\mathcal{K}}$, $\mathcal{U}_1$ converts π'_{s_2} to a proof π_{s_2} asserting validity w.r.t. the key committed in $c_{u_2}^{(1)}$. Note that this works since c_{u_1} was produced and randomized to $c_{u_1}^{(1)}$ by $\mathcal{U}_1$, who therefore knows its randomness. Finally, $\mathcal{U}_1$ sends $\mathcal{U}_2$ the following: $(\mathsf{coin}_1^{(1)}, c_{s_2}, \pi_{s_2})$.

$\mathcal{U}_2$ checks the proofs contained in $\mathsf{coin}_1^{(1)}$ and π_{s_2} and defines the transferred coin as

$$\mathsf{coin}_2 := (\mathsf{coin}_1^{(1)},\ c_{n_2}, d_{n_2}, \pi_{n_2},\ c_{u_2}, \tilde{c}_{u_2}, \tilde{\pi}_{u_2},\ c_{c_2}, \pi_{c_2},\ c_{s_2}, \pi_{s_2}) \ .$$

5.5 Deposit and Identify Procedures

To deposit a coin, a user spends it to the bank, that is, she runs the protocol from the last section with the bank playing the role of $\mathcal{U}_2$. In order to detect a double-spending given a coin, the detector $\mathcal{D}$ opens all the commitments $d_{n_0}^{(\ell)}, d_{n_1}^{(\ell)}, d_{n_2}^{(\ell-1)}, \cdots, d_{n_\ell}^{(1)}$ contained in it, using her extraction key. She thus obtains the serial number $n = n_0 \| n_1 \| \cdots \| n_\ell$ of this coin, which allows her to check whether the coin was double-spent.

To do so, $\mathcal{D}$ checks whether n_0 already exists in her database. If this is not the case then the $\mathsf{Deposit}$ is validated and the list $\mathcal{L}$ is updated by adding $n = n_0 \| n_1 \| \cdots \| n_\ell$. Otherwise, if a serial number $\tilde{n}$ beginning with n_0 already exists in her database then with overwhelming probability the coin was double-spent and $\mathcal{D}$ outputs $\perp_1$. She compares the two serial numbers $n = n_0 \| n_1 \| n_2 \| \cdots \| n_\ell$ and $\tilde{n} = n_0 \| \tilde{n}_1 \| \tilde{n}_2 \| \cdots \| \tilde{n}_\ell$ and stops at the last i_0 such that $n_{i_0} = \tilde{n}_{i_0}$. She finally asks for the execution of the $\mathsf{Identify}$ procedure by the Judge on input the two related spendings and i_0.

To identify the double spender, the judge extracts the value committed in $c_{u_{i_0}}$ using her extraction key, which reveals the public key upk_{i_0} of the defrauder. The proof τ_G of identification is a proof of correct opening of the commitment, as done in [8].

5.6 Security Considerations

We now sketch how to prove that our scheme is secure. We have to show that it fulfills all the security requirements given in Section 3.

Claim 1. *Our transferable e-cash system with a judge is secure under the following assumptions: unforgeability of the commuting signature scheme and soundness and witness indistinguishability of Groth-Sahai proofs.*

Unforgeability. Let us assume that an adversary is able to break the unforgeability of our transferable e-cash scheme. We use it as a black box to design a machine which breaks commuting signatures, i.e. their unforgeability under chosen-message attacks.

Given a challenge public key by our challenger, we use it as the bank's public key and generate the remaining parameters as described in our e-cash scheme (without any modifications), and send it to the adversary. We answer all oracle queries by the adversary either by using the appropriate key or by querying the signing oracle provided by our challenger (for BWith and With calls).

Suppose the adversary wins the game. For each of the q_D successfully deposited coins, using the judge's extraction key we open the commitments c_{s_1}, c_{n_0}, c_{n_1} and c_{u_1}. By soundness of π_{s_1}, the extracted signature s_1 is valid on $(n_0, n_1, \mathsf{upk}_1)$, the other extracted values, under the bank's public key. Since every deposit was successful, none of the q_D coins was double-spent, which means that their n_0 components are all different. We have thus q_D signatures on different triples $(n_0, n_1, \mathsf{upk}_1)$. On the other hand, there were fewer calls (q_W) to withdraw oracles, and thus fewer calls to our signing oracle. There must thus be a signature on a message which was not queried to the signing oracle. We output that signature/message pair as a forgery and win thus the unforgeability game with the same probability as the adversary.

Identification of double-spender. As for unforgeability, we use a successful adversary to break unforgeability of the commuting signature scheme. This time we use the public key given by our challenger as the judge's public key and set up the remaining parameters as described in our scheme. We can therefore answer any oracle call by the adversary, except for the certification of a new user in the system (oracle Create), for which we use our signing oracle.

At some point the adversary makes a call to the BDepot oracle that is answered as $(\perp_2, Id, c, c')$, i.e. a double-spending is detected. If the adversary is successful then $\mathsf{Identify}$ outputs (i^*, τ_G) such that either $\mathsf{VerifyGuilt}(\mathsf{pk}_{i^*}, \tau_G) = 0$ or $i^* = 0$. Since each valid coin must contain a valid certificate for the public key corresponding to each transfer, by soundness of the proofs, the adversary must have forged the certificate. Otherwise $\mathsf{Identify}$ would have output an existing user key.

Exculpability. This is shown similarly to unforgeability, except that here we focus on signatures issued by an honest user rather than the bank.

A user with public key upk is accused of double-spending when there are two coins c and c' with serial numbers n and n', such that for some index i, we have $n_0 = n'_0, \ldots, n_i = n'_i$ and $n_{i+1} \neq n'_{i+1}$, and moreover c_{u_i} contains the user's public key upk. Since both coins are valid, by the soundness of the proofs, they contain signatures on $(n_i, n_{i+1}, \mathsf{upk}_{i+1})$ and $(n_i, n'_{i+1}, \mathsf{upk}'_{i+1})$, respectively.

Since an honest user does not transfer or spend one of his coins twice, and it only happens with negligible probability that she chooses twice the same nonce n_i when receiving two different coins, one of the signatures must be a forgery.

The adversary we build against unforgeability of the commuting signature scheme receives the challenge public key and sets it as the public key of a randomly chosen user. It uses the signing oracle to simulate this user, whenever the adversary asks her to spend/transfer a coin. If the probability that the e-cash adversary wins the exculpability game is non-negligible then so is the probability that he wins by framing the user chosen by the simulator. We break thus unforgeability of the commuting signature.

Anonymity properties.

- To achieve Spend-the-Observe Full Anonymity, it suffices to encrypt the messages sent between the users when transferring a coin; this was shown in [4].
- Spend-then-Receive Full Anonymity (formerly known as PA_2) is harder to achieve, since the adversary is given the challenge coin, which he could already have owned before; the adversary therefore must not know the key to detect double-spendings.

  Groth-Sahai proofs are witness indistinguishable in the following sense: if the commitment key is set up as perfectly hiding then the commitments are random values independent of the committed values and the proofs are distributed equally for any such values—as long as they satisfy the equations. Thus, if we set up all commitment keys (the two for the judge and one for the double-spending detector), a coin would not reveal anything about the chosen nonces, the public keys and certificates of its owners and their signatures. Moreover, after being transferred, the coins are perfectly unlinkable, since a randomization transforms one set of random values into an independent set of random values (conditioned on the fact that the values that could have been committed satisfy the equations).

  However, if the coins do not contain any information, we cannot correctly simulate the experiment for StR-FA. In particular, we cannot simulate the deposit and identification oracles, which rely on the detector's and the judge's extraction keys. This is the reason why we introduced the commitments d and $\tilde{c}$, which double some of the values committed in the c's, namely the nonces and the user public keys.

  The anonymity properties are shown by a sequence of game hops. The first game is the original game, and in the second we extract from the commitments d and $\tilde{c}$ to detect and trace double-spendings. In a third game, we set up the judge's key for the commitments c as perfectly hiding. Under SXDH this changes the adversary's behavior only negligibly. In a forth game we simulate the proofs π_{n_i} and $\tilde{\pi}_{u_i}$ of equality of commitments under different keys. This can be done using the trapdoor information for the key for the c-commitments.

  Finally, we mentioned in Section 4.2 that commitments under binding keys are actually ElGamal encryptions of the committed value. Under SXDH we can thus replace such encryptions by random pairs of elements from the corresponding group. When we perform the challenge spending via Spd in the experiment, we replace the commitments/encryptions d_{n_i} and $\tilde{c}_{u_i}$ by random values. This is done by a sequence of hybrid games, replacing one value after the other.

  In the final game now the challenge coin is perfectly random, and does thus not contain any information about the bit b. The adversary's probability of winning the game is thus $\frac{1}{2}$. This concludes the proof for StR-FA as the final game is indistinguishable from the original game.

– The remaining notion, OtR-FA, is proved similarly, but here we cannot replace values d by random ones, as the adversary gets the corresponding extraction key, contained in sk_B. However, we can simply leave the values d in the challenge coin unchanged, as the adversary has never seen them before: he has never owned the coin and does not get the value d_{n_1} when impersonating the bank during a withdraw.

References

1. Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO'10*, volume 6223 of *LNCS*, pages 209–236. Springer, 2010.
2. Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In *CRYPTO'09*, volume 5677 of *LNCS*, pages 108–125, 2009.
3. Marina Blanton. Improved conditional e-payments. In *ACNS'08*, volume 5037 of *LNCS*, pages 188–206, 2008.
4. Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In *ACNS'08*, volume 5037 of *LNCS*, pages 207–223. Springer, 2008.
5. Sébastien Canard, Aline Gouget, and Jacques Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. In *Financial Cryptography'08*, volume 5143 of *LNCS*, pages 202–214. Springer, 2008.
6. David Chaum and Torben P. Pedersen. Transferred cash grows in size. In *EUROCRYPT'92*, volume 658 of *LNCS*, pages 390–407. Springer, 1992.
7. Georg Fuchsbauer. Commuting signatures and verifiable encryption. In *EUROCRYPT'11*, volume to appear of *LNCS*, pages ?–? Springer, 2011.
8. Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable anonymous constant-size fair e-cash. In *CANS'09*, volume 5888 of *LNCS*, pages 226–247. Springer, 2009.
9. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT'08*, volume 4965 of *LNCS*, pages 415–432. Springer, 2008.
10. Tatsuaki Okamoto and Kazuo Ohta. Disposable zero-knowledge authentications and their applications to untraceable electronic cash. In *CRYPTO'89*, volume 435 of *LNCS*, pages 481–496. Springer, 1989.
11. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *CRYPTO'91*, volume 576 of *LNCS*, pages 324–337. Springer, 1991.
12. Alfredo De Santis and Moti Yung. Crptograpic applications of the non-interactive metaproof and many-prover systems. In *CRYPTO'90*, volume 537 of *LNCS*, pages 366–377. Springer, 1990.
13. Hans van Antwerpen. *Electronic Cash.* PhD thesis, CWI, 1990.
14. Sebastiaan H. von Solms and David Naccache. On blind signatures and perfect crimes. *Computers & Security*, 11(6):581–583, 1992.