Divisible E-Cash from Constrained Pseudo-Random Functions

Florian Bourse and Olivier Sanders

Orange Labs, Applied Crypto Group, Cesson-Sévigné, France

Abstract. Electronic cash (e-cash) is the digital analogue of regular cash which aims at preserving users' privacy. Following Chaum's seminal work, several new features were proposed for e-cash to address the practical issues of the original primitive. Among them, *divisibility* has proved very useful to enable efficient storage and spendings. Unfortunately, it is also very difficult to achieve and, to date, only few constructions exist, all of them relying on complex mechanisms that can only be instantiated in one specific setting.
In this work, we study the links between divisible e-cash and constrained pseudo-random functions (PRFs), a primitive recently formalized. We show that one can construct divisible e-cash systems from constrained PRFs achieving some specific properties that we identify. Actually, we provide two frameworks for divisible e-cash that essentially differ in the kind of properties expected from the PRFs. We prove the security of our generic frameworks and provide examples of constrained PRFs satisfying our requirements. Finally, we exhibit a problem in many e-cash systems that invalidates some of their security proofs.

1 Introduction

Electronic payment systems offer high usage convenience to their users but at the cost of their privacy. Indeed, transaction data, such as payee's identity, date and location, leak sensitive information about the users, such as their whereabouts, their religious beliefs, their health status, etc.

However, secure e-payment and strong privacy are not incompatible, as shown by Chaum in 1982 [Cha82] when he introduced the concept of electronic cash (*e-cash*). Informally, e-cash can be thought of as the digital analogue of regular cash with special focus on users' privacy. Such systems indeed consider three kinds of parties: the bank, the user and the merchant. The bank issues coins that can be withdrawn by users and then spent to merchants. Eventually, the latter deposit the coins on their account at the bank. Compared to other electronic payment systems, the benefit of e-cash systems is that the bank is unable to identify the author of a spending. More specifically, it can neither link a particular withdrawal -even if it knows the user's identity at this stage- to a spending nor link two spendings performed by the same user.

At first sight, this anonymity property might seem easy to achieve: one could simply envision a system where the bank would issue the same coin (more specifically, one coin for each possible denomination) to each user. Such a system would obviously be anonymous but it would also be insecure. Indeed, although e-cash aims at mimicking regular cash, there is an intrinsic difference between them: e-cash, as any electronic data, can easily be duplicated. This is a major issue because it means that a user could spend the same coin to different merchants. Of course, some hardware countermeasures (such as storing the coins on a secure element) can be used to mitigate the threat but they cannot remove it. Moreover, the prospect of having an endless (and untraceable) reserve of coins will constitute a strong incentive to attack this hardware whose robustness is not without limits.

To deter this bad behaviour, e-cash systems must therefore enable (1) detection of re-used coins and (2) identification of defrauders. Besides invalidating the trivial solution sketched above (the fact that everyone uses the same coin prevents any identification procedure) these requirements impose very strong constraints on e-cash systems. They indeed mean that users should be anonymous as long as they act honestly while being traceable as soon as they will begin to overspend even one cent.

Chaum's idea, taken up by all subsequent works, was to associate each withdrawn coin with a unique identifier called a "serial number"[1]. The latter remains unknown to all parties, except the user, until the coin is spent. At this time, it becomes public and so can easily be compared to the set of all serial numbers of previously spent coins. A match then acts as a fraud alert for the bank which can then run a specific procedure to identify the cheater.

Unfortunately, by reproducing the features of regular cash, e-cash also reproduces its drawbacks, in particular the problem of paying the exact amount. Worse, as we explain below, the inherent limitations of e-cash compound this issue that becomes much harder to address in a digital setting. This has led cryptographers to propose a wide variety of solutions to mitigate the impact on user's experience. They include for example on-line e-cash, transferable e-cash or divisible e-cash.

1.1 Related Works

The original solution proposed by Chaum for anonymous payment was based on the concept of blind signature. This primitive, later formalized in [PS96, PS00], allows anyone to get a signature σ on a message m that is unknown to the signer. Moreover, the latter will be unable to link the pair (σ, m) to a specific issuance. Applying this idea to the payment context leads to the following e-cash system. A coin is a blind signature issued by a bank to a user during a withdrawal. To spend his coin, the user simply shows the signature to a merchant who is able to verify it using the bank's public key. Two cases may then appear. Either the e-cash system does not allow identification of defrauders, in which case the bank must be involved in the protocol to check that this coin has not already been spent. The resulting system is then referred to as *on-line* e-cash. Otherwise, the

[1] Actually, this specific terminology appeared later [CFN90] but this notion is implicit in the Chaum's paper.

coin may be deposited later to the bank, leading to an *off-line* e-cash system. Obviously, the latter solution is preferable since it avoids a costly connection to the servers of the bank during the payment. In the following, we will only consider off-line e-cash systems.

Theoretically, the problem of anonymous payment is thus solved by blind signatures for which several instantiations have been proposed (see *e.g.* [PS00]). However, as we mention above, it remains to address the problem of paying the exact amount, which becomes trickier in a digital setting. Indeed, let us consider a consumer that owns a coin whose denomination is €10 and that wants to pay €8.75. A first solution could be to contact his bank to exchange his coin against coins of smaller denominations but this would actually reintroduce the bank in the spending process and so would rather correspond to an on-line system. It then mainly remains two kinds of solutions: those where the merchant gives back change and those that only use coins of the smallest possible denomination (*e.g.* €0.01). They both gave rise to two main streams in e-cash: *transferable* e-cash and *compact*/*divisible* e-cash.

Let us go back to our example. At first sight, the simplest solution (inspired from regular cash) is the one where the merchant gives back change, by returning, for example, a coin of €0.05, one of €0.20 and one of €1. However, by receiving coins, the user technically becomes a merchant (in the e-cash terminology) which is not anonymous during deposit. Therefore, the only way to retain anonymity in this case is to ensure transferability of the coin, meaning that the user will be able to re-spend the received coins instead of depositing them. While this is a very attractive feature, it has unfortunately proved very hard to achieve. Worse, Chaum and Pedersen [CP93] have shown that a transferable coin necessarily grows in size after each spending. Intuitively, this is due to the fact that the coins must keep information about each of its owner to ensure identification of defrauders. In the same paper, Chaum and Pedersen also proved that some anonymity properties cannot be achieved in the presence of an unbounded adversary. Their result were later extended by Canard and Gouget [CG08] who proved that these properties were also unachievable under computational assumptions. More generally, identifying the anonymity properties that a transferable e-cash system can, and should, achieve has proved to be tricky [CG08, BCFK15].

All these negative results perhaps explain the small number of results on transferable e-cash, and even quite recent constructions ([CGT08, BCF+11, BCFK15]) are too complex for a large-scale deployment or rely on a very unconventional model [FPV09]. In particular, none of them achieves optimality with respect to the size, meaning that the coin grows much faster than the theoretical pace identified by Chaum and Pedersen.

Now let us consider our spending of €8.75 in the case where all coins are of the smallest possible denomination. This means that the user no longer has a coin of €10 but now has 1000 coins of €0.01. Such a system can handle any amount without change but must provide an efficient way to store and to spend hundreds of coins at once. A system offering efficient storage is said *compact* and a system supporting both efficient storage and spending is said *divisible*.

Anonymous compact e-cash was proposed by Camenisch, Hohenberger and Lysyanskaya [CHL05] and was informally based on the following idea. Let N be the amount of a wallet withdrawn by a user (*i.e.* the wallet contains N coins that all have the same value). During a withdrawal, a user gets a certificate on some secret value s[2] that will be used as a seed for a pseudo-random function (PRF) F, thus defining the serial numbers of the N coins as $F_s(i)$ for $i \in [1, N]$.

To spend the i-th coin, a user must then essentially reveal $F_s(i)$ and prove, in a zero-knowledge way, that it is well-formed, *i.e.* that (1) s has been certified and that (2) the serial number has been generated using F_s on an input belonging to the set $[1, N]$. All of these proofs can be efficiently instantiated in many settings. Anonymity follows from the zero-knowledge property of the proofs and from the properties of the pseudo-random function: intuitively it is hard to decide if $F_s(i)$ and $F_s(j)$ have been generated using the same function F_s.

Unfortunately, compact e-cash only provides a partial answer to the practical issues of spendings: storage is very efficient but the coins must still be spent one by one, which quickly becomes cumbersome. An ultimate answer to this issue was then provided by Okamoto and Ohta [OO92] and later named *divisible* e-cash. The core idea of divisible e-cash is that the serial numbers of a divisible coin[3] can be revealed by batches, leading to efficient spendings.

However, this is easier said than done, and it took 15 years to construct the first anonymous divisible e-cash system [CG07]. Moreover, the latter was more a proof of concept than a practical scheme, as pointed out in [ASM08, CG10]. Although several improvements followed (*e.g.* [ASM08,CG10,CPST15a,PST17]), the resulting constructions are still rather complex, which makes their analysis difficult. We highlight this issue by pointing out in this paper a problem that has been overlooked in the security proofs of these constructions. We cannot provide too many details in this introduction (we refer to Section 6 for a more extensive description of this problem) but informally this problem stems from the fact that those security proofs do not consider some strategies the adversary could use to frame honest users. More specifically, they assume that all transactions accusing an honest user must have been forged or generated with his key, which is not necessarily true due to the complex identification procedure of e-cash systems. The impact of this problem differs from a scheme to another. As explained in Section 6, it is possible to patch the security proofs of some papers without modifying the construction. However, this is not always true and some systems are likely to need important modifications to remain provably secure.

1.2 Our Contribution

We believe that the difficulty of constructing divisible e-cash systems is mainly due to the lack of generic frameworks for this primitive. This indeed forces designers of such systems to create and combine several ad-hoc mechanisms that

[2] several efficient protocols exist, such as the ones described in [CL04, PS16].

[3] The terminology can be confusing here: the "divisible coin" considered by most of the papers corresponds to the "wallet" of a compact e-cash system. In particular, the divisible coin contains several coins that are all associated to a serial number.

can hardly be reproduced for different settings. This also leads to complex security proofs that often rely on tailored computational assumptions, which may reduce confidence in the overall construction. This stands in sharp contrast with a related primitive, group signature, whose foundations were studied by Bellare *et al* [BMW03, BSZ05] and for which very efficient constructions exist.

In this work, we propose two generic frameworks that yield secure divisible e-cash systems from well-known cryptographic primitives. For each of the latter, we identify the properties they must achieve and so reduce the problem of constructing divisible e-cash systems to the simpler one of efficiently instantiating each of the building blocks. We additionally provide examples of instantiations to show that our frameworks are not artificial but can lead to practical schemes.

Starting from the work of Camenisch *et al* [CHL05] that defines the serial numbers as outputs of a PRF, we formalize the requirements on divisible e-cash systems as properties that must be achieved by the PRFs. Actually, the main requirement is that the serial numbers can be revealed by batches, which means that it must be possible to reveal some element $k_{\mathcal{S}}$ that (1) allows to compute $F_s(i)\ \forall i \in \mathcal{S} \subset [1, N]$ and (2) does not provide any information on the other serial numbers, *i.e.* on the outputs of the PRF outside $\mathcal{S}$. This exactly matches the definition of *constrained* PRF, a notion formalized in [BW13, KPTZ13, BGI14].

There are also several requirements that must implicitly be fulfilled by the *constrained key* $k_{\mathcal{S}}$. Different constrained keys generated from the same master key must indeed be unlinkable and $k_{\mathcal{S}}$ must hide some information on the subset $\mathcal{S}$, otherwise the resulting e-cash system could not be anonymous. All these notions were already defined in previous papers (*e.g.* [BFP+15,BLW17,BKM17]), although we only need here weaker versions of the original definitions.

Intuitively, all the properties listed above will ensure honest users' privacy. However, e-cash systems must also be able to deal with dishonest parties, potentially including the bank. In such a case, the adversary will have much more power than in usual PRF security games: it will have a total control on the seeds and could use it to create collisions between serial numbers or worse, falsely accuse an honest user. To thwart such attacks, we need to introduce a new security property for constrained PRF that we call collision resistance. It requires that different keys (even chosen by the adversary) yield different outputs, similarly to the standard collision resistance notion for hash functions. We provide more details in Section 2.2.

We then investigate two different scenarios, leading to two different (but related) frameworks. In the first one, we consider *homomorphic* constrained PRF [BFP+15] whereas we use *delegatable* constrained PRF [KPTZ13] in the second one. Interestingly, we note that all existing divisible e-cash systems can be associated with one of these frameworks, which brings two benefits. First, this means that it is possible to extract from existing systems constrained PRFs (either homomorphic or delegatable) that achieve all the properties we list above. We therefore believe that our results might be of independent interest outside e-cash since it draws attention on (implicit) constructions of constrained PRFs that might be ignored. Second, it means that some of the constructions affected by the problem mentioned at the end of Section 1.1 can easily be fixed by using the same tricks we introduce in our frameworks.

Once we have identified the sufficient properties for our PRFs, we explain how to use them to generically construct the serial numbers and the double spending tags, which is clearly the main difficulty of our paper. We then describe how to combine these PRFs with very standard primitives, namely digital signatures, commitment schemes and NIZK proofs, to get a divisible e-cash system.

Finally, we provide detailed proofs for each of our frameworks to show that the security of the overall construction generically holds under the security of each of the building blocks. Concretely, this means that, for any setting, one can construct a secure divisible e-cash system by essentially designing a constrained PRF achieving some simple properties.

1.3 Organisation

We recall in Section 2.1 the notion of constrained pseudo-random functions and describe in Section 2.2 the security properties that we need to construct divisible e-cash systems. Concrete instantiations of constrained PRFs achieving these properties can be found in Appendix E. The syntax and the security model of divisible e-cash are described in Section 3. We provide in Section 4 the intuition behind our two frameworks, however, due to space limitations, Section 5 only contains the formal description of our first framework, the second one being described in Appendix C. The security analysis of our generic constructions are respectively provided in Appendix A and Appendix D. Finally, we describe in Section 6 a problem in the security proofs of many divisible e-cash schemes.

2 Constrained Pseudo-Random Function

Our constructions of divisible e-cash systems will use constrained pseudo-random functions [BW13, KPTZ13, BGI14] with special features that we present below. But first, we recall the syntax of this primitive.

2.1 Syntax

For sake of simplicity, our PRF $\mathcal{K} \times \mathcal{S} \to \mathcal{Y}$ will only be constrained on subsets of $\mathcal{S}$. We will then not consider the more general setting where it is constrained according to a circuit. Our PRF thus consists of the following five algorithms.

– $\texttt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$: On input a security parameter λ and a set of admissible subsets $\mathcal{S}_i \subset \mathcal{S}$, this algorithm outputs the public parameters pp that will be implicitly taken as inputs by all the following algorithms.
– $\texttt{Keygen}()$: this algorithm outputs a master secret key $s \in \mathcal{K}$.
– $\texttt{Const}(s, \mathcal{X})$: On input the master key s and a set $\mathcal{X}$, this deterministic[4] algorithm outputs a constrained key $k_\mathcal{X} \in \mathcal{K}_\mathcal{X}$ or $\perp$.

[4] Although the general definition in [BW13] allows randomized Const algorithm, all our constructions will require this algorithm to be deterministic.

– $\texttt{Eval}(s, x)$: On input the master key s and an element $x \in \mathcal{S}$, this deterministic algorithm outputs a value $y \in \mathcal{Y}$.
– $\texttt{ConstEval}(\mathcal{X}, k_{\mathcal{X}}, x)$: On input a set $\mathcal{X}$, a constrained key $k_{\mathcal{X}}$ and an element $x \in \mathcal{S}$, this deterministic algorithm outputs a value $y \in \mathcal{Y}$.

In the following, we will always denote $\texttt{ConstEval}(\mathcal{X}, k_{\mathcal{X}}, x)$ by $\texttt{ConstEval}_{\mathcal{X}}(k_{\mathcal{X}}, x)$.

A constrained PRF is *correct* for a family of subsets $\{\mathcal{S}_i\}_{i=1}^n$ if, for all $\lambda \in \mathbb{N}$, $pp \leftarrow \texttt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$, $s \leftarrow \texttt{Keygen}()$ and $x \in \mathcal{S}$, we have:

$$\texttt{ConstEval}_{\mathcal{S}_i}(\texttt{Const}(s, \mathcal{S}_i), x) = \texttt{Eval}(s, x), \forall \mathcal{S}_i \ni x$$

with overwhelming probability.

Definition 1. *A constrained PRF is key homomorphic [BLMR13, BFP⁺15] if:*

1. $\mathcal{Y}$*,* $\mathcal{K}$ *and* $\mathcal{K}_{\mathcal{S}_i}$ *are groups* $\forall i \in [1, n]$
2. $\forall i \in [1, n]$, $\texttt{ConstEval}_{\mathcal{S}_i}(k_1 \cdot k_2, x) = \texttt{ConstEval}_{\mathcal{S}_i}(k_1, x) \cdot \texttt{ConstEval}_{\mathcal{S}_i}(k_2, x)$, $\forall k_1, k_2 \in \mathcal{K}_{\mathcal{S}_i}$ *and* $x \in \mathcal{S}_i$*.*
3. $\texttt{Const}(s_1 \cdot s_2, \mathcal{S}_i) = \texttt{Const}(s_1, \mathcal{S}_i) \cdot \texttt{Const}(s_2, \mathcal{S}_i)$, $\forall s_1, s_2 \in \mathcal{K}$ *and* $i \in [1, n]$

We use the multiplicative notation for our group operation. As in [BFP⁺15], we require that the $\texttt{Const}$ algorithm be a group homomorphism.

Finally, some of our constructions will require the ability to derive a constrained key $k_{\mathcal{S}_i}$ from any key $k_{\mathcal{S}_j}$ such that $\mathcal{S}_i \subset \mathcal{S}_j$. This requires the following modifications of the syntax and of the correctness property.

Definition 2. *A constrained pseudo-random function is delegatable if it additionally supports the following algorithm:*

– $\overline{\texttt{Const}}(k_{\mathcal{X}}, \mathcal{X}')$: *on input a constrained key* $k_{\mathcal{X}} \in \mathcal{K}_{\mathcal{X}}$ *and a set* $\mathcal{X}'$*, this algorithm outputs a constrained key* $k_{\mathcal{X}'} \in \mathcal{K}_{\mathcal{X}'}$ *or* $\perp$*.*

A delegatable constrained PRF is *correct* for a family of subsets $\{\mathcal{S}_i\}_{i=1}^n$ if, for all $\lambda \in \mathbb{N}$,$pp \leftarrow \texttt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$, $s \leftarrow \texttt{Keygen}()$ and $x \in \mathcal{S}$, we have:

$$\texttt{ConstEval}_{\mathcal{S}_i}(\texttt{Const}(s, \mathcal{S}_i), x) = \texttt{Eval}(s, x), \forall \mathcal{S}_i \ni x$$

and

$$\overline{\texttt{Const}}(k_{\mathcal{S}_j}, \mathcal{S}_i) = \texttt{Const}(s, \mathcal{S}_i), \forall \mathcal{S}_j \supset \mathcal{S}_i$$

with overwhelming probability.

2.2 Security Model

Our divisible e-cash constructions will use different types of constrained PRF, satisfying some of the following security requirements. Most of them have already been defined in previous works but we will need specific variants for some of them.

$\mathtt{Exp}_{\mathcal{A}}^{ps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ – Pseudorandomness

```
1. $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$
2. $s \leftarrow \mathtt{Keygen}()$
3. $x \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const},\mathcal{O}\mathtt{Eval}}(pp)$
4. $y_0 \leftarrow \mathtt{Eval}(s, x)$
5. $y_1 \overset{\$}{\leftarrow} \mathcal{Y}$
6. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const},\mathcal{O}\mathtt{Eval}}(pp, y_b)$
7. If $\mathcal{O}\mathtt{Eval}$ was queried on $x$ return 0
8. If $\mathcal{O}\mathtt{Const}$ was queried on $\mathcal{X} \ni x$, return 0
9. Return $b^*$
```

$\mathtt{Exp}_{\mathcal{A}}^{kps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ – Key-Pseudorandomness

```
1. $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$
2. $s \leftarrow \mathtt{Keygen}()$
3. $i^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const},\mathcal{O}\mathtt{Eval}}(pp)$
4. $k_0 \leftarrow \mathtt{Const}(s, \mathcal{S}_{i^*})$
5. $k_1 \overset{\$}{\leftarrow} \mathcal{K}_{\mathcal{S}_{i^*}}$
6. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const},\mathcal{O}\mathtt{Eval}}(pp, k_b)$
7. If $\mathcal{O}\mathtt{Eval}$ was queried on $x \in \mathcal{S}_{i^*}$ return 0
8. If $\mathcal{O}\mathtt{Const}$ was queried on $\mathcal{X}$ such that $\mathcal{X} \cap \mathcal{S}_{i^*} \neq \emptyset$, return 0
9. Return $b^*$
```

$\mathtt{Exp}_{\mathcal{A}}^{ckps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ – Combined Key-Pseudorandomness

```
1. $pp_j \leftarrow F_j.\mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n), \forall j \in [1, t]$
2. $s \leftarrow F_1.\mathtt{Keygen}()$
3. $i^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const},\mathcal{O}\mathtt{Eval}}(\{pp_j\}_{j=1}^t)$
4. $(k_0^1, \ldots, k_0^t) \leftarrow (F_1.\mathtt{Const}(s, \mathcal{S}_{i^*}), \ldots, F_t.\mathtt{Const}(s, \mathcal{S}_{i^*}))$
5. $(k_1^1, \ldots, k_1^t) \overset{\$}{\leftarrow} \mathcal{K}_{\mathcal{S}_{i^*}}^t$
6. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const},\mathcal{O}\mathtt{Eval}}(\{pp_j\}_{j=1}^t, (k_b^1, \ldots, k_b^t))$
7. If $\mathcal{O}\mathtt{Eval}$ was queried on $x \in \mathcal{S}_{i^*}$ return 0
8. If $\mathcal{O}\mathtt{Const}$ was queried on $\mathcal{X}$ such that $\mathcal{X} \cap \mathcal{S}_{i^*}$, return 0
9. Return $b^*$
```

Fig. 1. Pseudorandomness Games for Constrained Pseudo-Random Functions

Pseudorandomness. The first property one may expect from a constrained PRF is *pseudorandomness*, which informally requires that an adversary, even given access to constrained keys, cannot distinguish the PRF evaluation at the points it is not allowed to compute. It is defined by $\mathtt{Exp}_{\mathcal{A}}^{ps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ in Figure 1 where the adversary has access to the following oracles:

– $\mathcal{O}\mathtt{Const}(\mathcal{X})$: on input a set $\mathcal{X}$, this algorithm returns $\mathtt{Const}(s, \mathcal{X})$ if $\exists i \in [1, n]$ such that $\mathcal{X} = \mathcal{S}_i$ and $\perp$ otherwise.
– $\mathcal{O}\mathtt{Eval}(x)$: on input an element $x \in \mathcal{S}$, this algorithm returns $\mathtt{Eval}(s, x)$.

A constrained PRF is *pseudorandom* if $\mathtt{Adv}^{ps}(\mathcal{A}) = |\Pr[\mathtt{Exp}_{\mathcal{A}}^{ps-1}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]$ - $\Pr[\mathtt{Exp}_{\mathcal{A}}^{ps-0}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]|$ is negligible for any $\mathcal{A}$.

Key-Pseudorandomness. We note that the previous definition only requires pseudorandomness for the output of the PRF. As in [BFP+15] we extend this property to the constrained keys themselves, leading to a property that we call key-pseudorandomness. However, compared to [BFP+15], we additionally require some form of key privacy, in the sense of [KPTZ13]. In particular, we need that constrained keys issued for subsets of the same size[5] should be indistinguishable.

Let F be a constrained PRF defined for a family of subsets $\{\mathcal{S}_i\}_{i=1}^n$ verifying $\mathcal{K}_{\mathcal{S}_i} = \mathcal{K}_{\mathcal{S}_j}$ $\forall i, j$ such that $|\mathcal{S}_i| = |\mathcal{S}_j|$. F is *key-pseudorandom* if $\mathtt{Adv}^{kps}(\mathcal{A}) = |\Pr[\mathtt{Exp}_{\mathcal{A}}^{kps-1}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]$ - $\Pr[\mathtt{Exp}_{\mathcal{A}}^{kps-0}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]|$ is negligible for any $\mathcal{A}$, where the game $\mathtt{Exp}_{\mathcal{A}}^{kps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ is defined in Figure 1.

Combined Key-Pseudorandomness. In practice, divisible e-cash systems will require several pseudorandom values, some acting as the unique identifier of the coin (the serial number) and some being used to mask the spender's identity. If F is key-pseudorandom, a solution could be to split $k_{\mathcal{S}_i} \leftarrow \mathtt{Const}(s, \mathcal{S}_i)$ into several parts, each of them being used as a pseudorandom value. Unfortunately, combining this solution with zero-knowledge proofs would be very complex. In our frameworks, we will follow a different approach and will generate several pseudorandom values by using different PRFs $F_1, \ldots, F_t$ evaluated on the same master key s and the same subset $\mathcal{S}_i$. This can be done very easily by constructing each F_i similarly but with different public parameters.

For example, let us assume that $F_1.\mathtt{Const}(s, \mathcal{S}_i) = k^1_{\mathcal{S}_i} = g_1^{\alpha_i \cdot s}$ $\forall i$ for some generator g_1 of $\mathcal{K}_{\mathcal{S}_i}$. We can define other PRFs $F_2, \ldots F_t$ with the same input spaces by setting $F_j.\mathtt{Const}(s, \mathcal{S}_i) = k^j_{\mathcal{S}_i} = g_j^{\alpha_i \cdot s}$ for a different generator g_j. In such a case, we get t values $(k^1_{\mathcal{S}_i}, \ldots, k^t_{\mathcal{S}_i})$ which are indistinguishable from a random element of $\mathcal{K}^t_{\mathcal{S}_i}$ assuming key-pseudorandomness of F_1 and the DDH assumption (see Appendix E for more details).

This example shows that we can easily get several pseudorandom values without introducing new seeds or new subsets (this will help us to limit the complexity of our system) but also that, in such a case, we cannot simply rely on the key-pseudorandomness of F. We then need to extend this notion to cover this scenario, leading to the following definition of combined key-pseudorandomness.

Let $F_1 \ldots, F_t$ be constrained PRFs $\mathcal{K} \times \mathcal{S} \rightarrow \mathcal{Y}$ defined for the same family of subsets $\{\mathcal{S}_i\}_{i=1}^n$ verifying $\mathcal{K}_{\mathcal{S}_i} = \mathcal{K}_{\mathcal{S}_j}$ $\forall i, j$ such that $|\mathcal{S}_i| = |\mathcal{S}_j|$. We say that the family $(F_1, \ldots, F_t)$ achieves combined key-pseudorandomness if $\mathtt{Adv}^{ckps}(\mathcal{A}) = |\Pr[\mathtt{Exp}_{\mathcal{A}}^{ckps-1}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]$ - $\Pr[\mathtt{Exp}_{\mathcal{A}}^{ckps-0}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]|$ is negligible for any $\mathcal{A}$, where the game $\mathtt{Exp}_{\mathcal{A}}^{ckps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ is defined in Figure 1.

Collision Resistance. In our divisible e-cash constructions, the PRFs will mostly be used to generate serial numbers that act as unique identifiers of the coins. If a coin is spent twice (or more) the same serial number will appear in

[5] We note that our privacy requirements are weaker than the ones of [BLW17,BKM17] since we allow the constrained keys to leak the size of the subsets.

```
$\mathtt{Exp}^{col-1}_{\mathcal{A}}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ – Collision Resistance 1
  1. $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$
  2. $(s_1, s_2, x_1, x_2) \leftarrow \mathcal{A}(pp)$
  3. Return $\mathtt{Eval}(s_1, x_1) = \mathtt{Eval}(s_2, x_2) \wedge (s_1, x_1) \neq (s_2, x_2)$

$\mathtt{Exp}^{col-2}_{\mathcal{A}}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ – Collision Resistance 2
  1. $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$
  2. $(i, k_1, k_2, x) \leftarrow \mathcal{A}(pp)$
  3. Return $\mathtt{ConstEval}_{\mathcal{S}_i}(k_1, x) = \mathtt{ConstEval}_{\mathcal{S}_i}(k_2, x) \wedge k_1 \neq k_2$

$\mathtt{Exp}^{col-3}_{\mathcal{A}}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ – Collision Resistance 3
  1. $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$
  2. $(i, j, k, x) \leftarrow \mathcal{A}(pp)$
  3. Return $\mathtt{ConstEval}_{\mathcal{S}_i}(k, x) = \mathtt{ConstEval}_{\mathcal{S}_j}(k, x) \wedge i \neq j \wedge k \neq 1$
```

Fig. 2. Collision Resistance Games for Constrained Pseudo-Random Functions

several transactions, which provides a very simple way to detect frauds. However, it is important to ensure that collisions between serial numbers only occur in such cases. Otherwise, this could lead to false alerts and even false accusations against an honest user.

At first sight, it might seem that this property is implied by pseudorandomness. Unfortunately, this is not true in the context of e-cash where the adversary has total control of the master secret keys, contrarily to the adversary of the pseudorandomness game. We therefore need to define a new property that we call collision resistance. Informally, it says that it should be hard to generate collisions between the outputs of the PRFs. However, some subtleties arise because of the different kinds of keys (secret master keys, constrained keys) that we consider here. We then define three variants of this property that are described in Figure 2.

For $k \in \{1, 2, 3\}$, a constrained PRF achieves *collision resistance-k* if, for any $\mathcal{A}$, $\mathtt{Adv}^{col-k}(\mathcal{A}) = \Pr[\mathtt{Exp}^{col-k}_{\mathcal{A}}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]$ is negligible. We provide in Appendix E several examples of PRFs achieving these properties.

3 Divisible E-Cash

The syntax and the formal security model are drawn from [CPST15a, PST17]. We nevertheless introduce several changes to make it more generic but also to add some specifications that were previously implicit.

3.1 Syntax

A divisible e-cash system is defined by the following algorithms, that involve three types of entities, the bank $\mathcal{B}$, a user $\mathcal{U}$ and a merchant $\mathcal{M}$. Our model defines a unique value N for the divisible coin but it can easily be extended to support several different denominations.

- $\texttt{Setup}(1^\lambda, N)$: On input a security parameter λ and an integer N, this probabilistic algorithm outputs the public parameters pp for divisible coins of global value N. We assume that pp are implicit to the other algorithms, and that they include k and N. They are also an implicit input to the adversary, we will then omit them.
- $\texttt{BKeygen}()$: This probabilistic algorithm executed by the bank $\mathcal{B}$ outputs a key pair $(\mathsf{bsk}, \mathsf{bpk})$. It also sets L as an empty list, that will store all deposited coins. We assume that bsk contains bpk.
- $\texttt{Keygen}()$: This probabilistic algorithm executed by a user $\mathcal{U}$ (resp. a merchant $\mathcal{M}$) outputs a key pair $(\mathsf{usk}, \mathsf{upk})$ (resp. $(\mathsf{msk}, \mathsf{mpk})$). We assume that usk (resp. msk) contains upk (resp. mpk).
- $\texttt{Withdraw}(\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk}))$: This is an interactive protocol between the bank $\mathcal{B}$ and a user $\mathcal{U}$. At the end of this protocol, the user gets a divisible coin C of value N or outputs $\perp$ (in case of failure) while the bank stores the transcript of the protocol execution or outputs $\perp$.
- $\texttt{Spend}(\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, V), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, \mathsf{info}, V))$: This is an interactive protocol between a user $\mathcal{U}$ and a merchant $\mathcal{M}$. Here, info denotes a set of public information associated by the merchant to the transaction and V denotes the amount of this transaction. At the end of the protocol the merchant gets Z along with a proof of validity Π or outputs $\perp$. $\mathcal{U}$ then either updates C or outputs $\perp$.
- $\texttt{Deposit}(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (V, \mathsf{info}, Z, \Pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: This is an interactive protocol between a merchant $\mathcal{M}$ and the bank $\mathcal{B}$ where the former first sends a transcript $(V, \mathsf{info}, Z, \Pi)$ along with some additional data μ. $\mathcal{B}$ then checks (1) the validity of all these elements and (2) that this merchant has not already deposited a transcript associated with info. If condition (1) is not fulfilled, then $\mathcal{B}$ aborts and outputs $\perp$. If condition (2) is not fulfilled, then $\mathcal{B}$ returns another transcript $(V', \mathsf{info}, Z', \Pi')$ along with the associated μ'. Otherwise, $\mathcal{B}$ recovers the V serial numbers $\texttt{SN}_{i_0}, \ldots, \texttt{SN}_{i_{V-1}}$[6] derived from Z and compares them to the set L of all serial numbers of previously spent coins. If there is a match for some index i_k, then $\mathcal{B}$ returns a transcript $(V', Z', \Pi', \mathsf{info}')$ such that $\texttt{SN}_{i_k}$ is also a serial number derived from Z'. Else, $\mathcal{B}$ stores these new serial numbers in L and keeps a copy of $(V, \mathsf{info}, \mathsf{mpk}, Z, \Pi)$.
- $\texttt{Identify}((V, \mathsf{info}, \mathsf{mpk}, Z, \Pi), (V', \mathsf{info}', \mathsf{mpk}', Z', \Pi'), \mathsf{bpk})$: On inputs two transcripts, this deterministic algorithm outputs 0 if $\mathsf{info} = \mathsf{info}'$, if one of the transcripts is invalid, or if the serial numbers derived from these transcripts do not collide. Else it outputs a user's public key upk or $\perp$.
- $\texttt{CheckDeposit}([(V, \mathsf{info}, \mathsf{mpk}, Z, \Pi), \mu], \mathsf{bpk})$: This deterministic algorithm outputs 1 if $[(V, Z, \Pi, \mathsf{info}), \mu]$ are valid elements deposited by a merchant whose public key is mpk and 0 otherwise.

[6] We note that previous divisible e-cash systems assume that all the indices $i_0, \ldots, i_{V-1}$ belong to the same interval $\mathcal{I}$ of size V. However, there are some cases where the spender may not be able to reveal consecutive serial numbers. We therefore make no assumption on the indices to keep our model as general as possible.

Remark 3. We note that several important requirements on the value info or on the Deposit procedure were only informally stated or even implicit in previous works. We therefore define in the next section a new security property, called *clearing*, that will formalize these requirements. This game will require the CheckDeposit algorithm that does not exist in previous models.

Remark 4. Our model does not place restrictions on the values that can be spent nor on the size of a spending transcript. It is therefore more generic and in particular also fits compact e-cash systems where the serial numbers can only be revealed one by one.

3.2 Security Model

Existing security models essentially focus on the the user's and the bank's interests. The former must indeed be able to spend their coins anonymously without being falsely accused of frauds while the latter must be able to detect frauds and identify the perpetrators. This is formally defined by three security properties in [CPST15a]: *anonymity*, *exculpability* and *traceability*.

However, all these notions (and the corresponding ones in previous papers) fail to capture an important security property for the merchant, namely the fact that he must always be able to clear his transactions and in particular that he should be the only one able to deposit them. This is especially problematic for e-cash because users can reproduce the transcripts of their spendings. Designers of existing divisible e-cash systems seem to be more or less aware of this issue[7] because they usually attribute a signing key to the merchant. However, these systems do not specify the security properties expected from the corresponding signature scheme and most of them even do not specify which elements should be signed.

For completeness, we therefore add the property of *clearing* to the ones of *traceability*, *anonymity* and *exculpability* descibed in [CPST15a, PST17]. All of them are defined in Figure 3 and make use of the following oracles.

– $\mathcal{O}$Add() is an oracle used by the adversary $\mathcal{A}$ to register a new honest user (resp. merchant). The oracle runs the Keygen algorithm, stores usk (resp. msk) and returns upk (resp. mpk) to $\mathcal{A}$. In this case, upk (resp. mpk) is said *honest*.
– $\mathcal{O}$Corrupt(upk/mpk) is an oracle used by $\mathcal{A}$ to corrupt an honest user (resp. merchant) whose public key is upk (resp. mpk). The oracle then returns the corresponding secret key usk (resp. msk) to $\mathcal{A}$ along with the secret values of every coin withdrawn by this user. From now on, upk (resp. mpk) is said *corrupted*.

[7] The "correctness for merchant", informally defined in [ASM08], is related to this issue. It ensures that the transcript deposited by an honest merchant will be accepted, even if the spender is dishonest and double-spends his coin. However, it only considers an honest bank and it does not consider situations where the transcript would be deposited by another entity. In particular, the scheme in [ASM08] does not ensure that the merchant is the only one able to clear his coins.

```
$\mathtt{Exp}_{\mathcal{A}}^{tra}(1^\lambda, N)$ – Traceability Security Game
 1. $pp \leftarrow \mathtt{Setup}(1^\lambda, N)$
 2. $(\mathsf{bsk}, \mathsf{bpk}) \leftarrow \mathtt{BKeygen}()$
 3. $\{(V_i, \mathsf{info}_i, \mathsf{mpk}_i, Z_i, \Pi_i)\}_{i=1}^{u} \xleftarrow{\$} \mathcal{A}^{\mathcal{O}\mathtt{Add},\mathcal{O}\mathtt{Corrupt},\mathcal{O}\mathtt{AddCorrupt},\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}},\mathcal{O}\mathtt{Spend}}(\mathsf{bpk})$
 4. If $\sum_{i=1}^{u} V_i > m \cdot N$ and $\forall i \neq j$, $\mathtt{Identify}(\mathsf{Tr}_i, \mathsf{Tr}_j, \mathsf{bpk}) = \perp$, then return 1
 5. Return 0

$\mathtt{Exp}_{\mathcal{A}}^{excu}(1^\lambda, N)$ – Exculpability Security Game
 1. $pp \leftarrow \mathtt{Setup}(1^\lambda, N)$
 2. $\mathsf{bpk} \leftarrow \mathcal{A}()$
 3. $[\mathsf{Tr}_1, \mathsf{Tr}_2] \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Add},\mathcal{O}\mathtt{Corrupt},\mathcal{O}\mathtt{AddCorrupt},\mathcal{O}\mathtt{Withdraw}_{\mathcal{U}},\mathcal{O}\mathtt{Spend}}()$
 4. If $\mathtt{Identify}(\mathsf{Tr}_1, \mathsf{Tr}_2, \mathsf{bpk}) = \mathsf{upk}$ and $\mathsf{upk}$ not corrupted, then return 1
 5. Return 0

$\mathtt{Exp}_{\mathcal{A}}^{anon-b}(1^\lambda, N)$ – Anonymity Security Game
 1. $pp \leftarrow \mathtt{Setup}(1^\lambda, N)$
 2. $\mathsf{bpk} \leftarrow \mathcal{A}()$
 3. $(V, \mathsf{upk}_0, \mathsf{upk}_1, \mathsf{mpk}) \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Add},\mathcal{O}\mathtt{Corrupt},\mathcal{O}\mathtt{AddCorrupt},\mathcal{O}\mathtt{Withdraw}_{\mathcal{U}},\mathcal{O}\mathtt{Spend}}()$
 4. If $\mathsf{upk}_i$ is not registered for $i \in \{0,1\}$, then return 0
 5. If $c_{\mathsf{upk}_i} > m_{\mathsf{upk}_i} \cdot N - V$ for $i \in \{0,1\}$, then return 0
 6. $(V, Z, \Pi, \mathsf{info}) \leftarrow \mathtt{Spend}(\mathcal{C}(\mathsf{usk}_b, C, \mathsf{mpk}, V), \mathcal{A}())$
 7. $c_{\mathsf{upk}_{1-b}} \leftarrow c_{\mathsf{upk}_{1-b}} + V$
 8. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Add},\mathcal{O}\mathtt{Corrupt},\mathcal{O}\mathtt{AddCorrupt},\mathcal{O}\mathtt{Withdraw}_{\mathcal{U}},\mathcal{O}\mathtt{Spend}}()$
 9. If $\mathsf{upk}_i$ has been corrupted for $i \in \{0,1\}$, then return 0
10. Return $(b = b^*)$

$\mathtt{Exp}_{\mathcal{A}}^{clear}(1^\lambda, N)$ – Clearing Security Game
 1. $pp \leftarrow \mathtt{Setup}(1^\lambda, N)$
 2. $\mathsf{bpk} \leftarrow \mathcal{A}()$
 3. $[(V, \mathsf{info}, \mathsf{mpk}, Z, \Pi), \mu] \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Add},\mathcal{O}\mathtt{Corrupt},\mathcal{O}\mathtt{AddCorrupt},\mathcal{O}\mathtt{Receive},\mathcal{O}\mathtt{Deposit}}()$
 4. If $\mathtt{CheckDeposit}([(V, \mathsf{info}, \mathsf{mpk}, Z, \Pi), \mu], \mathsf{bpk}) = 0$, then return 0
 5. If $\mathsf{mpk}$ is corrupted, then return 0
 6. If $(\mathsf{mpk}, V, \mathsf{info})$ has been queried to $\mathcal{O}\mathtt{Deposit}$ then return 0
 7. Return 1
```

Fig. 3. Security Games for Divisible E-Cash.

– $\mathcal{O}\mathtt{AddCorrupt}(\mathsf{upk}/\mathsf{mpk})$ is an oracle used by $\mathcal{A}$ to register a new corrupted user (resp. merchant) whose public key is upk (resp. mpk). In this case, upk (resp. mpk) is said *corrupted*. The adversary could use this oracle on a public key already registered (during a previous $\mathcal{O}\mathtt{Add}$ query) but for simplicity, we do not consider such case as it will gain nothing more than using the $\mathcal{O}\mathtt{Corrupt}$ oracle on the same public key.
– $\mathcal{O}\mathtt{Withdraw}_{\mathcal{U}}(\mathsf{upk})$ is an oracle that executes the user's side of the $\mathtt{Withdraw}$ protocol. This oracle will be used by $\mathcal{A}$ playing the role of the bank against the user with public key upk.
– $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}(\mathsf{upk})$ is an oracle that executes the bank's side of the $\mathtt{Withdraw}$ protocol. This oracle will be used by $\mathcal{A}$ playing the role of a user whose public key is upk against the bank.

– $\mathcal{O}\texttt{Spend}(\mathsf{upk}, V)$ is an oracle that executes the user's side of the $\texttt{Spend}$ protocol for a value V. This oracle will be used by $\mathcal{A}$ playing the role of the merchant $\mathcal{M}$.
– $\mathcal{O}\texttt{Receive}(\mathsf{mpk}, V)$ is an oracle that executes the merchant's side of the $\texttt{Spend}$ protocol for a value V. This oracle will be used by $\mathcal{A}$ playing the role of a user.
– $\mathcal{O}\texttt{Deposit}(\mathsf{mpk}, V, \mathsf{info})$ is an oracle that executes the merchant's side of the $\texttt{Deposit}$ protocol for a transaction of amount V associated with the value info. This oracle cannot be queried on two inputs with the same value info. It will be used by $\mathcal{A}$ playing the role of the bank.

In the experiments, users are denoted by their public keys upk, c_{upk} denotes the amount already spent by user upk during $\mathcal{O}\texttt{Spend}$ queries, m_{upk} the number of divisible coins that he has withdrawn and Tr_i the tuple $(V_i, \mathsf{info}_i, \mathsf{mpk}_i, Z_i, \Pi_i)$ for any $i \in \mathbb{N}$. This means that the total amount available by a user upk is $m_{\mathsf{upk}} \cdot N$. The number of coins withdrawn by all users during an experiment is denoted by m.

For sake of simplicity, we assume that all merchants are corrupted, and added through $\mathcal{O}\texttt{AddCorrupt}$ queries, in the traceability, exculpability and anonymity experiments. We therefore do not need to add access to the $\mathcal{O}\texttt{Receive}$ and $\mathcal{O}\texttt{Deposit}$ oracles in the latter. We stress that this is not a restriction since the $\mathcal{O}\texttt{AddCorrupt}$ oracle provides more power to the adversary than the $\mathcal{O}\texttt{Add}$ and $\mathcal{O}\texttt{Corrupt}$ ones. Similarly, we assume that the bank and all the users are corrupted in the clearing game and so do not provide access to the $\mathcal{O}\texttt{Spend}$, $\mathcal{O}\texttt{Withdraw}_{\mathcal{U}}$ and $\mathcal{O}\texttt{Withdraw}_{\mathcal{B}}$ oracles in it.

Our clearing game ensures that no one can forge a valid deposit query from the merchant. This means in particular that the bank cannot rightfully refuse the deposit of an honest merchant (because it will not be able to provide a valid proof that the transcript has already been deposited) and that it cannot falsely accuse a merchant of trying to deposit the same transcript several times.

Let $\mathcal{A}$ be a probabilistic polynomial adversary. A divisible E-cash system is:

– *traceable* if $\texttt{Succ}^{tra}(\mathcal{A}) = \Pr[\texttt{Exp}_{\mathcal{A}}^{tra}(1^\lambda, N) = 1]$ is negligible for any $\mathcal{A}$;
– *exculpable* if $\texttt{Succ}^{excu}(\mathcal{A}) = \Pr[\texttt{Exp}_{\mathcal{A}}^{excu}(1^\lambda, N) = 1]$ is negligible for any $\mathcal{A}$;
– *anonymous* if $\texttt{Adv}^{anon}(\mathcal{A}) = |\Pr[\texttt{Exp}_{\mathcal{A}}^{anon-1}(1^\lambda, N) = 1]$ - $\Pr[\texttt{Exp}_{\mathcal{A}}^{anon-0}(1^\lambda, N) = 1]|$ is negligible for any $\mathcal{A}$.
– *clearable* if $\texttt{Succ}^{clear}(\mathcal{A}) = \Pr[\texttt{Exp}_{\mathcal{A}}^{clear}(1^\lambda, N) = 1]$ is negligible for any $\mathcal{A}$;

4 High-Level Description

Before introducing a generic framework for divisible e-cash, we focus on the heart of such systems, namely the construction of the serial numbers and of the double-spending tags.

Regarding the former, the fact that each serial number $\texttt{SN}$ must look random has led designers to use pseudo-random functions (PRF) to generate them. More specifically, every anonymous divisible e-cash scheme defines $\texttt{SN}_i$ as $F.\texttt{Eval}(s, i)$

where s is the master key and $i \in [1, N]$. However, to avoid a cost linear in the amount V it is necessary to provide a way to reveal these serial numbers by batches. Designers of divisible e-cash systems (*e.g.* [CG07, ASM08, CG10, CPST15a, CPST15b, PST17]) have thus constructed pseudo-random functions with a special feature: given s and a set $\mathcal{S}$, one can compute $k_{\mathcal{S}}$ allowing to evaluate the PRF only on the elements of $\mathcal{S} \subset [1, N]$. This matches the definition of constrained PRF that were later formalized in [BW13, KPTZ13, BGI14].

To spend a value V, the user can now simply reveal a constrained key $k_{\mathcal{S}}$ for a set $\mathcal{S}$ of size V. However additional properties are required here to achieve anonymity. Indeed, informally, the constrained key must hide information on the spender (more specifically on the master secret key) and on the subset $\mathcal{S}$[8] itself. All these properties are captured by key-pseudorandomness that we define in Section 2.

Finally, to avoid false positive in the detection process, we will need the collision resistance properties defined in the same section.

Therefore, constructing divisible e-cash with efficient double-spending detections is roughly equivalent to constructing a key-pseudorandom, collision resistant CPRF for subsets of $[1, N]$ that smoothly interacts with NIZK. However, detection of double spending is not enough, it must also be possible to identify double spenders by using the additional information contained in the double-spending tag. This adds further requirements on the PRF and leads to two constructions that we present below.

4.1 Construction using Key-Homomorphism

Our first construction of double-spending tag is reminiscent of the techniques used by compact e-cash systems [CHL05, LLNW17]. In these papers, the double spending tag T_i associated with $\mathtt{SN}_i$ is of the form $ID \cdot (F'.\mathtt{Eval}(s', i))^R$, where ID is the "identity" of the spender (usually his public key), F' is a PRF seeded with a master secret key s' (note that we may have $F = F'$ or $s = s'$ but not both) and R is a public identifier of the transaction.

Intuitively, the idea behind this tag is that $(F'.\mathtt{Eval}(s', i))^R$ will perfectly mask the user's identity as long as the latter does not overspend his coin. In case of double spendings, there will indeed be two tags $T_i^{(1)}$ and $T_i^{(2)}$ of the form $ID \cdot (F'.\mathtt{Eval}(s', i))^{R_1}$ and $ID \cdot (F'.\mathtt{Eval}(s', i))^{R_2}$. Therefore, by computing:

$$((T_i^{(1)})^{R_2} / (T_i^{(2)})^{R_1})^{1/(R_2 - R_1)}$$

the bank can directly recover the identity ID of the defrauder. This idea was adapted in [CPST15a, CPST15b, PST17] to the context of divisible e-cash by replacing $F'.\mathtt{Eval}(s', i)$ with a key constrained to the appropriate subset.

However, we note that this process of identification may be problematic to ensure exculpability, and that this point has been overlooked in the security

[8] Actually the size of $\mathcal{S}$ can leak since it corresponds to the public amount of the transaction.

proofs of previous works. We provide more details on the problem and on the affected papers in Section 6, but informally it arises from the fact that the formula above may output ID while involving tags $T_i^{(1)}$ and $T_i^{(2)}$ produced for different identities. Indeed, in the exculpability game, a malicious bank could cooperate with malicious users and merchants to select values ID_1, ID_2, R_1, R_2, s_1 and s_2 such that $((T_i^{(1)})^{R_2}/(T_i^{(2)})^{R_1})^{1/(R_2-R_1)} = ((ID_1 \cdot (F'.\texttt{Eval}(s_1, i_1))^{R_1})^{R_2}/(ID_2 \cdot (F'.\texttt{Eval}(s_2, i_2))^{R_2})^{R_1})^{1/(R_2-R_1)} = ID$. Unfortunately, this scenario has not been taken into account in previous papers, which invalidates many exculpability proofs. Worse, we describe in section 6 some situations were such proofs cannot be fixed unless modifying the protocol. This means that, in general, this tag construction cannot be used as it is.

To prevent this problem, our generic construction uses four PRF, that we will denote by F_1, F_2, F_3 and F_4, defined for the same family of subsets $\{\mathcal{S}_i\}_{i=1}^n$ and sharing the same key space $\mathcal{K}$. We additionally require F_2, F_3 and F_4 to be key-homomorphic.

Let $s \in \mathcal{K}$ be a secret master key and $\mathcal{S}_i$ be a subset of size V, the amount of the transaction. As previously[9], our first PRF will be used to reveal $k_{\mathcal{S}_i}^{(1)} \leftarrow F_1.\texttt{Const}(s, \mathcal{S}_i)$. Likewise, our third PRF will be used to generate an element of the form[10] $ID^R \cdot k_{\mathcal{S}_i}^3$, with $k_{\mathcal{S}_i}^3 \leftarrow F_3.\texttt{Const}(s, \mathcal{S}_i)$. The novelty here is that these values will only constitute a part of the serial number and of the double spending tag. The other parts will be derived from $k_{\mathcal{S}_i}^{(2)} \leftarrow F_2.\texttt{Const}(s \cdot id, \mathcal{S}_i)$, where id is some element of $\mathcal{K}$ associated with the public identity ID, and from $ID \cdot k_{\mathcal{S}_i}^4$ where $k_{\mathcal{S}_i}^4 \leftarrow F_4.\texttt{Const}(s, \mathcal{S}_i)$.

More specifically,

$$\texttt{SN}_j = F_1.\texttt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(1)}, j) || F_2.\texttt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(2)}, j)$$

and

$$T_{\mathcal{S}_i} = (ID^R \cdot k_{\mathcal{S}_i}^3, ID \cdot k_{\mathcal{S}_i}^4)$$

Intuitively, the fact that the master secret key of F_2 depends on id will ensure that no collision can occur for different users, which thwarts the previous attack. Moreover, the first part of $\texttt{SN}_j$ still ensures that collisions can only occur for spendings involving the same master key, evaluated on the same element $j \in \mathcal{S}$. The last element of the double-spending tag has a more technical purpose that we will explain below.

Therefore, if two spendings with respective tags $T_{\mathcal{S}_{i_1}}^{(1)}$ and $T_{\mathcal{S}_{i_2}}^{(2)}$ lead to a collision, then we have:

[9] For sake of clarity, we assume here that the elements associated with the users' identity live in the right spaces. Our formal definition will make use of suitable maps to ensure this fact.

[10] We need to apply the exponent R on the identity itself instead of the constrained key to rely on the correctness of `ConstEval`, but the principle is the same.

$$T^{(1)}_{\mathcal{S}_{i_1}} = (ID^{R_1} \cdot k^3_{\mathcal{S}_{i_1}}, ID \cdot k^4_{\mathcal{S}_{i_1}})$$
$$T^{(2)}_{\mathcal{S}_{i_2}} = (ID^{R_2} \cdot k^3_{\mathcal{S}_{i_2}}, ID \cdot k^4_{\mathcal{S}_{i_2}})$$

with $j \in \mathcal{S}_{i_1} \cap \mathcal{S}_{i_2}$. If $\mathcal{S}_{i_1} = \mathcal{S}_{i_2} = \mathcal{S}_i$, we can compute:

$$F_3.\texttt{ConstEval}_{\mathcal{S}_i}(T^{(1)}_{\mathcal{S}_i}[1], j) = F_3.\texttt{ConstEval}_{\mathcal{S}_i}(ID^{R_1}, j) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_i}(k^3_{\mathcal{S}_i}, j)$$
$$F_3.\texttt{ConstEval}_{\mathcal{S}_i}(T^{(2)}_{\mathcal{S}_i}[1], j) = F_3.\texttt{ConstEval}_{\mathcal{S}_i}(ID^{R_2}, j) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_i}(k^3_{\mathcal{S}_i}, j)$$

Since $k^3_{\mathcal{S}_{i_1}}$ and $k^3_{\mathcal{S}_{i_2}}$ are derived from the same master key, correctness ensures that

$$F_3.\texttt{ConstEval}_{\mathcal{S}_i}(k^3_{\mathcal{S}_i}, j) = F_3.\texttt{ConstEval}_{\mathcal{S}_i}(k^3_{\mathcal{S}_i}, j).$$

Therefore:

$$F_3.\texttt{ConstEval}_{\mathcal{S}_i}(T^{(2)}_{\mathcal{S}_i}[1], j) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_i}(T^{(1)}_{\mathcal{S}_i}[1], j)^{-1}$$
$$= F_3.\texttt{ConstEval}_{\mathcal{S}_i}(ID^{R_2}, j) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_i}(ID^{-R_1}, j)$$

The bank can then perform an exhaustive search on the set of public identities $\{ID_i\}$ until it gets a match. Identification of defrauders is then possible with a linear cost in the number of users of the system.

Now in the case where $\mathcal{S}_{i_1} \neq \mathcal{S}_{i_2}$, we have, for any identity ID^*:

$$F_4.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(T^{(1)}_{\mathcal{S}_{i_1}}[2]/(ID^*), j)$$
$$= F_4.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(ID/(ID^*), j) \cdot F_4.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(k^4_{\mathcal{S}_{i_1}}, j)$$
$$F_4.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(T^{(1)}_{\mathcal{S}_{i_2}}[2]/(ID^*), j)$$
$$= F_4.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(ID/(ID^*), j) \cdot F_4.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(k^4_{\mathcal{S}_{i_2}}, j)$$

Here again, $F_4.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(k^4_{\mathcal{S}_{i_1}}, j) = F_4.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(k^4_{\mathcal{S}_{i_2}}, j)$, therefore:

$$F_4.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(T^{(1)}_{\mathcal{S}_{i_1}}[2]/(ID^*), j)/F_4.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(T^{(1)}_{\mathcal{S}_{i_2}}[2]/(ID^*), j)$$
$$= F_4.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(ID/(ID^*), j)/F_4.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(ID/(ID^*), j)$$

and one can easily identify the case where $ID^* = ID$ since this it is the only one where the right member equals to $1_{\mathcal{Y}}$ if F_4 achieves collision resistance-3.

Remark 5. The use of two elements in the double spending tag may seem surprising, in particular because the equality

$$F_3.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(T^{(2)}_{\mathcal{S}_{i_2}}[1], j) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(T^{(1)}_{\mathcal{S}_{i_2}}[1], j)^{-1}$$
$$= F_3.\texttt{ConstEval}_{\mathcal{S}_{i_1}}(ID^{R_2}, j) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_{i_2}}(ID^{-R_1}, j)$$

still holds for the right ID in the case where $\mathcal{S}_{i_1} \neq \mathcal{S}_{i_2}$. However, in this case, we cannot ensure that this equality *only* holds for ID, it might also work for other identities, leading to obvious identification issues.

4.2 Construction using Delegation

Our second construction is inspired by what has been the main framework for divisible e-cash for many years (*e.g.* [CG07, CG10, Mär15]). It makes use of a family of two delegatable PRFs (F_1, F_2) and two functions[11] (H, H') such that $H : \mathcal{K} \rightarrow \mathbb{G}$ and $H' : \mathcal{K}_{\mathcal{S}_i} \rightarrow \mathbb{G}$ for some group $\mathbb{G}$ (we here assume that $\mathcal{K}_{\mathcal{S}_i} = \mathcal{K}_{\mathcal{S}_j}$ for all $i, j \in [1, N]$). We assume that the subsets $\{\mathcal{S}_i\}$ of $[1, N]$ supported by the PRFs F_1 and F_2 satisfy the following requirement:

$$\mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset \Rightarrow \mathcal{S}_i \subset \mathcal{S}_j \text{ or } \mathcal{S}_j \subset \mathcal{S}_i$$

Therefore, for each subset $\mathcal{S}_i \neq [1, N]$, it is possible to define the smallest subset containing strictly $\mathcal{S}_i$. Its index is given by a function D.

To spend a value V, a user whose coin secret key is s selects a subset $\mathcal{S}_i$ containing V elements and will reveal the following information:

1. $k_{\mathcal{S}_i}^{(1)} \leftarrow F_1.\mathtt{Const}(s, \mathcal{S}_i)$
2. $k_{\mathcal{S}_i}^{(2)} \leftarrow \mathsf{upk} \cdot F_2.\mathtt{Const}(s, \mathcal{S}_i)$
3. $\mathtt{T}_{\mathcal{S}_i} \leftarrow \mathsf{upk} \cdot H'(F_1.\mathtt{Const}(s, \mathcal{S}_{D(\mathcal{S}_i)}))^R$

for some public element R. The first element will be used by the bank to derive the serial numbers $\mathtt{SN}_t \leftarrow F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(1)}, t)\ \forall t \in \mathcal{S}_i$. The second element prevents the problem we mention in Section 4.1: it will be used to discard collisions between spendings involving different users. Finally, the last element is the double spending tag but the identification process is more subtle than in the previous case, as we explain below.

Let $(k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i})$ and $(k_{\mathcal{S}_j}^{(1)}, k_{\mathcal{S}_j}^{(2)}, \mathtt{T}_{\mathcal{S}_j})$ be two spendings leading to a collision, *i.e.* such that there are $t_i \in \mathcal{S}_i$ and $t_j \in \mathcal{S}_j$ verifying the equation:

$$F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(1)}, t_i) = F_1.\mathtt{ConstEval}_{\mathcal{S}_j}(k_{\mathcal{S}_j}^{(1)}, t_j)$$

Collision resistance of F_1 implies that $t_i = t_j$ and that $k_{\mathcal{S}_i}^{(1)}$ and $k_{\mathcal{S}_j}^{(1)}$ were both derived from the same master secret key. Moreover, $t_i \in \mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset$ which implies that $\mathcal{S}_i \subset \mathcal{S}_j$ or $\mathcal{S}_j \subset \mathcal{S}_i$. Let us assume that $\mathcal{S}_j \subset \mathcal{S}_i$. We then distinguish the two following cases.

- Case 1: $\mathcal{S}_j \subsetneq \mathcal{S}_i$, which implies that $\mathcal{S}_{D(\mathcal{S}_j)} \subset \mathcal{S}_i$. From $k_{\mathcal{S}_i}^{(1)}$, one can then compute $\mathtt{T}^* \leftarrow H'(F_1.\overline{\mathtt{Const}}(k_{\mathcal{S}_i}^{(1)}, \mathcal{S}_{D(\mathcal{S}_j)}))$ and thus recover $\mathsf{upk} = \mathtt{T}_{\mathcal{S}_j}/(\mathtt{T}^*)^{R_i}$.

[11] The requirements placed on these functions are specified in Appendix C.

– Case 2: $\mathcal{S}_j = \mathcal{S}_i$. In such a case, $k^{(2)}_{\mathcal{S}_i} = k^{(2)}_{\mathcal{S}_j}$ if and only if both elements have been generated using the same public key upk. Therefore, one aborts if this equality does not hold. Else, one computes $\mathsf{upk} \leftarrow (\mathtt{T}^{R_j}_{\mathcal{S}_i} / \mathtt{T}^{R_i}_{\mathcal{S}_j})^{1/(R_j - R_i)}$.

Remark 6. To our knowledge, all anonymous divisible e-cash systems can be associated with one of these frameworks. The main difference is that existing constructions require less PRFs but, as we explain in Section 6, this leads to a problem that has been overlooked in the proofs. Although some of them can be patched without adding new PRFs, we note that this patch is very specific to some constructions and so cannot be applied to our generic frameworks.

Starting from the seminal work of Canard et Gouget [CG07], several schemes [ASM08, CG10, Mär15]) implicitly followed the second framework and so constructed (or re-used) delegatable PRFs satisfying the properties listed above. Unfortunately, the resulting PRFs do not interact nicely with NIZK, leading to quite complex constructions.

Recently, a series of papers [CPST15a, CPST15b, PST17] followed a different approach that actually matches our first framework. It it is then possible to extract from these papers constrained-key homomorphic PRFs that achieve key-pseudorandomness. Moreover, these PRFs interact smoothly with NIZK, even in the standard model, leading to very efficient constructions.

However, in practice, efficiency does not only depend on the compatibility with NIZK proofs. Divisible e-cash indeed achieves its ultimate goal when it allows the user to spend efficiently the V coins associated with a transaction of amount V. This means that the family of subsets $\{\mathcal{S}_i\}$ supported by the PRF must be as rich and as diverse as possible. For decades, the constructions have only been compatible with intervals of the form $[1+j\cdot 2^k, (j+1)2^k]$ due to the use of binary trees. It is only recently that Pointcheval, Sanders and Traoré [PST17] proposed a construction supporting *any* interval $[a, b] \subset [1, N]$. This led to the first constant-size divisible e-cash systems.

5 Our Framework

We now elaborate on the solutions sketched in the previous section to construct a full divisible e-cash system. We only consider here constructions based on key-homomorphic constrained PRFs but describe those based on delegatable PRF in Appendix C.

5.1 Building Blocks

Our framework makes use of three standard cryptographic primitives, namely digital signature, commitment scheme and non-interactive zero-knowledge (NIZK) proofs that we recall below, along with their respective security properties.

Digital Signature. A digital signature scheme Σ is defined by three algorithms:

- $\mathtt{Keygen}(1^\lambda)$: on input a security parameter λ, this algorithm outputs a pair of signing and verification keys $(\mathsf{sk}, \mathsf{pk})$;
- $\mathtt{Sign}(\mathsf{sk}, m)$: on input the signing key sk and a message m, this algorithm outputs a signature σ;
- $\mathtt{Verify}(\mathsf{pk}, m, \sigma)$: on input the verification key pk, a message m and its alleged signature σ, this algorithm outputs 1 if σ is a valid signature on m under pk, and 0 otherwise.

The standard security notion for a signature scheme is *existential unforgeability under chosen message attacks* (EUF-CMA) [GMR88]: it means that it is hard, even given access to a signing oracle, to output a valid pair (m, σ) for a message m never asked to the signing oracle. The formal definition is provided in Figure 4 and makes use of an oracle $\mathcal{O}\mathtt{Sign}$ that, on input a message m, returns $\mathtt{Sign}(\mathsf{sk}, m)$. A signature scheme is EUF-CMA secure if $\Pr[\mathtt{Exp}_{\mathcal{A}}^{\mathtt{euf-cma}}(1^\lambda) = 1]$ is negligible for any $\mathcal{A}$.

```
$\mathtt{Exp}_{\mathcal{A}}^{\mathtt{euf-cma}}(1^\lambda)$ – EUF-CMA security Game
 1. $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathtt{Keygen}(1^\lambda)$
 2. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Sign}}(\mathsf{pk})$
 3. If $\mathtt{Verify}(\mathsf{pk}, m^*, \sigma^*) = 0$ or $\mathcal{O}\mathtt{Sign}$ queried on $m^*$, then return 0
 4. Return 1
```

Fig. 4. Security Game for Digital Signature

Commitment Scheme. A commitment scheme Γ is defined by the following two algorithms:

- $\mathtt{Keygen}(1^\lambda)$: on input a security parameter λ, this algorithm outputs a commitment key ck that specifies a message space $\mathcal{M}$, a randomizer space $\mathcal{R}$ along with a commitment space $\mathcal{C}$;
- $\mathtt{Commit}(\mathsf{ck}, m, r)$: on input ck, an element $r \in \mathcal{R}$ and a message $m \in \mathcal{M}$, this algorithm returns a commitment $c \in \mathcal{C}$.

Informally, a commitment should not be equivocal and should hide the committed message. This is formally defined by the games $\mathtt{Exp}_{\mathcal{A}}^{\mathtt{hid}-b}(1^\lambda)$ and $\mathtt{Exp}_{\mathcal{A}}^{\mathtt{bid}}(1^\lambda)$ of Figure 5. A commitment scheme is computationally (resp. perfectly) hiding if $\Pr[\mathtt{Exp}_{\mathcal{A}}^{\mathtt{hid}-1}(1^\lambda)] - \Pr[\mathtt{Exp}_{\mathcal{A}}^{\mathtt{hid}-0}(1^\lambda)]$ is negligible (resp. equal to 0). A commitment scheme is computationally (resp. perfectly) binding if $\Pr[\mathtt{Exp}_{\mathcal{A}}^{\mathtt{bind}}(1^\lambda)] = 1$ is negligible (resp. equal to 0).

NIZK Proofs. Let R be an efficiently computable relation with triples $(\mathtt{crs}, \phi, w)$, where $\mathtt{crs}$ is called the *common reference string* and w is said to be a *witness* to the *instance* ϕ. A NIZK proofs system is defined by the following three algorithms:

```
$\mathtt{Exp}_{\mathcal{A}}^{\mathtt{hid}-b}(1^\lambda)$ – Hiding security Game
  1. $(\mathsf{ck}) \leftarrow \mathtt{Keygen}(1^\lambda)$
  2. $m \leftarrow \mathcal{A}(\mathsf{ck})$
  3. $r \xleftarrow{\$} \mathcal{R}$, $c_0 \leftarrow \mathtt{Commit}(\mathsf{ck}, m, r)$
  4. $c_1 \xleftarrow{\$} \mathcal{C}$
  5. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Sign}}(\mathsf{ck}, c_b)$
  6. Return $(b = b^*)$

$\mathtt{Exp}_{\mathcal{A}}^{\mathtt{bind}}(1^\lambda)$ – Binding security Game
  1. $(\mathsf{ck}) \leftarrow \mathtt{Keygen}(1^\lambda)$
  2. $(m_0, m_1, r_0, r_1) \leftarrow \mathcal{A}(\mathsf{ck})$
  3. If $m_0 = m_1$ or $\mathtt{Commit}(\mathsf{ck}, m_0, r_0) \neq \mathtt{Commit}(\mathsf{ck}, m_1, r_1)$, then return 0
  4. Return 1
```

Fig. 5. Security Game for Commitment Schemes

– $\mathtt{Setup}(1^\lambda)$: on input a security parameter λ, this algorithm outputs the common reference string $\mathtt{crs}$.
– $\mathtt{Prove}(\mathtt{crs}, w, \phi)$: on input a triple $(\mathtt{crs}, w, \phi) \in R$, this algorithm outputs a proof π.
– $\mathtt{Verify}(\mathtt{crs}, \phi, \pi)$: on input $\mathtt{crs}$, a proof π and an instance ϕ this algorithm outputs either 1 (accept) or 0 reject.

A NIZK proof is correct if the probability that $\mathtt{Verify}(\mathtt{crs}, \phi, \mathtt{Prove}(\mathtt{crs}, w, \phi))$ returns 0 is negligible for all $(\mathtt{crs}, w, \phi) \in R$. We will additionally requires the properties of *zero-knowledge* and *extractability*. Both of them are defined in Figure 6. Extractability requires the existence of an extractor $X_{\mathcal{A}}$ that takes as input the transcript $\mathtt{trans}_{\mathcal{A}}$ of the adversary $\mathcal{A}$. Zero-knowledge requires the existence of a simulator consisting of the algorithms $\mathtt{SimSetup}$ and $\mathtt{SimProve}$ that share state with each other. In the security experiment $\mathtt{Exp}_{\mathcal{A}}^{\mathtt{zk}-b}(1^\lambda)$, the adversary has access to the following oracle:

– $\mathcal{O}\mathtt{Prove}\text{-}b(w, \phi)$: on input (w, ϕ), this algorithm returns $\perp$ if $(\mathtt{crs}_b, w, \phi) \notin R$. Else, it returns $\mathtt{Prove}(\mathtt{crs}_b, w, \phi)$ if $b = 0$ and $\mathtt{SimProve}(\mathtt{crs}_b, \phi)$ otherwise.

A NIZK proof is zero-knowledge if $\Pr[\mathtt{Exp}_{\mathcal{A}}^{\mathtt{zk}-1}(1^\lambda)] - \Pr[\mathtt{Exp}_{\mathcal{A}}^{\mathtt{zk}-0}(1^\lambda)]$ is negligible. It is extractable if $\Pr[\mathtt{Exp}_{\mathcal{A}}^{\mathtt{ext}}(1^\lambda) = 1]$ is negligible.

5.2 Construction

Our construction makes use of a digital signature scheme Σ, a commitment scheme Γ and a NIZK proof system Π as described above. The difficulty here is to provide the description of a framework that encompasses very different settings such as cyclic groups or lattices. For example, the element ID^R of Section 4 that was involved in double-spending tags does not make sense in a lattice setting and

```
$\mathtt{Exp}_{\mathcal{A}}^{\mathtt{zk}-b}(1^\lambda)$ – Zero-Knowledge Game
  1. $\mathtt{crs}_0 \leftarrow \mathtt{Setup}(1^\lambda)$
  2. $\mathtt{crs}_1 \leftarrow \mathtt{SimSetup}(1^\lambda)$
  3. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Prove}-b}(\mathtt{crs}_b)$
  4. Return $(b = b^*)$

$\mathtt{Exp}_{\mathcal{A}}^{\mathtt{ext}}(1^\lambda)$ – Extractability Game
  1. $\mathtt{crs} \leftarrow \mathtt{Setup}(1^\lambda)$
  2. $(\phi, \pi) \leftarrow \mathcal{A}(\mathtt{crs})$
  3. $w \leftarrow X_{\mathcal{A}}(\mathtt{trans}_{\mathcal{A}})$
  4. If $\mathtt{Verify}(\mathtt{crs}, \phi, \pi) = 0$ or $(\mathtt{crs}, w, \phi) \in R$, then return 0
  5. Return 1
```

Fig. 6. Security Game for NIZK proofs

would in practice be replaced by $R \cdot ID$ where R is a matrix and ID is a vector. To remain as generic as possible, we will then introduce several functions that will abstract the properties we need. In our example, we need that $ID^{R+R'} = ID^R \cdot ID^{R'}$ and that $(R+R') \cdot ID = R \cdot ID + R' \cdot ID$ and so will represent ID^R and $R \cdot ID$ by $G(ID, R)$ where G is a bilinear map (see remark 8 for more details). Such functions make the description of our framework rather complex but we stress that they are actually very easy to instantiate. In particular, we emphasize that the following framework essentially formalises the high-level ideas described in Section 4 and does not significantly increase the practical complexity of our construction.

- $\mathtt{Setup}(1^\lambda, N)$: To generate the public parameters pp, the algorithm first computes $\mathtt{crs} \leftarrow \Pi.\mathtt{Setup}(1^\lambda)$. It then selects four constrained PRFs F_1, F_2, F_3 and F_4 with the same master key space $\mathcal{K}$ and that support the same subsets $\mathcal{S}_1, \ldots, \mathcal{S}_n$ with $\mathcal{S}_i \subset [1, N]\ \forall i \in [1, n]$. For sake of simplicity, we assume that $\mathcal{K}_{\mathcal{S}_i} = \mathcal{K}_{\mathcal{S}_j} = \mathcal{K}_{\mathcal{S}}$ for all $i, j \in [1, n]$. F_2, F_3 and F_4 must additionally be key-homomorphic. Finally, it selects a hash function $H : \{0,1\}^* \to \mathbb{G}$ for some group $\mathbb{G}$, two functions $G_1 : \{0,1\}^* \to \mathcal{K}$, $G_2 : \{0,1\}^* \to \mathcal{K}_{\mathcal{S}}$ along with a non degenerate bilinear map $G_3 : \mathcal{K}_{\mathcal{S}} \times \mathbb{G} \to \mathcal{K}_{\mathcal{S}}$ (see remark 8). The public parameters pp are then set as $\mathtt{crs}, F_1, F_2, F_3, F_4, H, G_1, G_2, G_3$.
- $\mathtt{BKeygen}()$: The bank generates a commitment key $\mathsf{ck} \leftarrow \Gamma.\mathtt{Keygen}(1^\lambda)$ and a key pair $(\mathsf{sk}_B, \mathsf{pk}_B) \leftarrow \Sigma.\mathtt{Keygen}(1^\lambda)$. It then sets bsk as sk_B and bpk as $(\mathsf{ck}, \mathsf{pk}_B)$.
- $\mathtt{Keygen}()$: The user (resp. the merchant) generates a signature key pair $(\mathsf{usk}, \mathsf{upk})$ (resp. $(\mathsf{msk}, \mathsf{mpk})$) using $\Sigma.\mathtt{Keygen}$.
- $\mathtt{Withdraw}(\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk}))$: To withdraw a divisible coin, the user first generates $s \leftarrow F_1.\mathtt{Keygen}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ and a random element r from the randomizer space $\mathcal{R}$ of Γ. It then sends $c \leftarrow \Gamma.\mathtt{Commit}(\mathsf{ck}, [s, \mathsf{upk}], r)$ to the bank along with a signature $\tau_c \leftarrow \Sigma.\mathtt{Sign}(\mathsf{usk}, c)$.

If τ_c is valid, the bank returns a signature $\sigma_c \leftarrow \Sigma.\mathtt{Sign}(\mathsf{sk}_B, c)$ to the user. The latter can then set its coin C as (c, s, r, σ_c).

– $\mathtt{Spend}(\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, V), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, \mathsf{info}, V))$: During a spending of amount V, the merchant first selects a string info that he never used before[12] and sends it to the user along with his public key mpk.
The user then selects a subset $\mathcal{S}_i$ with $|\mathcal{S}_i| = V$ such that $\mathtt{SN}_j$ has never been revealed for all $j \in \mathcal{S}_i$, and computes $k_{\mathcal{S}_i}^{(1)} \leftarrow F_1.\mathtt{Const}(s, \mathcal{S}_i)$, $k_{\mathcal{S}_i}^{(2)} \leftarrow F_2.\mathtt{Const}(s \cdot G_1(\mathsf{upk}), \mathcal{S}_i)$ and $\mathtt{T}_{\mathcal{S}_i} \leftarrow (G_3(G_2(\mathsf{upk}), H(\mathsf{mpk}||\mathsf{info})) \cdot k_{\mathcal{S}_i}^{(3)}, G_2(\mathsf{upk}) \cdot k_{\mathcal{S}_i}^{(4)})$ where $k_{\mathcal{S}_i}^{(3)} = F_3.\mathtt{Const}(s, \mathcal{S}_i)$ and $k_{\mathcal{S}_i}^{(4)} = F_4.\mathtt{Const}(s, \mathcal{S}_i)$.
Finally, it generates a signature $\tau \leftarrow \Sigma.\mathtt{Sign}(\mathsf{usk}, (\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}))$ along with a NIZK proof π of $(\mathsf{upk}, s, c, r, \sigma_c, \mathcal{S}_i, \tau)$ such that:

1. $\exists i^* \in [1, n] : \mathcal{S}_i = \mathcal{S}_{i^*} \wedge |\mathcal{S}_i| = V$
2. $c = \Gamma.\mathtt{Commit}(\mathsf{ck}, [s, \mathsf{upk}], r)$
3. $1 = \Sigma.\mathtt{Verify}(\mathsf{pk}_B, c, \sigma_c)$
4. $k_{\mathcal{S}_i}^{(1)} = F_1.\mathtt{Const}(s, \mathcal{S}_i)$
5. $k_{\mathcal{S}_i}^{(2)} = F_2.\mathtt{Const}(s \cdot G_1(\mathsf{upk}), \mathcal{S}_i)$
6. $\mathtt{T}_{\mathcal{S}_i} = (G_3(G_2(\mathsf{upk}), H(\mathsf{mpk}||\mathsf{info})) \cdot F_3.\mathtt{Const}(s, \mathcal{S}_i), G_2(\mathsf{upk}) \cdot F_4.\mathtt{Const}(s, \mathcal{S}_i))$
7. $1 = \Sigma.\mathtt{Verify}(\mathsf{upk}, (\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}), \tau)$

The elements $(k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}, \pi)$ are then sent to the merchant who accepts them as a payment if π is valid.

– $\mathtt{Deposit}(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}, \pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: To deposit a transaction, the merchant sends its transcript $\mathsf{Tr} \leftarrow (V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}, \pi)$ along with a signature $\mu \leftarrow \Sigma.\mathtt{Sign}(\mathsf{msk}, \mathsf{Tr})$. The bank then checks that (1) the proof π is valid, (2) π proves knowledge of a signature on a tuple whose first coordinate is mpk, (3) $\Sigma.\mathtt{Verify}(\mathsf{mpk}, \mathsf{Tr}, \mu) = 1$ and (4) that this merchant has not previously deposited a transaction associated with info. If one of the first three conditions is not satisfied, then the bank returns $\perp$. If the last condition is not satisfied then the bank knows another transcript $(V', \mathsf{info}, k_{\mathcal{S}_j}^{(1)}, k_{\mathcal{S}_j}^{(2)}, \mathtt{T}_{\mathcal{S}_j}, \pi')$ along with a signature μ'. All these elements, along with $[\mathsf{Tr}, \mu]$ constitute a proof of double-deposit.
Else, the bank recovers the serial numbers $\mathtt{SN}_j \leftarrow F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(1)}, j)||F_2.\mathtt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(2)}, j)$ for all $j \in \mathcal{S}_i$ (see remark 9 below). It then distinguishes the following two cases:

- $\exists j^* \in \mathcal{S}_i$ such that $\mathtt{SN}_{j^*}$ already belongs to L. In such a case, the bank recovers the first transcript $(V', \mathsf{info}', \mathsf{mpk}', k_{\mathcal{S}_{i'}}^{(1)}, k_{\mathcal{S}_{i'}}^{(2)}, \mathtt{T}_{\mathcal{S}_{i'}}, \pi')$ that yields this serial number and returns it along with Tr.
- $\mathtt{SN}_j \notin L \; \forall j \in \mathcal{S}_i$, in which case the bank simply adds these serial numbers to L

[12] This string can simply be a counter incremented by the merchant after each transaction, or include information that uniquely identifies the transaction such as the date and the hour.

– $\mathtt{Identify}((V, \mathsf{info}, \mathsf{mpk}, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi), (V', \mathsf{info}', \mathsf{mpk}', k^{(1)}_{\mathcal{S}_j}, k^{(2)}_{\mathcal{S}_j}, \mathtt{T}_{\mathcal{S}_j}, \pi'), \mathsf{bpk})$: Given two transcripts, this algorithm first checks that (1) $\mathsf{mpk}||\mathsf{info} \neq \mathsf{mpk}'||\mathsf{info}'$ and (2) both proofs π and π' are valid. If one of these conditions is not satisfied, then it returns 0. Else, it checks that there is a collision between the serial numbers derived from these transcripts, *i.e.* there are $x \in \mathcal{S}_i$ and $x' \in \mathcal{S}_j$ such that $F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(k^{(1)}_{\mathcal{S}_i}, x)||F_2.\mathtt{ConstEval}_{\mathcal{S}_i}(k^{(2)}_{\mathcal{S}_i}, x) = F_1.\mathtt{ConstEval}_{\mathcal{S}_j}(k^{(1)}_{\mathcal{S}_j}, x')||F_2.\mathtt{ConstEval}_{\mathcal{S}_j}(k^{(2)}_{\mathcal{S}_j}, x')$. If there is no collision, it outputs 0.
  Else, it proceeds as in Section 4.1 to identify the defrauder. If $\mathcal{S}_i = \mathcal{S}_j$, it computes $R = H(\mathsf{mpk}||\mathsf{info})$, $R' = H(\mathsf{mpk}'||\mathsf{info}')$ along with

$$F_3.\mathtt{ConstEval}_{\mathcal{S}_i}(\mathtt{T}_{\mathcal{S}_i}[1], x)/F_3.\mathtt{ConstEval}_{\mathcal{S}_j}(\mathtt{T}_{\mathcal{S}_j}[1], x')$$

  and $F_3.\mathtt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}), R), x)/F_3.\mathtt{ConstEval}_{\mathcal{S}_j}(G_3(G_2(\mathsf{upk}), R'), x)$ for all upk until it gets a match. It then returns the corresponding public key upk^* (or $\perp$ if the exhaustive search fails).
  If $\mathcal{S}_i \neq \mathcal{S}_j$, it computes

$$F_4.\mathtt{ConstEval}_{\mathcal{S}_{i_1}}(T^{(1)}_{\mathcal{S}_{i_1}}[2]/G_2(\mathsf{upk}), j)/F_4.\mathtt{ConstEval}_{\mathcal{S}_{i_2}}(T^{(1)}_{\mathcal{S}_{i_2}}[2]/G_2(\mathsf{upk}), j)$$

  for all public keys upk until it gets $1_{\mathcal{Y}}$. It then returns the corresponding public key upk^* (or $\perp$ if the exhaustive search fails).
– $\mathtt{CheckDeposit}([(V, \mathsf{info}, \mathsf{mpk}, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi), \mu], \mathsf{bpk})$: this algorithm checks that π is valid and that $1 = \Sigma.\mathtt{Verify}(\mathsf{mpk}, (V, \mathsf{info}, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi), \mu)$ in which case it outputs 1. Else, it returns 0.

Remark 7. We provide in Appendix E an example of instantiation of our PRFs that helps to assess the practical complexity of our construction. Nevertheless, we note that a spending essentially consists in generating 4 constrained keys along with a zero-knowledge proof that they have been correctly computed from a certified master key. In bilinear groups, such proofs can easily be produced in the random oracle model or by using Groth-Sahai proofs [GS08] if one selects an appropriate digital signature scheme for Σ. The case of lattices is more complex but we note that the proofs and the signature scheme required here are similar to those described in [LLNW17].

Remark 8. The only purpose of the functions G_1, G_2 and G_3 is to project the different elements of our system on the appropriate spaces, which ensures compatibility with most PRFs. As we illustrate on concrete examples in Appendix E, these functions are in practice very simple (for example G_2 is usually the identity function) and nicely interact with zero-knowledge proofs. In particular, our bilinear map G_3 can easily be instantiated in most settings. For example, when $\mathcal{K}_{\mathcal{S}}$ is a cyclic group of order p, we will simply have $\mathbb{G} = \mathbb{Z}_p$ and $G_3(x, y) = x^y$. Similarly, when $\mathcal{K}_{\mathcal{S}} = \mathbb{F}_q^n$, we will have $\mathbb{G} \subset \mathcal{M}_{m,n}$ and $G_3(x, A) = A \cdot x$.

We will also manage to make G_1 and G_2 injective in practice which means that the collision resistance will be trivially satisfied. We recall that the bilinear map G_3 is non degenerate if $G_3(x, y) = 1_{\mathcal{K}_{\mathcal{S}}}$ implies $x = 1_{\mathcal{K}_{\mathcal{S}}}$ or $y = 1_{\mathbb{G}}$.

Remark 9. Note that, even if the bank does not know the subset $\mathcal{S}_i$, it is always able to recover all the serial numbers $\texttt{SN}_j \leftarrow \texttt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}, j)$, for $j \in \mathcal{S}_i$. Indeed, it can generates the list L containing $\texttt{SN}_k \leftarrow \texttt{ConstEval}_{\mathcal{S}}(k_{\mathcal{S}_i}, k)$, for all $\mathcal{S}$ containing V elements and $k \in \mathcal{S}$. Such a list contains the valid serial numbers (those for which $\mathcal{S} = \mathcal{S}_i$) and so can still be used to detect double-spendings. Moreover, due to the properties of PRF, the "invalid" serial numbers (those for which $\mathcal{S} \neq \mathcal{S}_j$) are random elements and so are unlikely to create false positives (collisions in the list L that are not due to double-spendings).

However, we stress that this is only a generic solution that works for any instantiation of our construction. In practice, it leads to quite complex deposits and so should be avoided, if possible. Actually, to our knowledge, it is only used in [CPST15a]. All other divisible e-cash systems manage to construct PRFs that can be evaluated on the elements of $\mathcal{S}_i$ without knowing $\mathcal{S}_i$. More specifically, theses PRFs are compatible with an algorithm $\overline{\texttt{ConstEval}}$ that takes as input a constrained key and the size of the corresponding subset and that outputs $\overline{\texttt{ConstEval}}(k_{\mathcal{S}_i}, |\mathcal{S}_i|) = \{\texttt{ConstEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}, x), \forall x \in \mathcal{S}_i\}$.

The security of our construction is stated by the following theorem, proven in Appendix A.

Theorem 10. *Our divisible e-cash system is*

- *traceable if F_1 and F_2 achieve collision resistance-1, Γ is computationally binding, Σ is EUF-CMA secure, Π is extractable, and G_1 is collision resistant.*
- *exculpable if Σ is EUF-CMA secure, Π is extractable, F_1 and F_2 achieve collision resistance-1, F_3 achieves collision resistance-2, F_4 achieves collision resistance-3 and H, G_1 and G_2 are collision resistant.*
- *clearable if Σ is EUF-CMA secure.*
- *anonymous if (F_1, F_2, F_3, F_4) achieves combined key-pseudorandomness, Γ is computationally hiding and Π is zero-knowledge.*

Remark 11. Most existing constructions require a collaborative generation of the coin secret values. Our framework can easily support this feature if Γ is homomorphic. In such a case, traceability no longer requires collision resistance for F_1 and F_2 because the randomness added by the bank (which is honest in this game) will make collisions very unlikely. Unfortunately, the collaborative generation has no effect on exculpability since both parties (the user and the bank) can be corrupted in this game. We therefore choose to simplify our withdrawal protocol by removing this step since we need collision resistance of F_1 and F_2 anyway.

6 A Note on Exculpability of Previous Constructions

Exculpability implies that a coin withdrawn by a user whose public key is upk^* can only be spent by the latter. All e-cash constructions enforce this property by requiring a signature (potentially a signature of knowledge) on the transaction under upk^*. Intuitively, this seems sufficient: a transaction accusing an honest user of fraud should contain a signature (or more specifically a proof of knowledge of a signature) under upk^* and so would imply a forgery. Actually, this argument is ubiquitous in previous papers[13] and leads to quite simple security proofs. It is explicitly stated in Section D.3 of the full version of [LLNW17] and in Section 4.6 of [CG10], and implicitly used in Section 6.3 of [CPST15a], in Section 6.2 of [PST17] and in the security proofs (p22) of the full version of [CHL05].

Unfortunately, this argument is not correct because of the complex identification process of e-cash systems. Indeed, the public key upk^* returned by the `Identify` algorithm is not extracted from the signature itself, but from a complex formula involving several elements, such as PRF seeds, scalars, etc. An adversary might then select appropriate values that will lead `Identify` to output upk^* while taking as input two transactions generated with different public keys (as explained in Section 4.1). This scenario, that has not been taken into account in previous papers, invalidates their proofs[14] because, in such a case, the transactions do not contain a valid signature under upk^*.

To illustrate this problem, let us consider the lattice-based construction proposed by Libert *et al* [LLNW17]. In this system, each user selects a short vector $\mathbf{e}$ and defines their public key as $\mathbf{F}.\mathbf{e}$ for some public matrix $\mathbf{F}$. Each coin withdrawn by this user is associated with two vectors $\mathbf{k}$ and $\mathbf{t}$. The former is used to generate the i-th serial number $y_S = \lfloor \mathbf{A}_i \cdot \mathbf{k} \rfloor_p$ for some public matrix $\mathbf{A}_i$ while the latter is used to generate the double-spending tag $y_T = \mathsf{upk} + \mathbf{H(R)} \cdot \lfloor \mathbf{A}_i \cdot \mathbf{t} \rfloor_p$, where $\mathbf{H(R)}$ is a matrix derived from public information associated with the transaction.

If two transactions yield the same serial number, then one computes $y^* = (\mathbf{H(R)} - \mathbf{H(R')})^{-1}(y_T - y'_T)$ and returns $y_T - \mathbf{H(R)} \cdot y^*$. One can note that this formula indeed returns a public key upk^* if *both* transactions have been generated by the user upk^*. However, there is no equivalence here, and an adversary might manage to generate $\mathbf{R}, \mathbf{R'}, \mathbf{t}, \mathbf{t'}, \mathsf{upk}, \mathsf{upk}'$ (recall that in the exculpability game the adversary controls the bank, the merchants and all dishonest users) such that

$$\mathsf{upk}^* = y_T - \mathbf{H(R)} \cdot (\mathbf{H(R)} - \mathbf{H(R')})^{-1}(y_T - y'_T)$$

And even if we modify the original protocol to ensure that collisions only occur when $\mathbf{t} = \mathbf{t}'$, the previous relation still gives us

$$\begin{aligned}\mathsf{upk}^* &= y_T - \mathbf{H(R)} \cdot (\mathbf{H(R)} - \mathbf{H(R')})^{-1}(y_T - y'_T) \\ &= \mathsf{upk} - \mathbf{H(R)} \cdot (\mathbf{H(R)} - \mathbf{H(R')})^{-1}(\mathsf{upk} - \mathsf{upk}')\end{aligned}$$

[13] Our comment obviously only applies to papers that provide a security proof.
[14] We stress that the problem is located in the proofs and not in the definition of the exculpability property.

from which it is not possible to conclude that $\mathsf{upk}^* = \mathsf{upk} = \mathsf{upk}'$. In particular, it does not seem possible to extract from these transactions a short vector $\mathbf{e}^*$ such that $\mathsf{upk}^* = \mathbf{F} \cdot \mathbf{e}^*$, which invalidates the original proof.

This problem is not exclusive to lattice-based constructions but we note that the proofs can be fixed in the case where $\mathsf{upk} = g^x$ for some secret scalar x and where the transactions contain a signature of knowledge of the different secret values (including x). This is actually quite frequent in existing constructions (*e.g.* [CHL05,CG07,CG10] and the ROM constructions of [CPST15a,CPST15b, PST17]).

Indeed, in such a case, the double-spending tag is of the form $T = \mathsf{upk} \cdot F.\mathtt{Eval}(s,i)^R$ where s is a seed, $i \in [1, N]$ is an integer and R is derived from public information. In case of double-spending, there are two tags T and T' from which one can recover upk by computing $(T^{R'}/(T')^R)^{\frac{1}{R'-R}}$.

Here again, an adversary might generate $\mathsf{upk}, s, R, \mathsf{upk}', s', R'$ such that the corresponding tags T and T' satisfy $(T^{R'}/(T')^R)^{\frac{1}{R'-R}} = \mathsf{upk}^*$, for some honest public key upk^*. However, in this case, the reduction can recover the discrete logarithm of upk^* by extracting all the secret values from the proofs generated by the adversary. This means that exculpability can still be proven under the discrete logarithm assumption and so that the original proofs can easily be fixed by adding this remark.

Unfortunately, this patch is inherent to signatures of knowledge of discrete logarithms and so cannot be applied to other settings (*e.g.* lattices) or to standard model constructions. Regarding the latter, we note that the proofs of all existing constructions [CPST15a,CPST15b,PST17] no longer work but this does not lead to straightforward attacks either. Indeed, even if the adversary manages to generate $\mathsf{upk}, s, R, \mathsf{upk}', s', R'$ such that $(T^{R'}/(T')^R)^{\frac{1}{R'-R}} = \mathsf{upk}^*$, its attack only succeeds if it is also able to produce valid signatures under upk and upk'. Unfortunately, it seems very hard to formalize this fact and we would therefore recommend to update these constructions as described in Section 5.2 (*i.e.* to use 4 PRFs instead of 2) to prevent further issues.

7 Conclusion

Decades after their introduction, divisible e-cash systems are still remarkably hard to design, and even to analyse. Existing schemes are based on intricate mechanisms, tailored to very specific settings, and so can hardly be reproduced in different contexts. Moreover, such mechanisms often rely on ad-hoc computational problems whose difficulty is hard to assess.

In this paper we introduce the first frameworks for divisible e-cash systems that only use constrained PRFs and very standard cryptographic primitives. We prove the security of our global constructions assuming that each of the building blocks achieve some properties that we identify.

Our work thus presents this complex primitive in a new light, highlighting its strong relations with constrained PRFs. More specifically, it shows that the bulk of the design of a divisible e-cash system is the construction of a constrained PRF with some specific features. We therefore hope that our results will encourage designers of constrained PRFs to add these features to their construction, so as to implicitly define a new divisible e-cash scheme. We in particular believe that it is an important step towards a post-quantum divisible e-cash system.

Acknowledgements

This work is supported by the European Union PROMETHEUS project (Horizon 2020 Research and Innovation Program, grant 780701).

References

ASM08. Man Ho Au, Willy Susilo, and Yi Mu. Practical anonymous divisible e-cash from bounded accumulators. In Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 287–301. Springer, Heidelberg, January 2008.

BCF$^+$11. Olivier Blazy, Sébastien Canard, Georg Fuchsbauer, Aline Gouget, Hervé Sibert, and Jacques Traoré. Achieving optimal anonymity in transferable e-cash with a judge. In Abderrahmane Nitaj and David Pointcheval, editors, *AFRICACRYPT 11*, volume 6737 of *LNCS*, pages 206–223. Springer, Heidelberg, July 2011.

BCFK15. Foteini Baldimtsi, Melissa Chase, Georg Fuchsbauer, and Markulf Kohlweiss. Anonymous transferable E-cash. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 101–124. Springer, Heidelberg, March / April 2015.

BCN$^+$10. Patrik Bichsel, Jan Camenisch, Gregory Neven, Nigel P. Smart, and Bogdan Warinschi. Get shorty via group signatures without encryption. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 381–398. Springer, Heidelberg, September 2010.

BFP$^+$15. Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 31–60. Springer, Heidelberg, March 2015.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014.

BKM17. Dan Boneh, Sam Kim, and Hart William Montgomery. Private puncturable PRFs from standard lattice assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 415–445. Springer, Heidelberg, May 2017.

BLMR13. Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013.

BLW17. Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In Serge Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 494–524. Springer, Heidelberg, March 2017.

BMW03. Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629. Springer, Heidelberg, May 2003.

BSZ05. Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 136–153. Springer, Heidelberg, February 2005.

BW13. Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013.

CFN90. David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327. Springer, Heidelberg, August 1990.

CG07. Sébastien Canard and Aline Gouget. Divisible e-cash systems can be truly anonymous. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 482–497. Springer, Heidelberg, May 2007.

CG08. Sébastien Canard and Aline Gouget. Anonymity in transferable e-cash. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS 08*, volume 5037 of *LNCS*, pages 207–223. Springer, Heidelberg, June 2008.

CG10. Sébastien Canard and Aline Gouget. Multiple denominations in e-cash with compact transaction data. In Radu Sion, editor, *FC 2010*, volume 6052 of *LNCS*, pages 82–97. Springer, Heidelberg, January 2010.

CGT08. Sébastien Canard, Aline Gouget, and Jacques Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. In Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 202–214. Springer, Heidelberg, January 2008.

Cha82. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

CHL05. Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 302–321. Springer, Heidelberg, May 2005.

CL04. Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, August 2004.

CP93. David Chaum and Torben P. Pedersen. Transferred cash grows in size. In Rainer A. Rueppel, editor, *EUROCRYPT'92*, volume 658 of *LNCS*, pages 390–407. Springer, Heidelberg, May 1993.

CPST15a. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible E-cash made practical. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 77–100. Springer, Heidelberg, March / April 2015.

CPST15b. Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Scalable divisible E-cash. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 287–306. Springer, Heidelberg, June 2015.

Duc10. Léo Ducas. Anonymity from asymmetry: New constructions for anonymous HIBE. In Josef Pieprzyk, editor, *CT-RSA 2010*, volume 5985 of *LNCS*, pages 148–164. Springer, Heidelberg, March 2010.

FPV09. Georg Fuchsbauer, David Pointcheval, and Damien Vergnaud. Transferable constant-size fair e-cash. In Juan A. Garay, Atsuko Miyaji, and Akira Otsuka, editors, *CANS 09*, volume 5888 of *LNCS*, pages 226–247. Springer, Heidelberg, December 2009.

GGM84. Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th FOCS*, pages 464–479. IEEE Computer Society Press, October 1984.

GMR88. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

GPS08. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

GS08. Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 415–432. Springer, Heidelberg, April 2008.

KPTZ13. Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 669–684. ACM Press, November 2013.

LLNW17. Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based PRFs and applications to E-cash. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 304–335. Springer, Heidelberg, December 2017.

Mär15. Patrick Märtens. Practical divisible e-cash. *IACR Cryptology ePrint Archive*, 2015:318, 2015.

OO92. Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 324–337. Springer, Heidelberg, August 1992.

PS96. David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 252–265. Springer, Heidelberg, November 1996.

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.

PS16. David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Heidelberg, February / March 2016.

PST17. David Pointcheval, Olivier Sanders, and Jacques Traoré. Cut down the tree to achieve constant complexity in divisible E-cash. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 61–90. Springer, Heidelberg, March 2017.

Auxiliary Material

A Proof of Theorem 10

A.1 Proof of Traceability

A successful adversary $\mathcal{A}$ against the traceability is able to spend more than it has withdrawn without being identified. Formally, this means that there are u transcripts $\mathsf{Tr}_i = \{(V_i, \mathsf{info}_i, \mathsf{mpk}_i, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}, \pi_i)\}_{i=1}^{u}$ where:

- $\sum_{i=1}^{u} V_i > m.N$, with m the number of withdrawn coins
- $\mathtt{Identify}(\mathsf{Tr}_i, \mathsf{Tr}_j) = \perp \ \forall i, j \in [1, u]$

The latter condition implies that all the proofs π_i are valid (otherwise `Identify` would have returned 0 and not $\perp$). Therefore, the challenger is able to extract from them tuples $W_i = (\mathsf{upk}_i, s_i, r_i, \sigma_i, c_i, \mathcal{S}_i, \tau_i)$ satisfying the 7 relations defined by the `Spend` algorithm. Else, $\mathcal{A}$ could be trivially converted into an adversary against the extractability of Π.

We then distinguish the following cases:

- Type 1 Adversary: $\exists i \in [1, u]$ such that none of the coins generated through $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ queries contains the commitment c_i.
- Type 2 Adversary: all the commitments c_i have been generated during a query to the $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ oracle but the serial numbers do not collide.
- Type 3 Adversary: all the commitments c_i have been generated during a query to the $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ oracle and there is a collision in the list of serial numbers.

Informally, the first two types of adversary essentially imply an attack against the signature scheme or the commitment scheme. The last type of adversary implies a collision of one of the functions involved in our construction. This is formally stated by the following lemmas.

Lemma 12. *Any type 1 adversary $\mathcal{A}$ can be converted into an adversary succeeding against the EUF-CMA security of Σ with the same probability.*

Proof. The reduction $\mathcal{R}$ generates as usual the public parameters along with the commitment key ck but sets $\mathsf{pk}_{\mathcal{B}}$ as pk^*, where pk^* is the public key received from the challenger of the EUF-CMA security experiment.

Each time it receives an $\mathcal{O}\mathtt{Add}$ query it generates a new key pair for the user, allowing it to answer any $\mathcal{O}\mathtt{Spend}$ query.

Upon receiving an $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ query, it forwards the commitment c to its signing oracle. It then receives a signature σ_c that it can send to $\mathcal{A}$.

The simulation is perfect so $\mathcal{A}$ eventually outputs several transcripts, one of which involving a commitment c^* and a signature σ^* such that $\Sigma.\mathtt{Verify}(\mathsf{pk}^*, c^*, \sigma^*) = 1$. Since we here consider the first type of adversary, c^* has never been submitted to the signing oracle, which means that (c^*, σ^*) is a valid forgery. $\mathcal{R}$ is then able to extract this pair and so to break the EUF-CMA security of Σ. $\square$

Lemma 13. *Any type 2 adversary $\mathcal{A}$ can be converted into an adversary succeeding against the binding security of Γ, with the same probability.*

Proof. The reduction $\mathcal{R}$ generates the public parameters and $(\mathsf{sk}_B, \mathsf{pk}_B) \leftarrow \Sigma.\mathtt{Keygen}(1^\lambda)$ but receives $\mathtt{ck}$ from the challenger of the binding security experiment. $\mathcal{R}$ is able to answer any oracle query because it knows all the corresponding secret keys. At the end of the game, $\mathcal{A}$ then outputs u transcripts Tr_i that foil the mechanisms of identification. $\mathcal{R}$ extracts the $\sum_{i=1}^{u} V_i > m \cdot N$ serial numbers which are assumed to be distinct since we only consider type 2 adversary.

Since all the commitments c_i have been certified during a query to the $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ oracle (otherwise $\mathcal{A}$ would be of type 1) and since there have been m withdrawals, there are at most m distinct commitments c_i. Let (s_k, upk_k) be the secret master key/user public key pair extracted from Tr_k, for $k \in [1, u]$. We note that $\{(s_k, \mathsf{upk}_k)\}_{k=1}^{u}$ contains at least $m+1$ elements, otherwise these pairs would yield at most $m \cdot N < \sum_{i=1}^{u} V_i$ serial numbers and there would be at least one collision.

Therefore, there are only m commitments c_i that can be opened to at least $m+1$ distinct pairs (s_k, upk_k). This implies that there is at least one commitment c^* that can be opened in two different ways. By extracting all the commitments c_i and all the pairs (s_k, upk_k), $\mathcal{R}$ can easily recover c^* along with the different openings that constitute a valid attack against Γ. ◻

Lemma 14. *Any type 3 adversary $\mathcal{A}$ succeeding with probability ϵ can be converted into an adversary succeeding against the collision resistance-1 of F_1 or F_2 or against the one of G_1, with the same probability.*

Proof. The reduction $\mathcal{R}$ generates the public parameters and the public key of the bank as usual, and is thus able to answer any query. At the end of the experiment, the type 3 adversary outputs u transcripts that yield at least two identical serial numbers $\mathtt{SN}_i$ and $\mathtt{SN}_j$ such that

$$\begin{aligned}
\mathtt{SN}_i =& F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(F_1.\mathtt{Const}(s_i, \mathcal{S}_i), x_i) \\
&|| F_2.\mathtt{ConstEval}_{\mathcal{S}_i}(F_2.\mathtt{Const}(s_i \cdot G_1(\mathsf{upk}_i), \mathcal{S}_i), x_i) \\
\text{and}& \\
\mathtt{SN}_j =& F_1.\mathtt{ConstEval}_{\mathcal{S}_j}(F_1.\mathtt{Const}(s_j, \mathcal{S}_j), x_j) \\
&|| F_2.\mathtt{ConstEval}_{\mathcal{S}_j}(F_2.\mathtt{Const}(s_j \cdot G_1(\mathsf{upk}_j), \mathcal{S}_j), x_j)
\end{aligned}$$

The relation $\mathtt{SN}_i = \mathtt{SN}_j$ implies that $F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(F_1.\mathtt{Const}(s_i, \mathcal{S}_i), x_i) = F_1.\mathtt{ConstEval}_{\mathcal{S}_j}(F_1.\mathtt{Const}(s_j, \mathcal{S}_j), x_j)$ and hence that $s_i = s_j$ and $x_i = x_j$, otherwise $\mathcal{R}$ could be straightforwardly converted into an adversary against the collision resistance of F_1.

If we now consider the second part of the serial numbers, we have:

$$F_2.\texttt{ConstEval}_{\mathcal{S}_i}(F_2.\texttt{Const}(s_i \cdot G_1(\mathsf{upk}_i), \mathcal{S}_i), x_i) \\ = F_2.\texttt{ConstEval}_{\mathcal{S}_j}(F_2.\texttt{Const}(s_i \cdot G_1(\mathsf{upk}_j), \mathcal{S}_j), x_i)$$

We can simplify the previous relation by using the homomorphism of F_2 and thus get:

$$F_2.\texttt{ConstEval}(F_2.\texttt{Const}(G_1(\mathsf{upk}_i), \mathcal{S}_i), x_i) \\ = F_2.\texttt{ConstEval}(F_2.\texttt{Const}(G_1(\mathsf{upk}_j), \mathcal{S}_j), x_i)$$

We distinguish three cases:

– Case 1: $G_1(\mathsf{upk}_i) \neq G_1(\mathsf{upk}_j)$
– Case 2: $\mathsf{upk}_i \neq \mathsf{upk}_j$ but $G_1(\mathsf{upk}_i) = G_1(\mathsf{upk}_j)$
– Case 3 $\mathsf{upk}_i = \mathsf{upk}_j$

Case 1 implies that $\mathcal{A}$ managed to generate two master secret keys $G_1(\mathsf{upk}_i)$ and $G_1(\mathsf{upk}_j)$ such that $F_2.\texttt{Eval}(G_1(\mathsf{upk}_i), x_i) = F_2.\texttt{Eval}(G_1(\mathsf{upk}_j), x_i)$, which means that $\mathcal{A}$ can directly be converted into an adversary against the collision resistance of F_2. Similarly, case 2 implies a collision of the function G_1. Let us consider the third case. Let $R_i = H(\mathsf{mpk}_i||\mathsf{info}_i)$ and $R_j = H(\mathsf{mpk}_j||\mathsf{info}_j)$. We have:

$$F_3.\texttt{ConstEval}_{\mathcal{S}_i}(\texttt{T}_{\mathcal{S}_i}[1], x_i) \\ = F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}_i), R_i) \cdot F_3.\texttt{Const}(s_i, \mathcal{S}_i), x_i)$$

and $F_3.\texttt{ConstEval}_{\mathcal{S}_j}(\texttt{T}_{\mathcal{S}_j}[1], x_i)$

$$= F_3.\texttt{ConstEval}_{\mathcal{S}_j}(G_3(G_2(\mathsf{upk}_i), R_j) \cdot F_3.\texttt{Const}(s_i, \mathcal{S}_j), x_i)$$

Therefore,

$$F_3.\texttt{ConstEval}_{\mathcal{S}_i}(\texttt{T}_{\mathcal{S}_i}[1], x_i)/F_3.\texttt{ConstEval}_{\mathcal{S}_j}(\texttt{T}_{\mathcal{S}_j}[1], x_i) \\ = F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}_i), R_i), x_i)/F_3.\texttt{ConstEval}_{\mathcal{S}_j}(G_3(G_2(\mathsf{upk}_i), R_j), x_i)$$

However, in such a case, the `Identify` algorithm would necessarily output a public key (at least upk_i), which contradicts our assumption on $\mathcal{A}$. This third case thus cannot occur if $\mathcal{A}$ is a successful adversary against traceability. □

Remark 15. We recall that traceability only requires to output a public key in case of double-spending but does not care if this public key is correct. This latter property is ensured by exculpability. This explains why our proof of traceability does not place any requirements on the last element of the double-spending tag, nor on the functions G_2 or G_3.

A.2 Proof of Exculpability

Let $\mathcal{A}$ be a successful adversary that outputs two valid transcripts $\mathsf{Tr} = (V, \mathsf{info}, \mathsf{mpk}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}, \pi)]$, $\mathsf{Tr}' = (V', \mathsf{info}', \mathsf{mpk}', k_{\mathcal{S}_j}^{(1)}, k_{\mathcal{S}_j}^{(2)}, \mathtt{T}_{\mathcal{S}_j}, \pi')]$ accusing an honest user upk^* of double spendings, *i.e.* such that $\mathtt{Identify}(\mathsf{Tr}, \mathsf{Tr}') = \mathsf{upk}^*$.

Since upk^* is an honest user, at least one of this transcript must have been forged by the adversary. Let us assume that it is the first one. It is possible to extract from π the tuple $(\mathsf{upk}, s, c, r, \sigma_c, \mathcal{S}_i, \tau)$, otherwise $\mathcal{A}$ could be used against the extractability of Π. We then distinguish the two following cases.

– Type 1 Adversary: $\mathsf{upk} = \mathsf{upk}^*$.
– Type 2 Adversary: $\mathsf{upk} \neq \mathsf{upk}^*$.

The first case implies that the adversary has generated a valid signature under upk^*, which leads to an attack against the signature scheme Σ, as stated below.

Lemma 16. *Any type 1 adversary $\mathcal{A}$ succeeding with probability ϵ can be converted into an adversary succeeding against the EUF-CMA security of Σ with probability $\frac{\epsilon}{q_A}$, where q_A is a bound on the number of $\mathcal{O}\mathtt{Add}$ queries.*

Proof. the reduction $\mathcal{R}$ generates the public parameter as usual and receives the public key of the bank from $\mathcal{A}$. It then selects a random $i^* \xleftarrow{\$} [1, q_A]$ and answer the i-th $\mathcal{O}\mathtt{Add}$ query as follows:

– If $i \neq i^*$, then $\mathcal{R}$ generates a new key pair $(\mathsf{usk}, \mathsf{upk})$ for this user.
– Else, it returns the public key pk^* that it has received from the challenger of the EUF-CMA security experiment.

One can note that $\mathcal{R}$ may answer any query that involves $\mathsf{upk} \neq \mathsf{pk}^*$ because it knows the corresponding signing key usk. For the other ones, $\mathcal{R}$ uses its signing oracle and forwards the signature to $\mathcal{A}$.

$\mathcal{A}$ then eventually outputs a forged transcript from which it is possible to extract a valid signature τ under some public key upk. If $\mathsf{upk} \neq \mathsf{pk}^*$, then $\mathcal{R}$ aborts. Else, $\mathcal{R}$ recovers the signature τ on $(\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i})$. The elements $k_{\mathcal{S}_i}^{(1)}$, $k_{\mathcal{S}_i}^{(2)}$ and $\mathtt{T}_{\mathcal{S}_i}$ have never been part of a query to the signing oracle (otherwise the reduction would have double-spent its own coin), which means that $[(\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}), \tau]$ is a valid forgery. $\mathcal{R}$ then breaks the security of Σ if it does not abort, which occurs with probability at least $\frac{1}{q_A}$.

Lemma 17. *Any type 2 adversary can be converted into an adversary succeeding against the collision resistance-1 of F_1 or F_2, the collision resistance-2 of F_3, the collision resistance-3 of F_4 or against the one of H, G_1 or G_2, with the same probability.*

Proof. The fact that $\texttt{Identify}$ does not output 0 on inputs Tr and Tr' means that there is at least one collision between the serial numbers derived from Tr and those derived from Tr'. There are thus $x \in \mathcal{S}_i$ and $x' \in \mathcal{S}_j$ such that

$$\begin{aligned}
&(1) \quad F_1.\texttt{ConstEval}_{\mathcal{S}_i}(k^{(1)}_{\mathcal{S}_i}, x) = F_1.\texttt{ConstEval}_{\mathcal{S}_j}(k^{(1)}_{\mathcal{S}_j}, x') \\
&(2) \quad F_2.\texttt{ConstEval}_{\mathcal{S}_i}(k^{(2)}_{\mathcal{S}_i}, x) = F_2.\texttt{ConstEval}_{\mathcal{S}_j}(k^{(2)}_{\mathcal{S}_j}, x')
\end{aligned}$$

Let s and s' be the master secret keys extracted from Π and Π' respectively. Since $k^{(1)}_{\mathcal{S}_i} = F_1.\texttt{Const}(s, \mathcal{S}_i)$ and $k^{(1)}_{\mathcal{S}_j} = F_1.\texttt{Const}(s', \mathcal{S}_j)$, the relation (1) implies that $s = s'$ and that $x = x'$, otherwise (s, s', x, x') could directly be used against the collision resistance-1 of F_1.

The equation (2) thus becomes:

$$\begin{aligned}
&F_2.\texttt{ConstEval}_{\mathcal{S}_i}(F_2.\texttt{Const}(s \cdot G_1(\mathsf{upk})), \mathcal{S}_i), x) \\
&\qquad = F_2.\texttt{ConstEval}_{\mathcal{S}_j}(F_2.\texttt{Const}(s \cdot G_1(\mathsf{upk}')), \mathcal{S}_j), x)
\end{aligned}$$

which means that

$$\begin{aligned}
&F_2.\texttt{ConstEval}_{\mathcal{S}_i}(F_2.\texttt{Const}(G_1(\mathsf{upk})), \mathcal{S}_i), x) \\
&\qquad = F_2.\texttt{ConstEval}_{\mathcal{S}_j}(F_2.\texttt{Const}(G_1(\mathsf{upk}')), \mathcal{S}_j), x).
\end{aligned}$$

We then distinguish three cases.

- Case 1: $G_1(\mathsf{upk}) \neq G_1(\mathsf{upk}')$
- Case 2: $\mathsf{upk} \neq \mathsf{upk}'$ but $G_1(\mathsf{upk}) = G_1(\mathsf{upk}')$
- Case 3: $\mathsf{upk} = \mathsf{upk}'$

The first case implies that $(G_1(\mathsf{upk}), G_1(\mathsf{upk}'), x, x)$ can directly be used against the collision resistance-1 of F_2. The second case implies an obvious attack against the collision resistance of G_1. So let us consider the third case.

If $\mathcal{S}_i = \mathcal{S}_j$, we have:

$$\begin{aligned}
F_3.\texttt{ConstEval}_{\mathcal{S}_i}(\texttt{T}_{\mathcal{S}_i}[1], x) &= F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}), R) \cdot k^{(3)}_{\mathcal{S}_i}, x) \\
&= F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}), R), x) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_i}(k^{(3)}_{\mathcal{S}_i}, x) \\
&\text{and} \\
F_3.\texttt{ConstEval}_{\mathcal{S}_j}(\texttt{T}_{\mathcal{S}_j}[1], x) &= F_3.\texttt{ConstEval}_{\mathcal{S}_j}(G_3(G_2(\mathsf{upk}), R') \cdot k^{(3)}_{\mathcal{S}_j}, x) \\
&= F_3.\texttt{ConstEval}_{\mathcal{S}_j}(G_3(G_2(\mathsf{upk}), R'), x) \cdot F_3.\texttt{ConstEval}_{\mathcal{S}_j}(k^{(3)}_{\mathcal{S}_j}, x)
\end{aligned}$$

Since $k^{(3)}_{\mathcal{S}_i}$ and $k^{(3)}_{\mathcal{S}_j}$ are derived from the same master secret key s, we additionally know that $F_3.\texttt{ConstEval}_{\mathcal{S}_i}(k^{(3)}_{\mathcal{S}_i}, x) = F_3.\texttt{ConstEval}_{\mathcal{S}_j}(k^{(3)}_{\mathcal{S}_j}, x)$. During the identification process, one then gets

$$\begin{aligned}&F_3.\texttt{ConstEval}_{\mathcal{S}_i}(\mathtt{T}_{\mathcal{S}_i}[1], x)/F_3.\texttt{ConstEval}_{\mathcal{S}_j}(\mathtt{T}_{\mathcal{S}_j}[1], x)\\ &= F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}), R), x)/F_3.\texttt{ConstEval}_{\mathcal{S}_j}(G_3(G_2(\mathsf{upk}), R'), x)\\ &= F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}), R/R'), x)\end{aligned}$$

since $\mathcal{S}_i = \mathcal{S}_j$. The fact that $\mathcal{A}$ is a type 2 adversary means that:

$$F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}), R/R'), x) = F_3.\texttt{ConstEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}^*), R/R'), x)$$

with $\mathsf{upk} \neq \mathsf{upk}^*$. We can then distinguish two cases:

- $G_3(G_2(\mathsf{upk}), R/R') \neq G_3(G_2(\mathsf{upk}^*), R/R')$, which implies an attack against the collision resistance-2 of F_3.
- $G_3(G_2(\mathsf{upk}), R/R') = G_3(G_2(\mathsf{upk}^*), R/R')$ which is equivalent to $1 = G_3(G_2(\mathsf{upk})/G_2(\mathsf{upk}^*), R/R')$. Since G_3 is non degenerate, this means that $G_2(\mathsf{upk}) = G_2(\mathsf{upk}^*)$ or that $R = R'$. But $\mathsf{upk} \neq \mathsf{upk}^*$ and $(R, R') = (H(\mathsf{mpk}_i||\mathsf{info}_i), H(\mathsf{mpk}_j||\mathsf{info}_j))$ with $(\mathsf{mpk}_i||\mathsf{info}_i) \neq (\mathsf{mpk}_j||\mathsf{info}_j)$. Therefore, this second case necessarily implies a collision against G_2 or against H.

Now, let us consider the case $\mathcal{S}_i \neq \mathcal{S}_j$. The fact that the identification process returns upk^* means that

$$F_4.\texttt{ConstEval}_{\mathcal{S}_i}(T^{(1)}_{\mathcal{S}_i}[2]/G_2(\mathsf{upk}^*), x)/F_4.\texttt{ConstEval}_{\mathcal{S}_j}(T^{(1)}_{\mathcal{S}_j}[2]/G_2(\mathsf{upk}^*), x) = 1.$$

We can use the fact that $F_4.\texttt{ConstEval}_{\mathcal{S}_i}(k^{(4)}_{\mathcal{S}_i}, x) = F_4.\texttt{ConstEval}_{\mathcal{S}_j}(k^{(4)}_{\mathcal{S}_j}, x)$ to simplify the left member of the previous equation and thus get:

$$F_4.\texttt{ConstEval}_{\mathcal{S}_i}(G_2(\mathsf{upk})/G_2(\mathsf{upk}^*), x)/F_4.\texttt{ConstEval}_{\mathcal{S}_j}(G_2(\mathsf{upk})/G_2(\mathsf{upk}^*), x) = 1.$$

with $\mathsf{upk} \neq \mathsf{upk}^*$. If $G_2(\mathsf{upk})/G_2(\mathsf{upk}^*) \neq 1$, then $(i, j, G_2(\mathsf{upk})/G_2(\mathsf{upk}^*), x)$ constitutes a valid attack against the collision resistance 3 of F_4. Else, $G_2(\mathsf{upk}) = G_2(\mathsf{upk}^*)$, which implies a collision against G_2.

A.3 Proof of Anonymity

Let $\mathcal{A}$ be an adversary breaking the anonymity of our construction with probability ϵ. We define the following sequence of games to show that this advantage ϵ is negligible. For each game i, we define $\texttt{Adv}_i = |Pr(S_i) - 1/2|$, where S_i is the event that A succeeds in game i.

Game 1. Our first game is exactly the anonymity game defined in Figure 3 with a random bit b where the reduction $\mathcal{R}$ generates normally the secret values. The advantage $\texttt{Adv}_1$ is then ϵ.

Game 2. In our second game, we make a guess on the coin that will be available to user upk_b at the challenge time. More specifically, the reduction selects a random $\ell^* \in [1, q_w]$ where q_w is a bound on the number of queries and aborts if the ℓ^*-th coin issued by the bank has not been withdrawn by upk_b or if this user has entirely spent it before the challenge phase. The reduction still generates all the secret values and so the simulation is perfect unless the guess of $\mathcal{R}$ is not correct. We then have $\mathtt{Adv}_2 \geq \frac{1}{q_w}\epsilon$.

Game 3. In our third game, $\mathcal{R}$ generates a simulated common reference string using the `SimSetup` and uses the `SimProve` algorithm to output the NIZK proofs. Any change in the behaviour of $\mathcal{A}$ in this game can then be used against the zero-knowledge property of Π. We then have $\mathtt{Adv}_3 \geq \mathtt{Adv}_2 - \mathtt{Adv}_{\mathcal{A}}^{\mathtt{zk}}$.

Game 4. In our fourth game, $\mathcal{R}$ proceeds as in the previous game except that during the ℓ^*-th withdrawal query it replaces the commitment c by a random element from the commitment space. We note that this is not a problem for further NIZK proofs since the `SimProve` algorithm does not need witnesses as inputs. Any significant change in the behaviour of $\mathcal{A}$ then yields an adversary against the hiding property, which means that $\mathtt{Adv}_4 \geq \mathtt{Adv}_3 - \mathtt{Adv}_{\mathcal{A}}^{\mathtt{hid}}$.

Game 5. In our fifth game, $\mathcal{R}$ proceeds as in the previous game, but it answers the challenge query $(V, \mathsf{upk}_0, \mathsf{upk}_1, \mathsf{mpk})$ as follows. First, it selects a subset $\mathcal{S}$ containing V elements that have never been involved in previous spendings of the ℓ^*-th coin. We recall that Game 2 ensures that such subset exists, otherwise $\mathcal{R}$ would abort. Moreover, this game also ensures that this coin belongs to upk_0 or upk_1. Let us denote this user by upk^*. $\mathcal{R}$ then selects four random values $(k_1, k_2, k_3, k_4) \xleftarrow{\$} \mathcal{K}_{\mathcal{S}}^4$ and returns $(k_{\mathcal{S}}^{(1)}, k_{\mathcal{S}}^{(2)}) = (k_1, k_2 \cdot F_2.\mathtt{Const}(G_1(\mathsf{upk}^*), \mathcal{S}))$ along with $\mathtt{T}_{\mathcal{S}} = (G_3(G_2(\mathsf{upk}^*), H(\mathsf{mpk}||\mathsf{info})) \cdot k_3, G_2(\mathsf{upk}^*) \cdot k_4)$. Here again, the NIZK proofs can still be produced by using the `SimProve` algorithm. The simulation is perfect unless $\mathcal{A}$ is able to distinguish the random tuple (k_1, k_2, k_3, k_4) from $(F_1.\mathtt{Const}(s, \mathcal{S}), F_2.\mathtt{Const}(s, \mathcal{S}), F_3.\mathtt{Const}(s, \mathcal{S}), F_4.\mathtt{Const}(s, \mathcal{S}))$, which would yield an adversary against the combined key-pseudorandomness of the family (F_1, F_2, F_3, F_4). We therefore have $\mathtt{Adv}_5 \geq \mathtt{Adv}_4 - \mathtt{Adv}_{\mathcal{A}}^{ckps}$.

We can note that in Game 5, the values received by the adversary are simulated proofs along with random elements $(k_{\mathcal{S}}^{(1)}, k_{\mathcal{S}}^{(2)})$ and $\mathtt{T}_{\mathcal{S}}$. The advantage $\mathtt{Adv}_5$ of $\mathcal{A}$ in this game is then 0. This means that:

$$\frac{\epsilon}{q_w} \leq \mathtt{Adv}_{\mathcal{A}}^{\mathtt{zk}} + \mathtt{Adv}_{\mathcal{A}}^{\mathtt{hid}} + \mathtt{Adv}_{\mathcal{A}}^{ckps}$$

which concludes the proof since q_w is polynomial in the security parameter λ.

A.4 Proof of Clearability

An adversary succeeding in the clearing security game with probability ϵ is able to output a valid proof $[\mathsf{Tr}, \mu]$ of deposit of a transcript $\mathsf{Tr} = (V, \mathsf{info}, \mathsf{mpk}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)},$

$\mathtt{T}_{\mathcal{S}_i}, \pi$) that has not been generated by the merchant mpk. Informally, since the $\mathtt{CheckDeposit}$ algorithm mostly consists in verifying the signature μ, this means that $\mathcal{A}$ has managed to forge the latter and so that $\mathcal{A}$ can be used against the security of the signature scheme.

Proof. The reduction $\mathcal{R}$ generates all the parameters as usual but selects a random $\ell^* \in [1, q_a]$ where q_a is a bound on the number of $\mathcal{O}\mathtt{Add}$ queries for merchants. In the simulation, it proceeds as usual except that it answers the ℓ^*-th $\mathcal{O}\mathtt{Add}$ query by returning the public key mpk^* received from the challenger $\mathcal{C}$ of the EUF-CMA security experiment for Σ. Each time mpk^* is involved in a query to the $\mathcal{O}\mathtt{Deposit}$ oracle, $\mathcal{R}$ simply sends the corresponding transaction transcript Tr_i to $\mathcal{C}$ and then forwards the signature μ_i to the adversary.

At the end of the game, $\mathcal{A}$ returns $[\mathsf{Tr}, \mu]$ involving an honest merchant mpk. If $\mathsf{mpk} \neq \mathsf{mpk}^*$, then $\mathcal{R}$ aborts. Else, the fact that $\mathcal{A}$ succeeds means that $\mathcal{O}\mathtt{Deposit}$ has never been queried on this transaction and so that Tr has never been sent to the signing oracle provided by $\mathcal{C}$. This means that μ is a valid forgery that $\mathcal{R}$ can forward to $\mathcal{C}$.

$\mathcal{R}$ then outputs a valid forgery unless it aborts. It then succeeds against the EUF-CMA security of Σ with probability at least $\frac{\epsilon}{q_a}$.

B Additional Security Notion for Delegatable PRF

Strong Key-Pseudorandomness. In the case where F is a delegatable constrained PRF, we will need a variant of the key-pseudorandomness property that we call strong key-pseudorandomness. The latter requires that key-pseudorandomness hold even if some information leaks during the delegation process. For sake of simplicity, in this context, we will only consider a family of subsets $\{\mathcal{S}_i\}$ of $\mathcal{S}$ such that:

$$\mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset \Rightarrow \mathcal{S}_i \subset \mathcal{S}_j \text{ or } \mathcal{S}_j \subset \mathcal{S}_i$$

We can therefore define the function D which returns for each subset $\mathcal{S}_i \neq \mathcal{S}$ the index $j \in [1, n]$ of the smallest subset containing (strictly) $\mathcal{S}_i$. We will define our leakage by introducing a family of functions $H, \{H_i\}_{i=1}^n$ such that $H : \mathcal{K} \to \mathbb{G}$ and $H_i : \mathcal{K}_{\mathcal{S}_i} \to \mathbb{G}$ for some group $\mathbb{G}$. In our new experiment, when $b = 0$, the adversary still receives the constrained key $k_{\mathcal{S}_i}$ for the subset $\mathcal{S}_i$ of its choice, but now it additionally receives the evaluation of $H_{D(\mathcal{S}_i)}$ on the constrained key $k_{D(\mathcal{S}_i)}$ of its parent subset[15]. We require this pair to be indistinguishable from a random pair generated from the corresponding domains. We note that this property is achieved by any key-pseudorandom delegatable PRF when H and $\{H_i\}_{i=1}^n$ are random oracles. We will also show in Appendix E that it possible to construct such PRFs and functions H and $\{H_i\}_{i=1}^n$ in the standard model.

A function F achieves strong key-pseudorandomness for a family of function $H, \{H_i\}_{i=1}^n$ if $\mathtt{Adv}^{skps}(\mathcal{A}) = |\Pr[\mathtt{Exp}_{\mathcal{A}}^{skps-1}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n, \{H_i\}_{i=1}^n) = 1] -$

[15] the case $\mathcal{S}_i = \mathcal{S}$ is addressed separately.

$\Pr[\mathtt{Exp}_{\mathcal{A}}^{skps-0}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n, \{H_i\}_{i=1}^n) = 1]|$ is negligible for any $\mathcal{A}$, where the game $\mathtt{Exp}_{\mathcal{A}}^{skps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n, \{H_i\}_{i=1}^n)$ is defined in Figure 7.

Finally, we define the *strong combined key-pseudorandomness* where the adversary receives a tuple of constrained keys $(k_0^1, \dots, k_0^t) \leftarrow (F_1.\mathtt{Const}(s, \mathcal{S}_{i^*}), \dots, F_t.\mathtt{Const}(s, \mathcal{S}_{i^*}))$ in addition to $H_{D(\mathcal{S}_i^*)}(\mathtt{Const}(s, \mathcal{S}_{D(\mathcal{S}_{i^*})}))$, instead of just one constrained key.

```
$\mathtt{Exp}_{\mathcal{A}}^{skps-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n, H, \{H_i\}_{i=1}^n)$ – Strong Key-Pseudorandomness
 1. $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$
 2. $s \leftarrow \mathtt{Keygen}()$
 3. $i^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const}, \mathcal{O}\mathtt{Eval}}(pp)$
 4. If $\mathcal{S}_{i^*} = \mathcal{S}$
 5.     $(k_0^0, k_0^1) \leftarrow (\mathtt{Const}(s, \mathcal{S}), H(s))$
 6.     $(k_1^0, k_1^1) \xleftarrow{\$} \mathcal{K} \times \mathbb{G}$
 7. else
 8.     $(k_0^0, k_0^1) \leftarrow (\mathtt{Const}(s, \mathcal{S}_{i^*}), H_{D(\mathcal{S}_i^*)}(\mathtt{Const}(s, \mathcal{S}_{D(\mathcal{S}_{i^*})})))$
 9.     $(k_1^0, k_1^1) \xleftarrow{\$} \mathcal{K}_{\mathcal{S}_{i^*}} \times \mathbb{G}$
10. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Const}, \mathcal{O}\mathtt{Eval}}(pp, H, \{H_i\}_{i=1}^n, (k_b^0, k_b^1))$
11. If $\mathcal{O}\mathtt{Eval}$ was queried on $x \in \mathcal{S}_{i^*}$ return 0
12. If $\mathcal{O}\mathtt{Const}$ was queried on $\mathcal{X}$ such that $\mathcal{X} \cap \mathcal{S}_{i^*}$, return 0.
13. Return $b^*$
```

Fig. 7. Strong Key-Pseudorandomness

C Framework based on Delegatable PRFs

As in Section 5.2, our construction makes uses of a digital signature scheme Σ, an homomorphic commitment scheme Γ and a NIZK proof system Π.

- $\mathtt{Setup}(1^\lambda, N)$: To generate the public parameters pp, the algorithm first computes $\mathtt{crs} \leftarrow \Pi.\mathtt{Setup}(1^\lambda)$. It then selects two constrained PRFs F_1, F_2 with the same master key space $\mathcal{K}$ and that support the same subsets $\mathcal{S}_1, \dots, \mathcal{S}_n$ with $\mathcal{S}_i \subset [1, N]$ $\forall i \in [1, n]$. For sake of simplicity, we assume that $\mathcal{K}_{\mathcal{S}_i} = \mathcal{K}_{\mathcal{S}_j} = \mathcal{K}_{\mathcal{S}}$ for all $i, j \in [1, n]$. As we describe in 4.2, the algorithm also defines two functions (H, H') such that $H : \mathcal{K} \to \mathbb{G}_1$ and $H' : \mathcal{K}_{\mathcal{S}_i} \to \mathbb{G}_1$ for some group $\mathbb{G}_1$. We recall that our family of subsets has the following property:

$$\mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset \Rightarrow \mathcal{S}_i \subset \mathcal{S}_j \text{ or } \mathcal{S}_j \subset \mathcal{S}_i$$

  Finally, it selects a hash function $H_0 : \{0,1\}^* \to \mathbb{G}_2$ for some group $\mathbb{G}_2$, a function $G_1 : \{0,1\}^* \to \mathcal{K}_{\mathcal{S}}$, $G_2 : \{0,1\}^* \to \mathbb{G}_1$ along with a non degenerate bilinear map $G_3 : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_1$. The public parameters pp are then set as $\mathtt{crs}, F_1, F_2, H, H', H_0, G_1, G_2, G_3$.

- $\texttt{BKeygen}()$: The bank generates a commitment key $\mathsf{ck} \leftarrow \mathit{\Gamma}.\texttt{Keygen}(1^\lambda)$ and a key pair $(\mathsf{sk}_B, \mathsf{pk}_B) \leftarrow \mathit{\Sigma}.\texttt{Keygen}(1^\lambda)$. It then sets bsk as sk_B and bpk as $(\mathsf{ck}, \mathsf{pk}_B)$.
- $\texttt{Keygen}()$: The user (resp. the merchant) generates a signature key pair $(\mathsf{usk}, \mathsf{upk})$ (resp. $(\mathsf{msk}, \mathsf{mpk})$) using $\mathit{\Sigma}.\texttt{Keygen}$.
- $\texttt{Withdraw}(\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk}))$: To withdraw a divisible coin, the user first generates $s \leftarrow F_1.\texttt{Keygen}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ and a random element r from the randomizer space $\mathcal{R}$ of $\mathit{\Gamma}$. It then sends $c \leftarrow \mathit{\Gamma}.\texttt{Commit}(\mathsf{ck}, [s, \mathsf{upk}], r)$ to the bank along with a signature $\tau_c \leftarrow \mathit{\Sigma}.\texttt{Sign}(\mathsf{usk}, c)$.
  If τ_c is valid, the bank returns a signature $\sigma_c \leftarrow \mathit{\Sigma}.\texttt{Sign}(\mathsf{sk}_B, c)$ to the user. The latter can then set its coin C as (c, s, r, σ_c).
- $\texttt{Spend}(\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, V), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, \mathsf{info}, V))$: During a spending of amount V, the merchant first selects a string info that he never used before and sends it to the user along with his public key mpk.
  The user then selects an unspent subset $\mathcal{S}_i$ of V elements, *i.e.* one such that $\texttt{SN}_j$ has never been revealed for all $j \in \mathcal{S}_i$, and computes
  1. $k_{\mathcal{S}_i}^{(1)} \leftarrow F_1.\texttt{Const}(s, \mathcal{S}_i)$
  2. $k_{\mathcal{S}_i}^{(2)} \leftarrow G_1(\mathsf{upk}) \cdot F_2.\texttt{Const}(s, \mathcal{S}_i)$
  3. $\texttt{T}_{\mathcal{S}_i} \leftarrow G_3(G_2(\mathsf{upk}), H_0(\mathsf{mpk}||\mathsf{info})) \cdot H'(F_1.\texttt{Const}(s, \mathcal{S}_{D(\mathcal{S}_i)}))$

  if $\mathcal{S}_i \neq [1, N]$ and
  1. $k_{\mathcal{S}_i}^{(1)} \leftarrow F_1.\texttt{Const}(s, \mathcal{S}_i)$
  2. $k_{\mathcal{S}_i}^{(2)} \leftarrow G_1(\mathsf{upk}) \cdot F_2.\texttt{Const}(s, \mathcal{S}_i)$
  3. $\texttt{T}_{\mathcal{S}_i} \leftarrow G_3(G_2(\mathsf{upk}), H_0(\mathsf{mpk}||\mathsf{info})) \cdot H(s)$

  otherwise.
  Finally, it generates a signature $\tau \leftarrow \mathit{\Sigma}.\texttt{Sign}(\mathsf{usk}, (\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \texttt{T}_{\mathcal{S}_i}))$ along with a NIZK proof π of $(\mathsf{upk}, s, c, r, \sigma_c, \mathcal{S}_i, \tau)$ such that:
  1. $\exists i^* \in [1, n] : \mathcal{S}_i = \mathcal{S}_{i^*} \wedge |\mathcal{S}_i| = V$
  2. $c = \mathit{\Gamma}.\texttt{Commit}(\mathsf{ck}, [s, \mathsf{upk}], r)$
  3. $1 = \mathit{\Sigma}.\texttt{Verify}(\mathsf{pk}_B, c, \sigma_c)$
  4. $k_{\mathcal{S}_i}^{(1)} = F_1.\texttt{Const}(s, \mathcal{S}_i)$
  5. $k_{\mathcal{S}_i}^{(2)} \leftarrow G_1(\mathsf{upk}) \cdot F_2.\texttt{Const}(s, \mathcal{S}_i)$
  6. if $\mathcal{S}_i \neq [1, N]$, $\texttt{T}_{\mathcal{S}_i} \leftarrow G_3(G_2(\mathsf{upk}), H_0(\mathsf{mpk}||\mathsf{info})) \cdot H'(F_1.\texttt{Const}(s, \mathcal{S}_{D(\mathcal{S}_i)}))$
  7. else, $\texttt{T}_{\mathcal{S}_i} \leftarrow G_3(G_2(\mathsf{upk}), H_0(\mathsf{mpk}||\mathsf{info})) \cdot H(s)$
  8. $1 = \mathit{\Sigma}.\texttt{Verify}(\mathsf{upk}, (\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \texttt{T}_{\mathcal{S}_i}), \tau)$

  The elements $(k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \texttt{T}_{\mathcal{S}_i}, \pi)$ are then sent to the merchant who accepts them as a payment if π is valid.
- $\texttt{Deposit}(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \texttt{T}_{\mathcal{S}_i}, \pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: To deposit a transaction, the merchant sends its transcript $\mathsf{Tr} \leftarrow (V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \texttt{T}_{\mathcal{S}_i}, \pi)$ along with a signature $\mu \leftarrow \mathit{\Sigma}.\texttt{Sign}(\mathsf{msk}, \mathsf{Tr})$. The bank then checks that (1) the proof π is valid, (2) π proves knowledge of a signature on a tuple whose first coordinate is mpk, (3) $\mathit{\Sigma}.\texttt{Verify}(\mathsf{mpk}, \mathsf{Tr}, \mu) = 1$ and (4) that this merchant has not previously deposited a transaction associated with info. If one of the first three conditions is not satisfied, then the bank returns $\perp$. If the last condition is not satisfied then the bank knows another

transcript $(V', \mathsf{info}, k^{(1)}_{\mathcal{S}_j}, k^{(2)}_{\mathcal{S}_j}, \mathtt{T}_{\mathcal{S}_j}, \pi')$ along with a signature μ'. All these elements, along with $[\mathsf{Tr}, \mu]$ constitute a proof of double-deposit.

Else, the bank recovers the serial numbers $\mathtt{SN}_j \leftarrow F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(k^{(1)}_{\mathcal{S}_i}, j)$ for all $j \in \mathcal{S}_i$. It then distinguishes the following two cases:

- $\mathtt{SN}_j \notin L \; \forall j \in \mathcal{S}_i$, in which case the bank simply adds these serial numbers to L
- $\exists j^* \in \mathcal{S}_i$ such that $\mathtt{SN}_{j^*}$ already belongs to L. In such a case, the bank recovers the first transcript $\mathsf{Tr}' = (V', \mathsf{info}', \mathsf{mpk}', k^{(1)}_{\mathcal{S}_{i'}}, k^{(2)}_{\mathcal{S}_{i'}}, \mathtt{T}_{\mathcal{S}_{i'}}, \pi')$ that yields this serial number. If $\mathcal{S}_i \neq \mathcal{S}_{i'}$, then the bank returns this transcript, along with Tr. If $\mathcal{S}_i = \mathcal{S}_{i'}$ and $k^{(2)}_{\mathcal{S}_i} = k^{(2)}_{\mathcal{S}_{i'}}$, then the bank also returns $[\mathsf{Tr}, \mathsf{Tr}']$. Else, it does not return anything.

– $\mathtt{Identify}((V, \mathsf{info}, \mathsf{mpk}, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi), (V', \mathsf{info}', \mathsf{mpk}', k^{(1)}_{\mathcal{S}_j}, k^{(2)}_{\mathcal{S}_j}, \mathtt{T}_{\mathcal{S}_j}, \pi'), \mathsf{bpk})$: Given two transcripts, this algorithm first checks that (1) $\mathsf{mpk}||\mathsf{info} \neq \mathsf{mpk}'||\mathsf{info}'$ and (2) both proofs π and π' are valid. If one of these conditions is not satisfied, then it returns 0. Else, it checks that there is a collision between the serial numbers derived from these transcripts, *i.e.* there are $x \in \mathcal{S}_i$ and $x' \in \mathcal{S}_j$ such that $F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(k^{(1)}_{\mathcal{S}_i}, x) = F_1.\mathtt{ConstEval}_{\mathcal{S}_j}(k^{(1)}_{\mathcal{S}_j}, x')$. If there is no collision or if $x_i \neq x_j$, it outputs 0. If there is a collision with $\mathcal{S}_i = \mathcal{S}_j$ but $k^{(2)}_{\mathcal{S}_i} \neq k^{(2)}_{\mathcal{S}_j}$, then it outputs 0.

The fact that $x_i \in \mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset$ implies that $\mathcal{S}_i \subset \mathcal{S}_j$ or $\mathcal{S}_j \subset \mathcal{S}_i$. Let us assume that $\mathcal{S}_j \subset \mathcal{S}_i$. We then distinguish the two following cases.

- Case 1: $\mathcal{S}_j = \mathcal{S}_i$ (which is equivalent to $V = V'$). In such a case, it computes $\mathtt{T}_{\mathcal{S}_i}/\mathtt{T}_{\mathcal{S}_j}$ and $G_3(G_2(\mathsf{upk}), R/R')$ for all public key upk, with $R = H(\mathsf{mpk}||\mathsf{info})$ and $R' = H(\mathsf{mpk}'||\mathsf{info}')$. If it gets a match, then it returns the corresponding public upk. Else, it returns $\perp$.
- Case 2: $\mathcal{S}_j \subsetneq \mathcal{S}_i$, which implies that $\mathcal{S}_{D(\mathcal{S}_j)} \subset \mathcal{S}_i$. From $k^{(1)}_{\mathcal{S}_i}$, this algorithm computes $\mathtt{T}^* \leftarrow H'(F_1.\overline{\mathtt{Const}}(k^{(1)}_{\mathcal{S}_i}, \mathcal{S}_{D(\mathcal{S}_j)}))$ and $\mathtt{T} = \mathtt{T}_{\mathcal{S}_j}/(\mathtt{T}^*)$. It then compares T with $G_3(G_2(\mathsf{upk}), R')$ for all public key upk. If it gets a match, then it returns the corresponding public upk. Else, it returns $\perp$.

– $\mathtt{CheckDeposit}([(V, \mathsf{info}, \mathsf{mpk}, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi), \mu], \mathsf{bpk})$: this algorithm checks that π is valid and that $1 = \Sigma.\mathtt{Verify}(\mathsf{mpk}, (V, \mathsf{info}, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi), \mu)$ in which case it outputs 1. Else, it returns 0.

The security of our construction is stated by the following theorem.

Theorem 18. *Our divisible e-cash system is*

– *traceable if F_1 achieves collision resistance-1, Γ is computationally binding, Σ is EUF-CMA secure and Π is extractable.*
– *exculpable if Σ is EUF-CMA secure, Π is extractable, F_1 achieves collision resistance-1 and H_0, G_1 and G_2 are collision resistant.*
– *clearable if Σ is EUF-CMA secure.*
– *anonymous if (F_1, F_2) achieves strong combined key-pseudorandomness for the family (H, H'), Γ is computationally hiding and Π is zero-knowledge.*

D Proof of Theorem 18

All the proofs provided here are very similar to those provided in Appendix A. We therefore mostly focus on the parts that involve double-spending tags because these elements strongly differ from a framework to another. In particular, we do not describe a proof for clearability since the one for our previous framework still works here.

D.1 Proof of Traceability

A successful adversary $\mathcal{A}$ against the traceability is able to spend more than it has withdrawn without being identified. Formally, this means that there are u transcripts $\mathsf{Tr}_i = \{(V_i, \mathsf{info}_i, \mathsf{mpk}_i, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi_i)\}_{i=1}^{u}$ where:

- $\sum_{i=1}^{u} V_i > m.N$, with m the number of withdrawn coins
- $\mathtt{Identify}(\mathsf{Tr}_i, \mathsf{Tr}_j) = \perp\ \forall i, j \in [1, u]$

The latter condition implies that all the proofs π_i are valid (otherwise $\mathtt{Identify}$ would have returned 0 and not $\perp$). Therefore, the challenger is able to extract from them tuples $W_i = (\mathsf{upk}_i, s_i, r_i, \sigma_i, c_i, \mathcal{S}_i, \tau_i)$ satisfying the 7 relations defined by the $\mathtt{Spend}$ algorithm. Else, $\mathcal{A}$ could be trivially converted into an adversary against the extractability of Π.

We then distinguish the following cases:

- Type 1 Adversary: $\exists i \in [1, u]$ such that none of the coins generated through $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ queries contains the commitment c_i.
- Type 2 Adversary: all the commitments c_i have been generated during a query to the $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ oracle but the serial numbers do not collide.
- Type 3 Adversary: all the commitments c_i have been generated during a query to the $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ oracle and there is a collision in the list of serial numbers.

The first two types of adversary essentially imply an attack against the signature scheme or the commitment scheme. The security proofs are exactly the same as the one provided in Appendix A. We therefore only consider the third type of adversary.

Lemma 19. *Any type 3 adversary $\mathcal{A}$ succeeding with probability ϵ can be converted into an adversary succeeding against the collision resistance-1 of F_1.*

Proof. The reduction $\mathcal{R}$ generates the public parameters and the public key of the bank as usual, and is thus able to answer any query. At the end of the experiment, the type 3 adversary outputs u transcripts that yield at least two identical serial numbers. Among the colliding serial numbers, there is at least a pair $(\mathtt{SN}_i, \mathtt{SN}_j)$, with $\mathtt{SN}_i = \mathtt{SN}_j$, derived from $\mathsf{Tr}_i = (V_i, \mathsf{info}_i, \mathsf{mpk}_i, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi_i)$ and $\mathsf{Tr}_j = (V_j, \mathsf{info}_j, \mathsf{mpk}_j, k^{(1)}_{\mathcal{S}_j}, k^{(2)}_{\mathcal{S}_j}, \mathtt{T}_{\mathcal{S}_j}, \pi_j)$ such that $\mathcal{S}_i \neq \mathcal{S}_j$ or such that $\mathcal{S}_i = \mathcal{S}_j \wedge k^{(2)}_{\mathcal{S}_i} = k^{(2)}_{\mathcal{S}_j}$.

Indeed, let us extract for $t \in [1,u]$ and $x \in \mathcal{S}_t$ the pairs $(\mathtt{SN}_{t,x}, \mathsf{upk}_t)$ from the transcripts $\mathsf{Tr}_t = (V_t, \mathsf{info}_t, \mathsf{mpk}_t, k^{(1)}_{\mathcal{S}_t}, k^{(2)}_{\mathcal{S}_t}, \mathtt{T}_{\mathcal{S}_t}, \pi_t)$, *i.e.* the V_t serial numbers and the user's public key associated with each transaction. Since we here consider a type-3 adversary, all these transcripts involve at most m distinct pairs (s_k, upk_k). There are thus at most $m.N$ distinct pairs $(F_1.\mathtt{Eval}(s_k, x), \mathsf{upk}_k)$ for $x \in [1,N]$. We then have:

$$\{(\mathtt{SN}_{t,x}, \mathsf{upk}_t)\}_{t\in[1,u], x\in\mathcal{S}_t} \subset \{(F_1.\mathtt{Eval}(s_k,x), \mathsf{upk}_k)\}_{k\in[1,m]}, x \in [1,N]$$

Since $\sum_{i=1}^{u} V_i > m{\cdot}N$, there are at least two indices i, j such that $(\mathtt{SN}_{i,x_i}, \mathsf{upk}_i) = (\mathtt{SN}_{j,x_j}, \mathsf{upk}_j)$. Moreover, we necessarily have $i \neq j$, otherwise two serial numbers derived from the same transcript would collide, leading to an obvious attack against the collision resistance of F_1. Therefore, these serial numbers are derived from two different transcripts Tr_i and Tr_j involving respectively a subset $\mathcal{S}_i$ and $\mathcal{S}_j$. If $\mathcal{S}_i \neq \mathcal{S}_j$, we are done. Else, we recall that

$$\begin{aligned}&\mathtt{SN}_i = F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(F_1.\mathtt{Const}(s_i, \mathcal{S}_i), x_i)\\ &\text{and}\\ &\mathtt{SN}_j = F_1.\mathtt{ConstEval}_{\mathcal{S}_j}(F_1.\mathtt{Const}(s_j, \mathcal{S}_j), x_j).\end{aligned}$$

Therefore, the relation $\mathtt{SN}_i = \mathtt{SN}_j$ implies that $s_i = s_j$ (and so that $\mathsf{upk}_i = \mathsf{upk}_j$) and $x_i = x_j$, otherwise $\mathcal{R}$ could be straightforwardly converted into an adversary against the collisions resistance of F_1. Moreover we consider here the case $\mathcal{S}_i = \mathcal{S}_j$, which means that $k^{(2)}_{\mathcal{S}_i} = k^{(2)}_{\mathcal{S}_j}$ and thus concludes the proof.

We can then only consider the two following cases:

- Case 1: $\mathcal{S}_i = \mathcal{S}_j \wedge k^{(2)}_{\mathcal{S}_i} = k^{(2)}_{\mathcal{S}_j}$.
- Case 2: $\mathcal{S}_i \neq \mathcal{S}_j$.

Case 1. As we explain, this case implies $s_i = s_j$, $x_i = x_j$ and $\mathsf{upk}_i = \mathsf{upk}_j = \mathsf{upk}$. Therefore, $\mathtt{T}_{\mathcal{S}_i}$ and $\mathtt{T}_{\mathcal{S}_j}$ are of the form $G_3(G_2(\mathsf{upk}), R_i) \cdot \mathtt{T}^*$ and $G_3(G_2(\mathsf{upk}), R_j) \cdot \mathtt{T}^*$ for some element $\mathtt{T}^* \in \mathbb{G}_1$. Computing $\mathtt{T}_{\mathcal{S}_i}/\mathtt{T}_{\mathcal{S}_j}$ then gives $G_3(G_2(\mathsf{upk}), R_i/R_j)$. However in such a case, $\mathtt{Identify}$ would have at least returned upk.

Case 2. Here again, the collision between the serial numbers implies that $s_i = s_j$ and $x_i = x_j = x$. In particular, we have $x \in \mathcal{S}_i \cap \mathcal{S}_j$, which implies that $\mathcal{S}_i \subsetneq \mathcal{S}_j$ or that $\mathcal{S}_j \subsetneq \mathcal{S}_i$. We assume, without loss of generality, that the latter situation occurs. We then have $\mathcal{S}_{D(\mathcal{S}_j)} \subset \mathcal{S}_i$. From $k^{(1)}_{\mathcal{S}_i}$, this algorithm computes $\mathtt{T}^* \leftarrow H'(F_1.\overline{\mathtt{Const}}(k^{(1)}_{\mathcal{S}_i}, \mathcal{S}_{D(\mathcal{S}_j)}))$ and $\mathtt{T}_{\mathcal{S}_j}/(\mathtt{T}^*) = G_3(G_2(\mathsf{upk}_j), R_j)$. However, in this case $\mathtt{Identify}$ would have at least returned upk_j.

Therefore, if F_1 achieves collision resistance-1, then no adversary can succeed against the traceability of our scheme. □

D.2 Proof of Exculpability

Let $\mathcal{A}$ be a successful adversary that outputs two valid transcripts $\mathsf{Tr}_i = (V_i, \mathsf{info}_i, \mathsf{mpk}_i, k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i}, \pi_i)]$, $\mathsf{Tr}_j = (V_j, \mathsf{info}_j, \mathsf{mpk}_j, k^{(1)}_{\mathcal{S}_j}, k^{(2)}_{\mathcal{S}_j}, \mathtt{T}_{\mathcal{S}_j}, \pi_j)]$ accusing an honest user upk^* of double spendings, *i.e.* such that $\mathtt{Identify}(\mathsf{Tr}_i, \mathsf{Tr}_j) = \mathsf{upk}^*$.

Since upk^* is an honest user, at least one of this transcript must have been forged by the adversary. Let us assume that it is the first one. It is possible to extract from π_i the tuple $(\mathsf{upk}, s, c, r, \sigma_c, \mathcal{S}_i, \tau)$, otherwise $\mathcal{A}$ could be used against the extractability of Π. We then distinguish the two following cases.

– Type 1 Adversary: $\mathsf{upk} = \mathsf{upk}^*$.
– Type 2 Adversary: $\mathsf{upk} \neq \mathsf{upk}^*$.

The first case implies that the adversary has generated a valid signature under upk^*, which leads to an attack against the signature scheme Σ. The proof is exactly the same as the one provided in Appendix A so here we only consider type 2 adversary.

Lemma 20. *Any type 2 adversary can be converted into an adversary succeeding against the collision resistance-1 of F_1 or against the collision resistance of H_0, G_1 or G_2, with the same probability.*

Proof. The fact that $\mathtt{Identify}$ does not output $\perp$ on inputs Tr_i and Tr_j means that there is at least one collision between the serial numbers derived from these transcripts. There are thus $x_i \in \mathcal{S}_i$ and $x_j \in \mathcal{S}_j$ such that

$$\mathtt{SN}_i = F_1.\mathtt{ConstEval}_{\mathcal{S}_i}(F_1.\mathtt{Const}(s_i, \mathcal{S}_i), x_i)$$
and
$$\mathtt{SN}_j = F_1.\mathtt{ConstEval}_{\mathcal{S}_j}(F_1.\mathtt{Const}(s_j, \mathcal{S}_j), x_j).$$

By using the same argument as above, we know that $s_i = s_j$ and that $x_i = x_j = x$ if F_1 achieves collision resistance-1. Moreover we know that one of the following two cases necessarily occurs.

– Case 1: $\mathcal{S}_i = \mathcal{S}_j \wedge k^{(2)}_{\mathcal{S}_i} = k^{(2)}_{\mathcal{S}_j}$.
– Case 2: $\mathcal{S}_i \neq \mathcal{S}_j$.

Case 1. The equalities $k^{(2)}_{\mathcal{S}_i} = k^{(2)}_{\mathcal{S}_j}$ and $s_i = s_j$ imply that Tr_i and Tr_j have been generated by the same user upk. Moreover, it implies that $\mathtt{T}_{\mathcal{S}_i} = G_3(G_2(\mathsf{upk}), R_i) \cdot (\mathtt{T}^*)$ and $\mathtt{T}_{\mathcal{S}_j} = G_3(G_2(\mathsf{upk}), R_j) \cdot (\mathtt{T}^*)$ for some element $\mathtt{T}^* \in \mathbb{G}_1$. Therefore, $\mathtt{T}_{\mathcal{S}_i}/\mathtt{T}_{\mathcal{S}_j} = G_3(G_2(\mathsf{upk}), R_i/R_j)$. We additionally know that $\mathtt{Identify}$ returns upk^*, which means that:

$$G_3(G_2(\mathsf{upk}), R_i/R_j) = G_3(G_2(\mathsf{upk}^*), R_i/R_j)$$

Since G_3 is non degenerate, this is only possible if $R_i = R_j$ or if $G_2(\mathsf{upk}) = G_2(\mathsf{upk}^*)$. The former case implies a collision of H_0, whereas the second one implies a collision of G_2.

Case 2. By using the same argument as in the traceability proof, we know that $\mathtt{T}_{\mathcal{S}_j}/(\mathtt{T}^*) = G_3(G_2(\mathsf{upk}_j), R_j)$, where upk_j is the public key involved in the transaction Tr_j. Therefore, the fact that `Identify` returns upk^* means that

$$G_3(G_2(\mathsf{upk}_j), R_j) = G_3(G_2(\mathsf{upk}^*), R_j)$$

which implies a collision of G_2.

D.3 Proof of Anonymity

Let $\mathcal{A}$ be an adversary breaking the anonymity of our construction with probability ϵ. We define the following sequence of games to show that this advantage ϵ is negligible. For each game i, we define $\mathtt{Adv}_i = |Pr(S_i) - 1/2|$, where S_i is the event that A succeeds in game i.

Game 1. Our first game is exactly the anonymity game defined in Figure 3 with a random bit b where the reduction $\mathcal{R}$ generates normally the secret values. The advantage $\mathtt{Adv}_1$ is then ϵ.

Game 2. In our second game, we make a guess on the coin that will be available to user upk_b at the challenge time. More specifically, the reduction selects a random $\ell^* \in [1, q_w]$ where q_w is a bound on the number of queries and aborts if the ℓ^*-th coin issued by the bank has not been withdrawn by upk_b or if this user has entirely spent it before the challenge phase. The reduction still generates all the secret values and so the simulation is perfect unless the guess of $\mathcal{R}$ is not correct. We then have $\mathtt{Adv}_2 \geq \frac{1}{q_w}\epsilon$.

Game 3. In our third game, $\mathcal{R}$ generates a simulated common reference string using the `SimSetup` and uses the `SimProve` algorithm to output the NIZK proofs. Any change in the behaviour of $\mathcal{A}$ in this game can then be used against the zero-knowledge property of Π. We then have $\mathtt{Adv}_3 \geq \mathtt{Adv}_2 - \mathtt{Adv}_{\mathcal{A}}^{\mathtt{zk}}$.

Game 4. In our fourth game, $\mathcal{R}$ proceeds as in the previous game except that during the ℓ^*-th withdrawal query it replaces the commitment c by a random element from the commitment space. We note that this is not a problem for further NIZK proofs since the `SimProve` algorithm does not need witnesses as inputs. Any significant change in the behaviour of $\mathcal{A}$ then yields an adversary against the hiding property, which means that $\mathtt{Adv}_4 \geq \mathtt{Adv}_3 - \mathtt{Adv}_{\mathcal{A}}^{\mathtt{hid}}$.

Game 5. In our fifth game, $\mathcal{R}$ proceeds as in the previous game, but it answers the challenge query $(V, \mathsf{upk}_0, \mathsf{upk}_1, \mathsf{mpk})$ as follows. First, it selects a subset $\mathcal{S}$ containing V elements that have never been involved in previous spendings of the ℓ^*-th coin. We recall that Game 2 ensures that such a subset exists, otherwise $\mathcal{R}$ would abort. Moreover, this game also ensures that this coin belongs to upk_0 or upk_1. Let us denote this user by upk^*. $\mathcal{R}$ then selects three random values $(k_1, k_2, k_3) \overset{\$}{\leftarrow} \mathcal{K}_{\mathcal{S}}^2 \times \mathbb{G}_1$ and returns $(k_{\mathcal{S}}^{(1)}, k_{\mathcal{S}}^{(2)}) = (k_1, k_2 \cdot G_1(\mathsf{upk}))$ along with $\mathtt{T}_{\mathcal{S}} = G_3(G_2(\mathsf{upk}), H(\mathsf{mpk}||\mathsf{info})) \cdot k_3$. Here again, the NIZK proofs can still be produced by using the $\mathtt{SimProve}$ algorithm. We note that the simulation is perfect unless $\mathcal{A}$ is able to distinguish the random tuples (k_1, k_2, k_3) from $(F_1.\mathtt{Const}(s, \mathcal{S}), F_2.\mathtt{Const}(s, \mathcal{S}), H(s))$ if $V = N$ or from $(F_1.\mathtt{Const}(s, \mathcal{S}), F_2.\mathtt{Const}(s, \mathcal{S}), H'(F_1.\mathtt{Const}(s, \mathcal{S}_{D(\mathcal{S})})))$ otherwise, which would yield an adversary against the strong combined key-pseudorandomness of F_1 and F_2 for the family (H, H'). We therefore have $\mathtt{Adv}_5 \geq \mathtt{Adv}_4 - \mathtt{Adv}_{\mathcal{A}}^{ckps}$.

We can note that in Game 5, the values received by the adversary are simulated proofs along with random elements $(k_{\mathcal{S}}^{(1)}, k_{\mathcal{S}}^{(2)})$ and $\mathtt{T}_{\mathcal{S}}$. The advantage $\mathtt{Adv}_5$ of $\mathcal{A}$ in this game is then 0. This means that:

$$\frac{\epsilon}{q_w} \leq \mathtt{Adv}_{\mathcal{A}}^{\mathtt{zk}} + \mathtt{Adv}_{\mathcal{A}}^{\mathtt{hid}} + \mathtt{Adv}_{\mathcal{A}}^{ckps}$$

which concludes the proof since q_w is polynomial in the security parameter λ.

E Instantiations

In this section we describe some constrained PRFs that one can use to instantiate our generic frameworks above. All of them are derived from published e-cash schemes, we therefore do not claim novelty here but simply show that these PRFs achieve the properties required by our constructions.

E.1 Homomorphic Constrained PRF

The divisible e-cash schemes described in [CPST15a, CPST15b, PST17] yield efficient constrained PRFs in a bilinear setting that nicely interact with Groth-Sahai non-interactive proofs [GS08]. For sake of simplicity, we only consider the PRF implicitly defined in [CPST15a] but we stress that similar arguments apply to the ones defined in [CPST15b, PST17].

Before providing a formal definition of the PRF, we first recall the notion of bilinear groups and introduce some useful notations.

Definition 21. *Bilinear groups are a set of three cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p along with a map, called pairing, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ that is*

1. *bilinear: for any $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{ab}$;*
2. *non-degenerate: for any $g \in \mathbb{G}_1^*$ and $\widetilde{g} \in \mathbb{G}_2^*$, $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$;*
3. *efficient: for any $g \in \mathbb{G}_1$ and $\widetilde{g} \in \mathbb{G}_2$, $e(g, \widetilde{g})$ can be efficiently computed.*

There are different types of pairings [GPS08] but we will here only make use of *type 3* pairings that assume that no efficiently computable homomorphism exists between $\mathbb{G}_1$ and $\mathbb{G}_2$. In the following every element of $\mathbb{G}_2$ will be denoted with a $\widetilde{}$ (*e.g.* $\widetilde{g}$) to distinguish them from elements of $\mathbb{G}_1$.

Construction. Let $\mathcal{S} = [1, 2^m]$ for some integer $m \geq 0$. For $i \in [0, m]$ and $j \in [0, 2^i[$, we define $\mathcal{S}_{i,j} = [1 + j \cdot 2^{m-i}, (j+1)2^{m-i}]$. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be bilinear groups as defined above and let g (resp. $\widetilde{g}$) be a generator of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$). We define below a PRF: $\mathcal{K} \times \mathcal{S} \rightarrow \mathcal{Y}$, where $\mathcal{K} = \mathbb{Z}_p$, $\mathcal{K}_{\mathcal{S}_{i,j}} = \mathbb{G}_1$, $\forall i, j$ and $\mathcal{Y} = \mathbb{G}_T$.

– $\texttt{Setup}(1^\lambda, \{\mathcal{S}_{i,j}\}_{i,j}$: This algorithm first selects random scalars $r_{i,j}$ for $i \in [0, m]$ and $j \in [0, 2^i[$ along with ℓ_t for $t \in [1, 2^m]$. It then includes the following elements in the public parameters pp:
  - $g_{i,j} = g^{r_{i,j}}$ for $i \in [0, m]$ and $j \in [0, 2^i[$
  - $\widetilde{g}_{i,j,t} = \widetilde{g}^{\ell_t / r_{i,j}}$ for $i \in [0, m]$, $j \in [0, 2^i[$ and $t \in \mathcal{S}_{i,j}$.

– $\texttt{Keygen}()$: this algorithm returns a random $s \xleftarrow{\$} \mathbb{Z}_p$.

– $\texttt{Const}(s, \mathcal{S}_{i,j})$: On input $s \in \mathbb{Z}_p$ and a subset $\mathcal{S}_{i,j}$, this algorithms returns a constrained key $k_{\mathcal{S}_{i,j}} = g_{i,j}^s$.

– $\texttt{Eval}(s, t)$: On input the master key s and an element $t \in \mathcal{S}$, this algorithm returns $e(g_{0,0}^s, \widetilde{g}_{0,0,t})$.

– $\texttt{ConstEval}(\mathcal{S}_{i,j}, k_{\mathcal{S}_{i,j}}, t)$: On input a set $\mathcal{S}_{i,j}$, a constrained key $k_{\mathcal{S}_{i,j}}$ and an element $t \in \mathcal{S}$, this algorithm returns $e(k_{\mathcal{S}_{i,j}}, \widetilde{g}_{i,j,t})$.

Correctness. For all $i \in [0, m]$, $j \in [0, 2^i[$ and $t \in \mathcal{S}_{i,j}$, we have:

$$\begin{aligned}
\texttt{ConstEval}(\mathcal{S}_{i,j}, \texttt{Const}(s, \mathcal{S}_{i,j}), t) &= \texttt{ConstEval}(\mathcal{S}_{i,j}, g_{i,j}^s, t) \\
&= e(g_{i,j}^s, \widetilde{g}_{i,j,t}) \\
&= e(g, \widetilde{g})^{\ell_t \cdot s} \\
&= e(g_{0,0}^s, \widetilde{g}_{0,0,t}) \\
&= \texttt{Eval}(s, t)
\end{aligned}$$

which ensures correctness. The bilinearity of e directly implies the key homomorphism of this PRF.

Collision Resistance.

Theorem 22. *Our PRF achieves collision resistance 1 under the symmetric discrete logarithm assumption [BCN+10].*

Proof. We recall that the symmetric discrete logarithm problem is the bilinear version of the discrete logarithm problem that assumes it is hard to recover x from the pair $(g^x, \widetilde{g}^x)$. Given such an instance of this assumption, our reduction $\mathcal{R}$ selects a random $t^* \in [1, 2^m]$ and generates the parameters as usual except that it implicitly sets $\ell_{t^*} = x$. We note that it always possible given both g^x and $\widetilde{g}^x$.

The parameters are correctly distributed, therefore at the end of the game the adversary $\mathcal{A}$ returns (s_1, s_2, t_1, t_2) such that:

$$\texttt{Eval}(s_1, t_1) = \texttt{Eval}(s_2, t_2) \wedge (s_1, t_1) \neq (s_2, t_2)$$

If $t^* \notin \{t_1, t_2\}$, then $\mathcal{R}$ aborts. Else let us assume without loss of generality that $t^* = t_1$.

$$\begin{aligned} &\mathtt{Eval}(s_1, t_1) = \mathtt{Eval}(s_2, t_2) \\ \Leftrightarrow\ & e(g, \widetilde{g})^{s_1 \cdot x} = e(g, \widetilde{g})^{s_2 \cdot \ell_{t_2}} \end{aligned}$$

$\mathcal{R}$ can thus return $x = \frac{s_2 \cdot \ell_{t_2}}{s_1}$ unless it aborts, which does not occur with probability at least $\frac{2}{2^m}$. □

Let us now consider collision resistance 2. For all $i \in [0, m]$, $j \in [0, 2^i[$, $t \in \mathcal{S}_i$ and $k_1, k_2 \in \mathbb{G}_1$, we have

$$\begin{aligned} \mathtt{ConstEval}_{\mathcal{S}_{i,j}}(k_1, t) &= \mathtt{ConstEval}_{\mathcal{S}_{i,j}}(k_2, t) \\ e(k_1, \widetilde{g})^{\ell_t / r_{i,j}} &= e(k_2, \widetilde{g})^{\ell_t / r_{i,j}} \end{aligned}$$

which implies that $k_1 = k_2$. No adversary can then succeed against collision resistance 2 of our PRF.

Similarly, if $\mathtt{ConstEval}_{\mathcal{S}_{i,j}}(k, t) = \mathtt{ConstEval}_{\mathcal{S}_{i',j'}}(k, t)$, then

$$e(k, \widetilde{g})^{\ell_t / r_{i,j}} = e(k, \widetilde{g})^{\ell_t / r_{i',j'}}.$$

If $k \neq 1_{\mathbb{G}_1}$, then $r_{i,j} = r_{i',j'}$, which is very unlikely since these scalars have randomly chosen. No adversary can then succeed against collision resistance 3 of our PRF.

Key-Pseudorandomness. Our proof is derived from the anonymity proof provided in the full version of [CPST15a]. We only prove key-pseudorandomness since it actually implies pseudorandomness.

We first recall the weak-EXDH assumption from [Duc10, CPST15a].

Definition 23. *Given $(g, g^x, g^a, g^{a \cdot y}, g^z)$ and $(\widetilde{g}, \widetilde{g}^a, \widetilde{g}^y)$, the weak-*EXDH *assumption states that it is hard to distinguish whether $z = a \cdot x \cdot y$ or z is random.*

Theorem 24. *Our PRF is key-pseudorandom under the weak-*EXDH *assumption.*

Proof. The reduction $\mathcal{R}$ makes a guess on the set $\mathcal{S}_{i^*,j^*}$ selected by the adversary of the pseudorandomness and will abort if it is incorrect. We note that this does not occur with probability at least $\frac{1}{n}$, where $n = 2^{m+1} - 1$ in our context. Now it generates as usual random scalars $r_{i,j}$ and ℓ_t but sets the public parameters as follows.

- $g_{i,j} = (g^{y \cdot a})^{r_{i,j}}$ if $\mathcal{S}_{i,j} \subset \mathcal{S}_{i^*,j^*}$
- $g_{i,j} = (g^a)^{r_{i,j}}$ if $\mathcal{S}_{i,j} \supsetneq \mathcal{S}_{i^*,j^*}$
- $g_{i,j} = g^{r_{i,j}}$ otherwise

- $\widetilde{g}_{i,j,t} = \widetilde{g}^{\ell_t/r_{i,j}}$ if $\mathcal{S}_{i,j} \subset \mathcal{S}_{i^*,j^*}$
- $\widetilde{g}_{i,j,t} = (\widetilde{g}^{y})^{\ell_t/r_{i,j}}$ if $\mathcal{S}_{i,j} \supsetneq \mathcal{S}_{i^*,j^*}$ and $t \in \mathcal{S}_{i^*,j^*}$
- $\widetilde{g}_{i,j,t} = \widetilde{g}^{\ell_t/r_{i,j}}$ if $\mathcal{S}_{i,j} \supsetneq \mathcal{S}_{i^*,j^*}$ and $t \notin \mathcal{S}_{i^*,j^*}$
- $\widetilde{g}_{i,j,t} = (\widetilde{g}^{a})^{\ell_t/r_{i,j}}$ otherwise

We note that this defines $\ell_t = a \cdot y \cdot \ell_t$ if $t \notin \mathcal{S}_{i^*,j^*}$ and $\ell_t = a \cdot \ell_t$ otherwise. Next, the reduction implicitly defines the secret master key as x. We note that, if the guess of $\mathcal{R}$ is correct, $\mathcal{R}$ will always received $\mathcal{O}$`Const` query on subset $\mathcal{S}_{i,j}$ such that $\mathcal{S}_{i,j} \cap \mathcal{S}_{i^*,j^*} = \emptyset$. Therefore the constrained key is $g^{x \cdot r_{i,j}}$, which can always be computed by $\mathcal{R}$ since it knows $r_{i,j}$.

To answer the challenge query from $\mathcal{A}$ on $\mathcal{S}_{i^*,j^*}$, $\mathcal{R}$ simply returns $Z = (g^z)^{r_{i^*,j^*}}$. If $z = a \cdot x \cdot y$, then Z is a valid constrained key and $\mathcal{R}$ plays the key-pseudorandomness game for $b = 0$. Otherwise, Z is random and $\mathcal{R}$ plays the key-pseudorandomness game for $b = 1$. Any adversary succeeding against key-pseudorandomness is then able to break the weak-EXDH assumption. $\square$

It now only remains to explain how one can generate a family of PRFs that achieves combined key-pseudorandomness. Actually, the technique is implicitly used in [CPST15a, CPST15b, PST17]. Given the previous PRF, one can generate a family of PRFs $(F_1, \ldots, F_t)$ of arbitrary size by using the same random values in the `Setup` algorithm but with a different generator g_i for $i \in [1, t]$. Therefore, for any subset $\mathcal{S}_{i,j}$, the family of constrained key is $(k^{(1)}_{\mathcal{S}_{i,j}}, \ldots, k^{(t)}_{\mathcal{S}_{i,j}}) = (g_1^{r_{i,j} \cdot s}, \ldots, g_t^{r_{i,j} \cdot s})$. It was shown in [PST17] that this tuple is indistinguishable from a random element of $\mathbb{G}_1^t$ under the key-pseudorandomness of the underlying PRF and under the DDH assumption in $\mathbb{G}_1$.

In our context, since the weak-EXDH assumption implies the DDH one, we have combined key-pseudorandomness for free.

Remark 25. As we mention at the beginning of this section, we can extract from [CPST15b, PST17] alternative PRFs with better efficiency but that rely on stronger assumptions. However, we note that the one from [PST17] supports any subinterval of $[1, N]$, contrarily to the one we have just described.

Remark 26. In this bilinear setting, the user's key pair $(\mathsf{usk}, \mathsf{upk})$ is usually $(x, g^x) \in \mathbb{Z}_p \times \mathbb{G}_1$ for some secret scalar x. Therefore, one can define $G_1 : g^a \in \mathbb{G}_1 \mapsto a \in \mathbb{Z}_p$ and $G_2 : X \in \mathbb{G}_1 \mapsto X$. We stress that we do not need G_1 to be efficiently computable since it will only be used by the users on their own public key upk for which they know the discrete logarithm usk. Such functions G_1 and G_2 are injective and thus trivially achieve collision resistance. Finally we define $G_3 : (X, a) \in \mathbb{G}_1 \times \mathbb{Z}_p \mapsto X^a \in \mathbb{G}_1$ which is bilinear and non-degenerate.

E.2 Delegatable Constrained PRF

We note that the GGM construction [GGM84] instantiated with hash functions yields simple delegatable constrained PRFs that have already been used in e-cash systems [ASM08]. Unfortunately, they do not interact nicely with NIZK proofs leading either to impractical constructions or to systems [ASM08] where correct evaluation of the PRF can only be proven through cut-and-choose techniques that seriously impact the overall security.

We therefore choose to describe the construction from [CG07] which, despite being very complex, does not need general zero-knowledge arguments.

Construction. Let $\mathcal{S} = [1, 2^m]$ for some integer $m \geq 0$. For $i \in [0, m]$ and $j \in [0, 2^i[$, we define $\mathcal{S}_{i,j} = [1 + j \cdot 2^{m-i}, (j+1)2^{m-i}]$ and a sequence of prime p_i such that p_i divides $p_{i+1} - 1$. We define below a PRF: $\mathcal{K} \times \mathcal{S} \to \mathcal{Y}$, where $\mathcal{K} = \mathbb{Z}_{p_0 - 1}$, $\mathcal{K}_{\mathcal{S}_{i,j}} = \mathbb{Z}_{p_i}$, $\forall i, j$ and $\mathcal{Y} = \mathbb{Z}_{p_m}$.

– $\texttt{Setup}(1^\lambda, \{\mathcal{S}_{i,j}\}_{i,j}$: This algorithm selects a generator g_0 of $\mathbb{Z}_{p_0}^*$ and, for $i \in [1, m]$, two elements $g_{i,0}$ and $g_{i,1}$ of $\mathbb{Z}_{p_i}$ of order p_{i-1}. Note that these elements necessarily exist because p_i divides $p_{i+1} - 1$. All these elements are included in the public parameters.
– $\texttt{Keygen}()$: this algorithm returns a random $s \xleftarrow{\$} \mathcal{K}$.
– $\texttt{Const}(s, \mathcal{S}_{i,j})$: On input $s \in \mathcal{K}$ and a subset $\mathcal{S}_{i,j}$, this algorithms first selects $b_0, \ldots, b_{i-1} \in \{0, 1\}$ such that $j = \sum_{\ell=0}^{i-1} b_\ell \cdot 2^{i-1-\ell}$. Let $k = g_0^s$. For $\ell = 0, \ldots i-1$ it computes $k \leftarrow g_{\ell+1, b_\ell}^k$ and finally returns the constrained key $k_{\mathcal{S}_{i,j}} = k$.
– $\texttt{Eval}(s, t)$: On input the master key s and an element $t \in [1, 2^m]$, this algorithm returns $\texttt{Const}(s, \mathcal{S}_{m,t-1})$.
– $\texttt{ConstEval}(\mathcal{S}_{i,j}, k_{\mathcal{S}_{i,j}}, t)$: On input a set $\mathcal{S}_{i,j}$, a constrained key $k_{\mathcal{S}_{i,j}}$ and an element $t \in \mathcal{S}$, this algorithm first selects $b_0, \ldots, b_{m-1}$ such that $t - 1 = \sum_{\ell=0}^{m-1} b_\ell \cdot 2^{m-1-\ell}$. It then sets $k \leftarrow k_{\mathcal{S}_{i,j}}$ and computes, for $\ell = i, \ldots, m-1$, $k \leftarrow g_{\ell+1, b_\ell}^k$. Finally, it returns the last value of k.
– $\overline{\texttt{Const}}(k_{\mathcal{S}_{i,j}}, \mathcal{S}_{i',j'})$: on input a constrained key $k_{\mathcal{S}_{i,j}}$ and a set $\mathcal{S}_{i',j'}$, this algorithm returns $\perp$ if $\mathcal{S}_{i',j'} \not\subseteq \mathcal{S}_{i,j}$. Else, it selects $b_0, \ldots, b_{i'-1} \in \{0, 1\}$ such that $j' = \sum_{\ell=0}^{i'-1} b_\ell \cdot 2^{i'-1-\ell}$ and sets $k = k_{\mathcal{S}_{i,j}}$. It then computes, for $\ell = i, \ldots i'-1$, $k \leftarrow g_{\ell+1, b_\ell}^k$ and outputs k.

Correctness. Let $b_0, \ldots, b_{m-1} \in \{0, 1\}$ be such that $t - 1 = \sum_{\ell=0}^{m-1} b_\ell \cdot 2^{m-1-\ell}$. The value $\texttt{Eval}(s, t) = \texttt{Const}(s, \mathcal{S}_{m,t-1})$ is defined by induction as follows. Let $k = g_0^s$. For $\ell = 0, \ldots, m-1$, one computes $k \leftarrow g_{\ell+1, b_\ell}^k$ and finally sets $\texttt{Eval}(s, t) = k$.

Now, for any $i \in [0, m]$ and $j \in [0, 2^i[$, let $b_0^{(j)}, \ldots, b_{i-1}^{(j)} \in \{0, 1\}$ be such that $j = \sum_{\ell=0}^{i-1} b_\ell^{(j)} \cdot 2^{i-1-\ell}$. One can note that for any $t \in [1, 2^m]$, $t \in \mathcal{S}_{i,j} \Leftrightarrow b_\ell = b_\ell^{(j)} \quad \forall \ell \in [0, i-1]$. Indeed:

$$\begin{aligned} & t \in \mathcal{S}_{i,j} \\ \Leftrightarrow\ & j \cdot 2^{m-i} \leq t-1 \leq (j+1)2^{m-i} - 1 \\ \Leftrightarrow\ & 2^{m-i}\sum_{\ell=0}^{i-1} b_\ell^{(j)} \cdot 2^{i-1-\ell} \leq t-1 \leq 2^{m-i}(1+\sum_{\ell=0}^{i-1} b_\ell^{(j)} \cdot 2^{i-1-\ell}) - 1 \\ \Leftrightarrow\ & \sum_{\ell=0}^{i-1} b_\ell^{(j)} \cdot 2^{m-1-\ell} \leq t-1 \leq \sum_{\ell=0}^{i-1} b_\ell^{(j)} \cdot 2^{m-1-\ell} + 2^{m-i} - 1 \\ \Leftrightarrow\ & b_\ell = b_\ell^{(j)} \quad \forall \ell \in [0, i-1]. \end{aligned}$$

Therefore, the `Const` algorithm evaluated on $(s, \mathcal{S}_{i,j})$ computes the i first steps of the `Eval` algorithm that are common to all elements $t \in \mathcal{S}_{i,j}$ whereas $\texttt{ConstEval}(\mathcal{S}_{i,j}, k_{\mathcal{S}_{i,j}}, t)$ computes the missing steps. This means that

$$\texttt{ConstEval}_{\mathcal{S}_{i,j}}(\texttt{Const}(s, \mathcal{S}_{i,j}), t) = \texttt{Eval}(s,t), \forall \mathcal{S}_{i,j} \ni t.$$

It then only remain to prove that $\overline{\texttt{Const}}(k_{\mathcal{S}_{i,j}}, \mathcal{S}_{i',j'}) = \texttt{Const}(s, \mathcal{S}_{i',j'}), \forall \mathcal{S}_{i,j} \supset \mathcal{S}_{i',j'}$. The condition $\mathcal{S}_{i,j} \supset \mathcal{S}_{i',j'}$ means that:

$$1 + j \cdot 2^{m-i} \leq 1 + j' \cdot 2^{m-i'} \leq (j'+1)2^{m-i'} \leq (j+1)2^{m-i}$$

The first inequality implies that $j' \geq \sum_{\ell=0}^{i-1} b_\ell^{(j)} 2^{i'-1-\ell}$ whereas the last one implies that $j' \leq \sum_{\ell=0}^{i-1} b_\ell^{(j)} 2^{i'-1-\ell} + 2^{i'-i} - 1$. Due to the unicity of the binary decomposition of j', this is only possible if $b_\ell^{(j)} = b_\ell^{(j')}$ for all $\ell \in [0, i-1]$. Here again, this means that the $\overline{\texttt{Const}}$ algorithm evaluated on $(k_{\mathcal{S}_{i,j}}, \mathcal{S}_{i',j'})$ computes the missing steps of the induction to get $k_{\mathcal{S}_{i',j'}}$, which concludes the proof of correctness.

Collision Resistance. We recall that our framework based on delegatable CPRFs only requires collision resistance-1. So let us assume that an adversary manages to output two distinct pairs (s,t) and (s',t') such that $\texttt{Eval}(s,t) = \texttt{Eval}(s',t')$. First, we note that we cannot have $t = t'$. Indeed, one can easily prove by induction that this would mean $g_0^s = g_0^{s'}$ and so that $s = s'$.

Let us now write

$$\begin{aligned} t - 1 &= \sum_{\ell=0}^{m-1} b_\ell^{(t)} \cdot 2^{m-1-\ell} \\ t' - 1 &= \sum_{\ell=0}^{m-1} b_\ell^{(t')} \cdot 2^{m-1-\ell} \end{aligned}$$

the inequality $t \neq t'$ implies that there is an index $\ell^* \in [0, m-1]$ such that (1) $b_{\ell^*}^{(t)} \neq b_{\ell^*}^{(t')}$ and (2) $b_\ell^{(t)} = b_\ell^{(t')}$ $\forall \ell \in [\ell^*+1, m-1]$. The latter condition implies that the last steps of the computation of $\texttt{Eval}(s,t)$ and $\texttt{Eval}(s',t')$ are the same. More specifically, there are k_{m-1} and k'_{m-1}, such that $\texttt{Eval}(s,t) = g_{m,b^{(t)}_{m-1}}^{k_{m-1}} = g_{m,b^{(t)}_{m-1}}^{k'_{m-1}} = \texttt{Eval}(s',t')$, which means that $k_{m-1} = k'_{m-1}$ and so on until we get $g_{\ell^*+1,b^{(t)}_{\ell^*}}^{k_{\ell^*}} = g_{\ell^*+1,b^{(t')}_{\ell^*}}^{k'_{\ell^*}}$. The latter equation means that the discrete logarithm of $g_{\ell^*+1,b^{(t')}_{\ell^*}}$ in base $g_{\ell^*+1,b^{(t)}_{\ell^*}}$ is k_{ℓ^*}/k'_{ℓ^*}, which can easily be recovered from s and s'. Therefore, any adversary succeeding against the collision resistance of this PRF can be converted into an adversary against the discrete logarithm in one of the groups $\mathbb{Z}_{p_i}$.

Key-Pseudorandomness. The divisible e-cash system of [CG07] is claimed to be anonymous under a so-called Matching Multi Diffie-Hellman (MMDH) assumption introduced by the authors. Unfortunately, the latter do not provide any detail in the proof and it seems very difficult to introduce an MMDH instance in the security reduction while being able to answer to any adversarial query. We therefore introduce a new assumption which is quite similar to the MMDH assumption, but which allows us to prove the key-pseudorandomness of this PRF.

Definition 27. *Let $u > 0$ be an integer and, for $i \in [0, u-1]$, q_i be a prime number such that q_i divides $q_{i+1} - 1$. Let $g_{0,\alpha}$ and $g_{0,\beta}$ be generators of $\mathbb{Z}^*_{p_0}$ and, for $i \in [1, u]$, let $g_{i,\alpha}$ and $g_{i,\beta}$ be two elements of $\mathbb{Z}^*_{q_i}$ of order q_{i-1}. For any integer x and $i \in [0, u]$, we define by induction $P_i = g_{i,\alpha}^{P_{i-1}}$ with $P_0 = g_{0,\alpha}^x$.*

The u-MMDH assumption states that it is hard to distinguish $Z = P_u(x)$ from a random element of $\mathbb{Z}^*_{q_u}$ of order q_{u-1}, even given access to $\{(g_{i,\alpha}, g_{i,\beta})\}_{i=0}^u$ and $(g_{0,\beta}^x, g_{1,\beta}^{P_0}, \dots g_{u,\beta}^{P_{u-1}})$

Theorem 28. *The PRF described above is key-pseudorandom under the m-MMDH assumption.*

Proof. Given a m-MMDH instance, the reduction $\mathcal{R}$ makes a guess on the set $\mathcal{S}_{i^*,j^*}$ that the adversary will output. For $\ell \in [0, i^*]$, it sets $p_\ell = q_{m-i^*+\ell}$ and selects the other primes p_ℓ as usual for $\ell \in [i^*+1, m]$.

Let $b^*_0, \dots, b^*_{i^*-1}$ be such that $j = \sum_{\ell=0}^{i^*-1} b^*_\ell \cdot 2^{i^*-1-\ell}$. The reduction then sets $g_0 = g_{m-i^*,\alpha}$ and, for $\ell \in [1, i^*]$, $g_{\ell,b_{\ell-1}} = g_{m-i^*+\ell,\alpha}$ and $g_{\ell,\bar{b}_{\ell-1}} = g_{m-i^*+\ell,\beta}$, where $\bar{b}$ denotes $1-b$. All the other elements of the public parameters are randomly generated.

In the reduction, $\mathcal{R}$ implicitly defines the seed as P_{m-i^*-1}. Since the $\texttt{Eval}$ function returns the output of $\texttt{Const}$, we just explain how $\mathcal{R}$ can handle the $\mathcal{O}\texttt{Const}$ queries. Let $\mathcal{S}_{i,j}$ be the subset queried to the $\mathcal{O}\texttt{Const}$ oracle and $j = \sum_{\ell=0}^{i-1} b_\ell \cdot 2^{i-1-\ell}$. If the guess on $\mathcal{S}_{i^*,j^*}$ is correct, then there exists $\ell^* \in [0, min(i, i^*) - 1]$ such that $b^*_{\ell^*} \neq b_{\ell^*}$. Otherwise, we would have $\mathcal{S}_{i,j} \subset \mathcal{S}_{i^*,j^*}$ or $\mathcal{S}_{i^*,j^*} \subset \mathcal{S}_{i,j}$ and the guess would be invalid. Therefore, at the ℓ^*-th step of the computation of $k_{\mathcal{S}_{i,j}}$, the value k is of the form $g_{\ell^*+1,b_{\ell^*}}^{k'} = g_{m-i^*+\ell^*+1,\beta}^{k'}$ with k' depending on P_{m-i^*-1} and $\{(g_{i,\alpha}, g_{i,\beta})\}_{m-i^*}^{m}$. The element $g_{m-i^*+\ell^*+1,\beta}^{k'}$ can then easily be derived from the elements provided in the m-MMDH instance which ensures that the simulation is perfect.

When the adversary outputs the challenge subset, $\mathcal{R}$ aborts if its guess is incorrect. Otherwise, it returns Z from the m-MMDH instance. If $Z = P_u(x)$, then it is a valid constrained key and the adversary $\mathcal{A}$ plays the key-pseudorandomness game for $b = 0$. Else, it is a random element from the corresponding constrained key space, which corresponds to the case where $b = 1$.

Strong Key-Pseudorandomness. We note that, for any subset $\mathcal{S}_{i,j} \neq [1, 2^m]$, the constrained key $k_{\mathcal{S}_{i,j}}$ is of the form $g_{i,b}^{k_{\mathcal{S}_{i-1,j'}}}$ where $\mathcal{S}_{i-1,j'}$ is the smallest subset strictly containing $\mathcal{S}_{i,j}$. Now let h_0 be a generator of $\mathbb{Z}_{p_0}$ and h_i be an element of $\mathbb{Z}_{p_i}$ of order p_{i-1}. We can then define $H : s \mapsto h^s$ and $H_i : k \mapsto h_i^k$.

Therefore, in the strong key-pseudorandomness game, the adversary will now have to distinguish pairs of the form $(g_{i,b}^{k_{\mathcal{S}_{i-1,j'}}}, h_i^{k_{\mathcal{S}_{i-1,j'}}})$ from random elements. By using the same argument as in the key-homomorphic case, we can prove that this is not possible under the key-pseudorandomness of the PRF and the DDH assumption.