Achieving Optimal Anonymity in Transferable E-cash with a Judge*

Olivier Blazy[1], Sébastien Canard[2], Georg Fuchsbauer[3], Aline Gouget[4], 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

Abstract. Electronic cash (e-cash) refers to money exchanged electronically. The main features of traditional cash are usually considered also desirable in the context of e-cash. This is for example the case for *off-line transferability*, meaning the recipient of cash in a transaction can transfer received money 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*, as introduced at ACNS 2008. 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 the strongest notions of anonymity, and the commuting signatures by Fuchsbauer.

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 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.

* This work has been financially supported by the French Agence Nationale de la Recherche under the PACE 07 TCOM project and by the European Commission's Seventh Framework Programme (FP7) under contract number ICT-2007-216676 ECRYPT II, while the third author was at École normale supérieure, Paris, France.

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 [6] 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 spends. 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 [8] 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 [6]. Perfect anonymity can therefore not be achieved by a transferable e-cash scheme. Due to this impossibility result, Canard and Gouget [6] 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 [6] 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 [12, 13] or strong anonymity [15, 8, 7, 4]. A generic construction of a transferable e-cash system with FA and PA_1 security from a one satisfying strongly anonymity is shown in [6]. 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 [6] that cannot be implemented effectively. This is due to its use of complex meta-proofs [14] which allow the blinding of previous transfers of a coin, even for a previous owner of that coin.

Subsequently, Fuchsbauer et al. [10] 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 fraudster.

In conclusion, the remaining open problem is an efficient transferable e-cash scheme satisfying 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 [11, 2] and on the recent primitive of commuting signatures [9] 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 fraudster 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* [16]. 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 [6] to include the judge authority. Moreover, in accordance with [6], 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 [6], 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 $(\mathsf{sk}_\mathcal{U}, \mathsf{pk}_\mathcal{U})$.
- $\mathsf{Withdraw}(\mathcal{W}[\mathsf{sk}_\mathcal{W}, \mathsf{pk}_\mathcal{U}], \mathcal{U}[\mathsf{sk}_\mathcal{U}, \mathsf{pk}_\mathcal{B}])$ is an interactive protocol where $\mathcal{U}$ withdraws from $\mathcal{B}$ one transferable coin. 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 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 saves that c is a spent coin 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/fresh, 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 [6] is the additional of a new definition of the key generation algorithm $\mathsf{JKeyGen}()$ and the modification of the procedure to identify a fraudster 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 a transferable e-cash system: we introduce global variables (according to [6]) 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

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

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, is denoted by $\mathcal{RU}$, and the set of users who have spent a coin to the adversary, is denoted by $\mathcal{SU}$. These modifications are done in order to improve the understanding of the original description of oracles provided in [6].

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 input pk_i is $\perp$ then this 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.

- 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 passive. 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.
- The oracle $\mathsf{Spd\&Rcv}$ 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. [2]

[2] The main difference between these oracles and those described in [6] is the modification of the oracle $\mathsf{BDepo}()$ and the definition of the new oracle $\mathsf{Idt}(Id, c, c')$. In [6] 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.

- The oracle $\mathsf{BDepo}()$ plays the role of the bank during a $\mathsf{Deposit}$ protocol and interacts with the adversary. The oracle finally gives the output of $\mathsf{Deposit}$ 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 [8], 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 [6]. 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 experiment and definition.

```
$\underline{\mathbf{Exp}^{\mathsf{unfor}}_{\Pi,\mathcal{A}}(\lambda)}$
  – $(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
  – $cont \leftarrow \mathsf{true}$: $st \leftarrow \emptyset$;
  – While $(cont = \mathsf{true})$ do {
      – $(cont, st) \leftarrow \mathcal{A}^{\mathsf{Create,Corrupt,BWith,With,Rcv,Spd,BDepo,Depo}}(st, \mathsf{par}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{B})$;
          Let $q_W$, $q_S$ and $q_R$ denote the number of successful queries to
          the oracles BWith, Spd and Rcv, respectively.
      – If $(q_W + q_S < q_R)$ return 1; }
  – Return $\perp$.
```

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 $\mathcal{A}$.*

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 {
    – $(st) \leftarrow \mathcal{A}^{\mathsf{Create,Corrupt,BWith,With,Rcv,Spd,BDepo,Depo,Idt}}(st, \mathsf{par}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{B})$;
    – If a call to $\mathsf{BDepo}$ 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 $(i^* = 0)$ 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 $\boldsymbol{Succ}^{\mathit{ident}}_{\Pi,\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{\mathit{ident}}_{\Pi,\mathcal{A}}(\lambda) = 1]$. Π* identifies double spenders *if the function $\boldsymbol{Succ}^{\mathit{ident}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.*

3.3 Exculpability

This notion protects the 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.

```
$\underline{\mathbf{Exp}^{\mathsf{excul}}_{\Pi,\mathcal{A}}(\lambda)}$
 – $(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
 – $(Id^*, c_1^*, c_2^*, i^*, \tau^*) \leftarrow \mathcal{A}^{\mathsf{Create,Corrupt,UWith,Rcv,Spd,UDepo,Idt}}(st, \mathsf{par}, \mathsf{sk}_\mathcal{J}, \mathsf{sk}_\mathcal{B})$;
 – 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 $\boldsymbol{Succ}^{\mathit{excul}}_{\Pi,\mathcal{A}}(\lambda) = \Pr[\boldsymbol{Exp}^{\mathit{excul}}_{\Pi,\mathcal{A}}(\lambda) = 1]$. Π is said to be* exculpable *if the function $\boldsymbol{Succ}^{\mathit{excul}}_{\Pi,\mathcal{A}}(\cdot)$ is negligible for any polynomial-time adversary $\mathcal{A}$.*

3.4 Anonymity Properties in Transferable E-cash

Regarding anonymity, Canard and Gouget [6] distinguish between five different notions: weak anonmity (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 [6] but slightly modify the terminology[1].

- *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. to 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 [6].

We now formally define these anonymity notions, based on the corresponding experiments described below.

```
$\underline{\mathbf{Exp}^{\mathsf{otr\text{-}fa\text{-}}b}_{\Pi,\mathcal{A}}(\lambda) \quad // \ b \in \{0,1\},\ \mathcal{A} = (\mathcal{A}_{ch}, \mathcal{A}_c, \mathcal{A}_{gu})}$
  – $(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
  – $(i_0^*, i_1^*, st) \leftarrow \mathcal{A}_{ch}^{\mathsf{Create},\mathsf{Corrupt},\mathsf{UWith},\mathsf{Rcv},\mathsf{Spd},\mathsf{Spd\&Rcv},\mathsf{UDepo},\mathsf{Idt}}(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{J})$;
  – If $\mathsf{sk}_{i_0^*} =\perp \vee \mathsf{sk}_{i_1^*} =\perp \vee \mathsf{comp}(i_0^*, i_1^*) = 0 \vee\ i_0^* \in \mathcal{RU} \vee\ i_1^* \in \mathcal{RU}$,
    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 $st_c$;
  – $b^* \leftarrow \mathcal{A}_{gu}^{\mathsf{Create},\mathsf{Corrupt},\mathsf{UWith},\mathsf{Rcv},\mathsf{Spd},\mathsf{Spd\&Rcv},\mathsf{UDepo},\mathsf{Idt}}(st_c)$;
  – Return $b^*$.
```

```
$\underline{\mathbf{Exp}^{\mathsf{sto\text{-}fa\text{-}}b}_{\Pi,\mathcal{A}}(\lambda) \quad // \ b \in \{0,1\},\ \mathcal{A} = (\mathcal{A}_{ch}, \mathcal{A}_{gu})}$
  – $(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
  – $(i_0^*, i_1^*, i_2^*, st) \leftarrow \mathcal{A}_{ch}^{\mathsf{Create},\mathsf{Corrupt},\mathsf{UWith},\mathsf{Rcv},\mathsf{Spd},\mathsf{Spd\&Rcv},\mathsf{UDepo},\mathsf{Idt}}(\mathsf{par}, \mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{J})$;
  – 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 $out \leftarrow \mathsf{Spd\&Rcv}(j^*, i_b^*, i_2^*)$;
  – $b^* \leftarrow \mathcal{A}_{gu}^{\mathsf{Create},\mathsf{Corrupt},\mathsf{UWith},\mathsf{Rcv},\mathsf{Spd},\mathsf{Spd\&Rcv},\mathsf{UDepo},\mathsf{Idt}}(out, st_c)$;
  – If an oracle call involed the coin used in $\mathsf{Spd\&Rcv}$ then return $\perp$;
  – Return $b^*$.
```

```
$\underline{\mathbf{Exp}^{\mathsf{str\text{-}fa\text{-}}b}_{\Pi,\mathcal{A}}(\lambda) \quad // \ b \in \{0,1\},\ \mathcal{A} = (\mathcal{A}_{ch}, \mathcal{A}_c, \mathcal{A}_{gu})}$
  – $(\mathsf{par}, \mathsf{sk}_\mathcal{B} = (\mathsf{sk}_\mathcal{W}, \mathsf{sk}_\mathcal{D}), \mathsf{pk}_\mathcal{B} = (\mathsf{pk}_\mathcal{W}, \mathsf{pk}_\mathcal{D}), \mathsf{sk}_\mathcal{J}, \mathsf{pk}_\mathcal{J}) \leftarrow \mathsf{AllGen}(1^\lambda)$;
```

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

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

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 $\boldsymbol{Adv}_{\Pi,\mathcal{A}}^{\mathsf{c}}(\lambda) = \Pr[\boldsymbol{Exp}_{\Pi,\mathcal{A}}^{\mathsf{c\text{-}1}}(\lambda) = 1] - \Pr[\boldsymbol{Exp}_{\Pi,\mathcal{A}}^{\mathsf{c\text{-}0}}(\lambda) = 1]$. Π is said to be* Observe-then-Receive fully anonymous *(resp.* Spend-then-Observe fully anonymous*,* Spend-then-Receive fully anonymous*) if the above function $\boldsymbol{Adv}_{\Pi,\mathcal{A}}^{\mathsf{otr\text{-}fa}}(\cdot)$ (resp. $\boldsymbol{Adv}_{\Pi,\mathcal{A}}^{\mathsf{sto\text{-}fa}}(\cdot)$, $\boldsymbol{Adv}_{\Pi,\mathcal{A}}^{\mathsf{str\text{-}fa}}(\cdot)$) is negligible for any polynomial-time adversary $\mathcal{A}$.*

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 The SXDH Assumption

The SXDH assumption was introduced in [5] and can be defined as follows.

Definition 5 (Symmetric external DH Assumption (SXDH)). *Let $\mathbb{G}_1, \mathbb{G}_2$ be cyclic groups of prime order, $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear map. The SXDH assumption states that the DDH assumption holds in both $\mathbb{G}_1$ and $\mathbb{G}_2$.*

4.2 Groth-Sahai Proofs

Groth and Sahai proposed in [11] the first efficient non-interactive proof system for bilinear groups which does not need the random oracle model. In this paper, we will use the SXDH based Groth-Sahai (GS for short) proof system. Those proofs perfectly fit in our model, as they provide the required flexibility through their randomization property, and through their NIWI aspect they let us preserve the anonymity of the different players.

We use SXDH-based Groth-Sahai commitments in a pairing-friendly setting $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$, in order to commit elements while remaining able to prove relations satisfied by the associated plaintexts. The commitment key is:

$$\mathbf{u} = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} \in \mathbb{G}_1^{2\times 2} \quad \text{and} \quad \mathbf{v} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{pmatrix} \in \mathbb{G}_2^{2\times 2}.$$

Depending if we want to be in the perfectly hiding or the perfectly binding (mostly for simulations in security proof) setting, the initialization of the parameters will vary between: $\boldsymbol{u_1} = (g_1, u)$ with $u = g_1^\lambda$ and $\boldsymbol{u_2} = \boldsymbol{u_1}^\mu$ with $\lambda, \mu \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ (and so $\mathbf{u}$ is a Diffie-Hellman tuple in $\mathbb{G}_1$). Whereas, in the perfectly hiding setting, $\boldsymbol{u}_2 = \boldsymbol{u}_1{}^\mu \odot (1, g_1)^{-1}$: $\boldsymbol{u_1} = (g_1, g_1^\lambda)$ and $\boldsymbol{u_2} = (g_1^\mu, g_1^{\lambda\mu-1})$. The same goes in $\mathbb{G}_2$ for $\boldsymbol{v}$.

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

The same can be done in $\mathbb{G}_2$ if we consider $\boldsymbol{v}$ and g_2.

Proofs. Under the SXDH assumption, the two initializations of the commitment key are indistinguishable. The Groth-Sahai methodology gives us a way to build a pair $(\pi, \theta) \in \mathbb{G}_2^{2\times 2} \times \mathbb{G}_1^{2\times 2}$ proving the satisfiability of a set of equations over a bilinear group where such commitments are involved to hide the plaintexts. Being able to produce a valid pair implies knowing plaintexts verifying the appropriate relation.

Randomization. The commitments can easily be randomized given a commitment c in $\mathbb{G}_1^2$: one can choose two random values s'_1, s'_2 and compute a valid randomization $c' = (c_1 \cdot u_{1,1}^{s'_1} \cdot u_{2,1}^{s'_2}, c_2 \cdot u_{1,2}^{s'_1} \cdot u_{2,2}^{s'_2})$, which leads to some changes on (π, θ) depending on the original equation.

The proofs (π, θ) can easily be randomized on its own.

Optimization. Depending on the original equations the proofs can require less elements. For example, the main operation in the protocol will be to prove that two committed nonces are the same. Due to some limitation (coming from the extractability requirement) on the nonces, they have to be committed in different groups. The resulting equation is called a quadratic equation, which means that $(\pi, \theta) \in \mathbb{G}_2^2 \times \mathbb{G}_1^2$.

4.3 Commuting Signatures

Commuting signatures and verifiable encryption [9] is a primitive combining a signature scheme (the automorphic signature from [1] whose messages are group elements) with Groth-Sahai (GS) proofs. This allows 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 to a signature (under his secret key) on that message (without knowing it though) and a proof that the commitment contains a valid signature on the value committed in $\mathbf{C}$.
- $\mathsf{AdPrC}_{\mathcal{K}}$: 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, $\mathsf{AdPrC}_{\mathcal{K}}$ 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 given a GS proof of validity w.r.t. his signature verification key. Analogously, the output of $\mathsf{AdPrC}_{\mathcal{K}}$ is the same as a proof constructed for the known committed values.

We use commuting signatures to enable users to produce committed signatures on values that are only available as commitments and make a proof of validity w.r.t. their verification key, which is also given as a commitment.

5 A New Tranferable E-cash System with Judge

In this section, we describe our new transferable e-cash system with judge, based on the cryptographic building blocks described in Section 4. We describe a solution in which the withdrawer is anonymous *w.r.t.* the bank, which is not classical. This is motivated by the fact that our withdrawal is very similar to the spending protocol and that it is easy to make the withdrawer non-anonymous if one wants to. A customer may want to anonymously buy tickets (using some anonymous payment systems) from the transport provider that she will next transfer to other customers.

5.1 Overview of the 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 each owner of this coin. Indeed, n_0 is chosen by the bank, n_1 by the withdrawer, n_2 by the one who receives this coin from the withdrawer, etc. A double-spending is detected when there are two coins N and N' such that $n_0 = n'_0$. Next, the minimum value i such that $n_i \neq n'_i$ corresponds to the first protocol of spending of this coin such that a double spending has been done. The identity of the double-spender can finally be retrieved by the judge, using an extractable commitment of the user public key.

During a spending protocol between $\mathcal{U}_i$ and $\mathcal{U}_{i+1}$, the spender $\mathcal{U}_i$ signs (i) the nonce she has chosen during the reception of the coin (previous spending protocol, or withdrawal procedure), (ii) the nonce she is given by the actual receiver $\mathcal{U}_{i+1}$, and (iii) the signature public key of the latter. This element is signed to permit $\mathcal{U}_{i+1}$ to further spend this coin. In fact, $\mathcal{U}_{i+1}$ is the only one that can use the secret key which is related to the signed public key, and she is thus the only one that will be able to spend this coin. The commuting signature is used in this protocol since it provides the possibility to sign messages that are committed, using a key pair which is also committed and such that everyone can check that the result is correct.

During the spending protocol of a coin, its entire history (i.e. previous spending protocols of it) is transmitted. This is necessary to provide unforgeability, identification of double-spending and non-frameability, without requiring data to be stored by the user and provided later on-demand of a judge to prove his honesty (as was the case in the weaker model in [10]). Since we want the scheme to achieve perfect anonymity, the history of a coin is re-randomized by using the randomization techniques for extractable commitments and Groth-Sahai proofs [2].

5.2 Key-Generation Algorithms

During the generation phase, the judge $\mathcal{J}$ generates two different extraction keys (see extractable commitments in Section 4) for the identification of double spenders. Similarly, the double-spending detector $\mathcal{D}$ generates a pair of commitment/extraction key for the GS methodology.

We denote a commitment under $\mathcal{J}$'s key by either c (first key pair) or $\tilde{c}$ (second key pair) 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 $\mathsf{Sign}_{\mathcal{J}}(m)$. The bank $\mathcal{B}$ and each user $\mathcal{U}$ generate also generate key pairs $(\mathsf{bsk}, \mathsf{bpk})$ (resp. $(\mathsf{usk}, \mathsf{upk})$) for the commuting signature scheme (see Section 4). Moreover, each user $\mathcal{U}$ obtains from the judge $\mathcal{J}$ a signature on her public key as membership certificate: $\mathsf{cert} = \mathsf{Sign}_{\mathcal{J}}(\mathsf{upk})$. In the following, we differentiate the users by using numbers $\mathcal{U}_1$,$\mathcal{U}_2$, 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 also 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 actually registered. This can be done using the fact that the certificate is an *automorphic* signature (for which we can use GS proofs to prove that a committed value is a valid signature on another committed value, in this case upk_1).

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 w.r.t. $\mathcal{D}$'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 on top of each other, as was shown in [10].

We formalize the above in the following protocol:

1. [$\mathcal{U}_1$] picks at random a nonce n_1 and sends $\mathcal{B}$ 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, $\mathcal{U}_1$ sends 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_{u_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 the same.
2. [$\mathcal{B}$] 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 available in the clear). Finally, the bank sends to $\mathcal{U}_1$ the coin defined as

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

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 in randomizing all its components. Thus, $\mathsf{coin}_1^{(1)} = (c_{n_0}^{(1)}, c_{n_1}^{(1)}, d_{n_0}^{(1)}, d_{n_1}^{(1)}, \pi_{n_0}^{(1)}, \pi_{n_1}^{(1)}, c_{u_1}^{(1)}, \tilde{c}_{u_1}^{(1)}, c_{c_1}^{(1)}, \pi_{u_1}^{(1)}, \tilde{\pi}_{u_1}^{(1)}, c_{s_1}^{(1)}, \pi_{s_1}^{(1)})$. The randomization of the commitments and proofs is done as explained in Section 4.2 and [2].

5.4 Spending Protocol

This is a protocol between a user $\mathcal{U}_1$ holding a coin $\mathsf{coin}_1 = (c_{n_0}, c_{n_1}, d_{n_0}, d_{n_1}, \pi_{n_0}, \pi_{n_1}, c_{u_1}, \tilde{c}_{u_1}, c_{c_1}, \pi_{u_1}, \tilde{\pi}_{u_1}, c_{s_1}, \pi_{s_1})$ 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, 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 $\mathsf{AdPrC}_{\mathcal{K}}$ (see Section 4.3), she converts the proof into one stating 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).

1. [$\mathcal{U}_2$] picks at random a nonce n_2 and computes two commitments c_{n_2} and d_{n_2} to it (for $\mathcal{J}$ and $\mathcal{D}$) and a proof π_{n_2} that the two committed values are equal.
   Moreover, $\mathcal{U}_2$ makes 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 the proof π_{u_2} that c_{u_2} and c_{c_2} are consistent and a proof $\tilde{\pi}_{u_2}$ that the committed values in c_{u_2} and $\tilde{c}_{u_2}$ are equals. She sends $(c_{n_2}, d_{n_2}, \pi_{n_2}, c_{u_2}, \tilde{c}_{u_2}, c_{c_2}, \pi_{u_2}, \tilde{\pi}_{u_2})$ to $\mathcal{U}_1$
2. [$\mathcal{U}_1$] randomizes coin_1 to $\mathsf{coin}_1^{(1)}$ as described above and 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. Using $\mathsf{AdPrC}_{\mathcal{K}}, \mathcal{U}_1$ converts π'_{s_2} to a proof asserting validity w.r.t. the key committed in $c_{u_2}^{(1)}$, and sends the coin $\mathsf{coin}_2 = (\mathsf{coin}_1^{(1)}, c_{n_2}, d_{n_2}, \pi_{n_2}, c_{u_2}, \tilde{c}_{u_2}, c_{c_2}, \pi_{u_2}, \tilde{\pi}_{u_2}, c_{s_2}, \pi_{s_2})$ to $\mathcal{U}_2$.

Before $\mathcal{U}_2$ spends the coin, she randomizes it to obtain $\mathsf{coin}_2^{(1)} = (\mathsf{coin}_1^{(2)}, c_{n_2}^{(1)}, d_{n_2}^{(1)}, \pi_{n_2}^{(1)}, c_{u_2}^{(1)}, \tilde{c}_{u_2}^{(1)}, c_{c_2}^{(1)}, \pi_{u_2}^{(1)}, \tilde{\pi}_{u_2}^{(1)}, c_{s_2}^{(1)}, \pi_{s_2}^{(1)})$.

5.5 Deposit and Identify Procedures

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 the 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 beginning with n_0 already exists in her database, then this coin is not fresh (the depositary is cheating) and $\mathcal{D}$ outputs $\perp_1$. She then 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 first i_0 such that $n_{i_0} \neq \tilde{n}_{i_0}$. 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 uses either the commitment $c_{u_{i_0-1}}$ or the commitment $\tilde{c}_{u_{i_0-1}}$, using the appropriate extraction key, which gives her, in both cases, the value upk_{i_0-1} and thus the identity of the fraudster.

5.6 Security Considerations

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

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

Proof (sketch). We focus on each desired security properties.

- **Unforgeability**. We assume that an adversary is able to break the unforgeability of our transferable e-cash scheme and we use it, as a black box, to design a machine which breaks the unforgeability of the commuting signature.
  First of all, we use the public key given by our challenger as the bank's public key and give it to our adversary. Other parameters are generated as described in our e-cash scheme, without any modifications.
  We can answer any request to oracles by the adversary, either by using the appropriate key, or by making use of the commuting signing oracle during a Witdraw protocol (oracles BWith and Withdraw).
  After each successful call to a Rcv oracle, our machine uses the extraction key of the judge to open c_{s_1}, that is the commitment on the bank's commuting signature on the first spending (that is the withdrawal) to retrieve the embedded signature. There are then two cases:
  1. the extracted signature comes from the signing oracle. Then the experiment is continued.
  2. the extracted signature does not come from the signing oracle. This is then a forge and we have broken the unforgeability of the signing oracle.

  As the adversary wins the game when $q_W + q_S < q_R$, where q_W (resp. q_S, q_R) denotes the number of successful queries to the oracle BWith (resp. Spd, Rcv), we necessarily fall once into the second case above. As there are, by assumption, no detection of a double spending and as we have made in total q_W queries to the signing oracle, the q_R extractions necessarily give q_S correlated signatures (when the spending corresponds to the spending of an already spent coin) and $q_W + 1$ different signatures. This gives our machine the same advantage as the adversary to break the unforgeability of the commuting signature, which concludes the proof.
- **Identification of Double-Spender**. We assume that an adversary is able to break the identification of double-spender property of our transferable e-cash scheme and we use it, as a black box, to design a machine which breaks the unforgeability of the commuting signature.
  First of all, we use the public key given by our challenger as the judge's public key and give it to our adversary. Other parameters are generated as described in our e-cash scheme, without any modifications.

We can answer any request to oracles by the adversary, either by using the appropriate key, or by making use of the commuting signing oracle for the certification of a new user in the system (oracle Create).
At any time of the experiment, the adversary outputs a new state st such that a call to the BDepot oracle outputs $(\perp_2, Id, \pi, \pi^*)$ (a double-spending is detected). If the adversary is successful, then the $\mathsf{Identify}$ oracle outputs (i^*, τ_G) such that either $\mathsf{VerifyGuilt}(\mathsf{pk}_{i^*}, \tau_G) = 0$, or $i^* = 0$. Our machine next uses the extraction key of the judge to obtain the certificates into the two spendings. As the adversary wins with non negligible probability, the related certificate has necessarily not been requested to the signing commuting and is thus a forge, which concludes the proof.

- **Exculpability**. This result is quite similar to the unforgeability one, except that we now focus on the signature scheme of each honest user instead of the one of the bank. In fact, the adversary succeeds in the exculpability experiment if it succeeds in producing a commuting signature on behalf of an honest user. As an honest payer signs both the previously chosen nonce and the new one, this is not feasible to force her to sign twice the nonce she has previously chosen.
  The main difference is that we do not know a priori which user will be falsely accused of being a double-spender. For this, we use some kind of one-more problem [3] where the challenger gives us ℓ different signature public keys and access to the ℓ corresponding signing oracles. We are also able to "corrupt" at most $\ell - 1$ secret keys and our aim is to finally outputs ℓ forges, one for each signature public key. Using a reduction similar to the one for the unforgeability property, and using the above remark, we use the signing oracle of the right user each time the adversary ask this honest user to spend a coin and we use our corruption oracle each time the adversary corrupt a user. At the end, we extract the forged signature from the output of the adversary, and we use our corruption oracle to obtain the last secret keys, which concludes our proof.
- **Anonymity Properties**. Regarding Section 3.4, we have to consider the FA, PA_1 and the PA_2 properties. In fact, the related security proofs are very close and we only detail the FA one.
  In a nutshell, Groth-Sahai proofs are witness indistinguishable and are always re-randomized (see [2]). As a consequence, they do not give any information about user's identity. Moreover, in our scheme, extractable commitments are computationally hiding (and perfectly binding) but, using standard techniques (see for example [2, 10]), we make them perfectly hiding during our reduction. As a consequence, it follows that they do not reveal any more information about user's identity. Note that we can easily simulates the environment of the adversary. The only problem we may have is to correctly simulate the identification of a double-spending produced by the adversary. That's why we introduce c and $\tilde{c}$ in our scheme (see Section 5.2).
  More precisely, we start by running the experiment fa_0 for an adversary $\mathcal{A}$ and the simulator $\mathcal{B}$. We define a first game G_0 which is the real game, and we denote ϵ the advantage of the adversary. We then define a game

G_1, in which the Groth Sahai commitments c, d are replaced by perfectly hiding ones. We can then use random values in the commitments and adapt the proofs (thanks to the NIWI property of Groth Sahai methodology) so that the adversary can't distinguish this game from the previous one. The simulator still uses real values in the challenge commitments.
We then define a last game G_2, where we use random values for the challenge users. We still can simulate the oracles as follows.

- To answer a $\mathsf{CSpd}, \mathsf{USpd}, \mathsf{Spd}$ & Rcv query on one of the two anonymity challenges, $\mathcal{B}$ simply puts a random value in the appropriate commitment $\tilde{c}_{u_i}$ and adapts the proof, using standard techniques on Groth Sahai proofs.
- To answer any queries on any other honest users, the simulators runs the standard version of the protocol.
- To answer a Ident query, the simulator extracts the value from $\tilde{c}_{u_i}$ and returns it (basically an honest user shouldn't do any double spending).

In this final game, all the values supposed to be linked to the challenge users are now random values, therefore $\epsilon_2 = 0$, but, as Groth Sahai proofs are NIWI, we have $\epsilon_2 = \epsilon$.
We can run exactly the same experiment for the game fa_1. The corresponding game G_2 is exactly the same as in fa_0, so we have $\epsilon = 0$ too.
This concludes the proofs of the FA property. The PA_1 property can be proven similarly with very slight modifications. Regarding the PA_2 property, the fact that the adversary has already possessed the coin it is receiving does not change anything in our above games. The answers to the oracles are done similarly, since we use the re-randomization technique for both commitments and Groth Sahai proofs.

$\square$

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. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3):185–215, 2003.
4. Marina Blanton. Improved conditional e-payments. In *ACNS'08*, volume 5037 of *LNCS*, pages 188–206, 2008.
5. Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO'04*, volume 3152 of *LNCS*, pages 41–55. Springer, 2004.
6. Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In *ACNS'08*, volume 5037 of *LNCS*, pages 207–223. Springer, 2008.
7. 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.

8. David Chaum and Torben P. Pedersen. Transferred cash grows in size. In *EUROCRYPT'92*, volume 658 of *LNCS*, pages 390–407. Springer, 1992.
9. Georg Fuchsbauer. Commuting signatures and verifiable encryption. In *EUROCRYPT'11*, volume to appear of *LNCS*, pages ?–? Springer, 2011.
10. 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.
11. 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.
12. 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.
13. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *CRYPTO'91*, volume 576 of *LNCS*, pages 324–337. Springer, 1991.
14. 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.
15. Hans van Antwerpen. *Electronic Cash*. PhD thesis, CWI, 1990.
16. Sebastiaan H. von Solms and David Naccache. On blind signatures and perfect crimes. *Computers & Security*, 11(6):581–583, 1992.