Divisible E-Cash from Constrained Pseudo-Random Functions

Florian Bourse[1], David Pointcheval[2,3] and Olivier Sanders[1]

[1] Orange Labs, Applied Crypto Group, Cesson-Sévigné, France
[2] DIENS, Ecole normale supérieure, CNRS, PSL University, Paris, France
[3] INRIA, Paris, 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, quite a few constructions exist, all of them relying on complex mechanisms that can only be instantiated in one specific setting. In addition security models are incomplete and proofs sometimes hand-wavy.
In this work, we first provide a complete security model for divisible e-cash, and we study the links with constrained pseudo-random functions (PRFs), a primitive recently formalized by Boneh and Waters. We exhibit two frameworks of divisible e-cash systems from constrained PRFs achieving some specific properties: either key homomorphism or delegability. We then formally prove these frameworks, and address two main issues in previous constructions: two essential security notions were either not considered at all or not fully proven. Indeed, we introduce the notion of *clearing*, which should guarantee that only the recipient of a transaction should be able to do the deposit, and we show the *exculpability*, that should prevent an honest user to be falsely accused, was wrong in most proofs of the previous constructions. Some can easily be repaired, but this is not the case for most complex settings such as constructions in the standard model. Consequently, we provide the first construction secure in the standard model, as a direct instantiation of our framework.

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 [22] 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 completely 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 (a unique coin for each denomination) these requirements impose very strong constraints on e-cash systems: users should remain anonymous as long as they behave honestly while becoming traceable as soon as they begin overspending, from the first cent.

Chaum's idea, taken up by all subsequent works, was to associate each withdrawn coin with a unique identifier called a "serial number"[4]. 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 Work

On-line/Off-line Anonymous e-Cash. The original solution proposed by Chaum for anonymous payment was based on the concept of blind signature. This primitive, later formalized in [37,38], allows anyone to get a signature σ on

[4] Actually, this specific terminology appeared later [23] but this notion is implicit in the Chaum's paper.

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 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.

Transferable vs. Divisible e-Cash. In theory, the problem of anonymous payment is thus solved by blind signatures for which several instantiations have been proposed (see *e.g.* [38]). 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 (anonymously) 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 [24] 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 results were later extended by Canard and Gouget [17] 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 [3, 17].

All these negative results perhaps explain the small number of results on transferable e-cash, and quite recent constructions [3, 8, 19] are too complex for a

large-scale deployment or rely on a very unconventional model [26]. 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 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 called *compact* and a system supporting both efficient storage and spending is called *divisible*.

Anonymous Compact e-Cash. Anonymous compact e-cash was proposed by Camenisch, Hohenberger and Lysyanskaya [15] 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 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 properties of the proofs and from the properties of the pseudo-random function, as it is hard to decide whether $F_s(i)$ and $F_s(j)$ have been generated under the same secret key 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 actually provided by Okamoto and Ohta [34] and later named *divisible* e-cash. The core idea of divisible e-cash is that the serial numbers of a divisible coin[5] 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 [16]. Moreover, the latter was more a proof of concept than a practical scheme, as pointed out in [2, 18]. Although several improvements followed (*e.g.* [2, 18, 20, 36]), the resulting constructions are still rather complex, which makes their analysis difficult. We highlight this issue by pointing out below a problem on exculpability that has been overlooked in the security proofs of these constructions.

1.2 A Major Issue with Exculpability in Previous Constructions

Intuition of the Problem. Among the natural properties expected from an e-cash system is the one, called *exculpability*, stating that a coin withdrawn by

[5] 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.

a user whose public key is upk^* can only be spent by the latter. In particular this means that he cannot falsely be accused of double-spendings: in case of overspending detection, this user is necessarily guilty. All e-cash constructions enforce this property by requiring a signature (potentially a signature of knowledge) on the transaction under upk^*. Intuitively, this seems enough: 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[6] and leads to quite simple security proofs. It is explicitly stated in Section D.3 of the full version of [32] and in Section 4.6 of [18], and implicitly used in Section 6.3 of [20], in Section 6.2 of [36], and in the security proofs (page 22) of the full version of [15].

Unfortunately, this argument is not correct because of the complex identification process of e-cash systems, based on so-called double spending tags. Indeed, the public key upk^* returned by the identification 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 this algorithm to output upk^* while taking as input two transactions generated with different public keys. This scenario, that has not been taken into account in previous papers, invalidates their proofs[7] because, in such a case, the transactions do not contain a valid signature under upk^*.

Concrete Example. To illustrate this problem, let us consider the lattice-based construction proposed by Libert *et al* [32]. In this system, each user selects a short vector $\mathbf{e}$ and defines his 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 $\mathbf{R}$.

If two transactions $\mathbf{R}$ and $\mathbf{R'}$ 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^* and tag $\mathbf{t}$, as y^* is then $\lfloor \mathbf{A}_i \cdot \mathbf{t} \rfloor_p$. However, there is no equivalence here, and an adversary might manage to generate $\mathbf{R}, \mathbf{R'}, \mathbf{t}, \mathbf{t'}, \mathsf{upk}, \mathsf{upk}'$ (in the exculpability game the adversary controls the bank, the merchants and all dishonest users) such that $\mathsf{upk}^* = y_T - \mathbf{H(R)} \cdot (\mathbf{H(R)} - \mathbf{H(R')})^{-1}(y_T - y'_T)$.

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

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

[6] Our comment obviously only applies to papers that provide a security proof.

[7] 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.

Discussion. 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.* [15, 16, 18] and the ROM constructions of [20, 21, 36]).

Indeed, in such a case, the double-spending tag is of the form $\mathtt{T} = \mathsf{upk} \cdot F_s(i)^R$ 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 $\mathtt{T}$ and $\mathtt{T}'$ from which one can recover upk by computing $(\mathtt{T}^{R'}/(\mathtt{T}')^R)^{\frac{1}{R'-R}}$.

Here again, an adversary might generate $\mathsf{upk}, s, R, \mathsf{upk}', s', R'$ such that the corresponding tags $\mathtt{T}$ and $\mathtt{T}'$ satisfy $(\mathtt{T}^{R'}/(\mathtt{T}')^R)^{\frac{1}{R'-R}} = \mathsf{upk}^*$, 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 in the Random Oracle Model, and so cannot be applied to other settings (*e.g.* lattices [32]) or to standard model constructions [20, 21, 36]. In particular, this means that divisible e-cash secure in the standard model or even lattice-based compact e-cash is still an open problem.

1.3 Contributions

One can note that the above issue has remained undetected for more than a decade, whereas all compact/divisible e-cash systems are based on the same intuition. However, the latter has never been formalized. Intuition is necessary to design and understand a scheme but we must be very careful when it comes to complex primitives. This pleads for a more formal approach, where the common intuition are translated into a generic framework.

In addition, this lack of generic framework leads designers to create and combine several ad-hoc mechanisms, with complex security proofs that often rely on tailored computational assumptions. This stands in sharp contrast with a related primitive, group signature, whose foundations were studied by Bellare *et al* [5, 6] and for which very efficient constructions exist.

Two Generic Frameworks and Concrete Instantiations. In this work, we propose two generic frameworks that yield secure divisible e-cash systems from constrained PRFs, a well-known cryptographic primitive. For each framework, we identify the properties it must achieve and, so, we reduce the problem of

constructing divisible e-cash systems to a simpler one: efficient instantiations of the building blocks. We additionally provide examples of instantiations to show that our frameworks are not artificial but can lead to practical schemes.

Our Approach: Constrained Pseudo-Random Functions. Starting from the work of Camenisch *et al* [15] 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 [12, 14, 31].

There are also several requirements that must implicitly be fulfilled by the *constrained key* $k_{\mathcal{S}}$, for anonymity to hold, and namely unlinkability of the transactions: different constrained keys generated from the same master key must be unlinkable, which also requires $k_{\mathcal{S}}$ to hide any information on the subset $\mathcal{S}$ (besides its cardinality, which will represent the amount). All these notions were already defined in previous papers on constrained PRFs (*e.g.* [4, 9, 11]), although we only need here weaker versions of the original definitions.

Collision Resistance. Intuitively, unlinkability of $k_{\mathcal{S}}$ will ensure honest users' privacy. However, e-cash systems must also be able to deal with dishonest parties, including the bank itself. In such a case, the adversary has much more power than in usual PRF security games: it has 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 PRFs, 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.

Key Homomorphic vs. Delegable Constrained PRFs. We then investigate two different scenarios, leading to two different (but related) frameworks. In the former, we consider *key homomorphic* constrained PRF [4] whereas we use *delegatable* constrained PRF [31] in the latter. 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 get, from existing systems, constrained PRFs (either key 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 have been ignored. Second, it means that some of the constructions affected by the exculpability issue (see Section 1.2) could be fixed by using the same tricks we introduce in our frameworks.

Serial Numbers and Double Spending Tags. 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. This is definitely the main contribution of the 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.

First Divisible e-Cash System Secure in the Standard Model. Finally, we provide detailed proofs for both 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. To illustrate this point, we describe, by using our framework, the first divisible e-cash system secure in the standard model, since previous analyses in the standard model are all wrong, as explained above.

Several Security Issues. Another interesting outcome of our formalization process is that it highlights some security issues that have often been overlooked in previous papers.

First, there is the critical issue with exculpability, as discussed in Section 1.2.

Second, security models of e-cash systems only deal with the security of the users and the bank. We indeed note that (almost) no property related to the security of the merchant has ever been formalized. In particular, the ability of the merchants to deposit the electronic coins they received is not ensured by the e-cash scheme itself. For example, in most systems, nothing prevents the spender from depositing the coins he has just spent[8]: we define a new property, called *clearing*, that formalizes the security requirements for the merchants.

Eventually, in the withdrawal procedure, the coins secret values are traditionally generated collaboratively by the bank and the user. Our security analysis shows that this collaborative generation does not seem to provide any relevant benefit, at least for our frameworks.

1.4 Organization

We recall in Section 2.1 the notion of constrained pseudo-random functions and detail the security properties required in order to construct divisible e-cash systems in Section 2.2 (concrete instantiations of constrained PRFs can be found in the full version [13]). 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 the full

[8] Identification of the spender is not possible in this case because the two transcripts received by the bank (the one sent by the spender and the one sent by the merchant) are exactly the same.

version [13]. The security analysis of our generic constructions is provided in the full version. The latter also contains a concrete instantiation of our framework along with an additional security notion for delegatable constrained PRFs.

2 Constrained Pseudo-Random Function

Our constructions of divisible e-cash systems will heavily rely on constrained pseudo-random functions [12,14,31] 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.

- $\mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$: 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;
- $\mathtt{Keygen}()$: this algorithm outputs a master secret key $s \in \mathcal{K}$;
- $\mathtt{CKey}(s, \mathcal{X})$: On input the master key s and a set $\mathcal{X}$, this deterministic[9] algorithm outputs a constrained key $k_\mathcal{X} \in \mathcal{K}_\mathcal{X}$ or $\perp$;
- $\mathtt{Eval}(s, x)$: On input the master key s and an element $x \in \mathcal{S}$, this deterministic algorithm outputs a value $y \in \mathcal{Y}$;
- $\mathtt{CEval}(\mathcal{X}, k_\mathcal{X}, x)$: On input a set $\mathcal{X}$, a constrained key $k_\mathcal{X}$ and an element $x \in \mathcal{X}$, this deterministic algorithm outputs a value $y \in \mathcal{Y}$.

For conciseness, we will denote $\mathtt{CEval}(\mathcal{X}, k_\mathcal{X}, x)$ by $\mathtt{CEval}_\mathcal{X}(k_\mathcal{X}, x)$.

A constrained PRF is *correct* for a family of subsets $\{\mathcal{S}_i\}_{i=1}^n$ if, for all $\lambda \in \mathbb{N}$, $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$, $s \leftarrow \mathtt{Keygen}()$ and $x \in \mathcal{S}_i \subseteq \mathcal{S}$, we have, with overwhelming probability, $\mathtt{CEval}_{\mathcal{S}_i}(\mathtt{CKey}(s, \mathcal{S}_i), x) = \mathtt{Eval}(s, x)$. And this common value is $\mathrm{PRF}_s(x)$.

Definition 1. *A constrained PRF is key homomorphic [4, 10] if:*

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

We use the multiplicative notation for our group operations, in $\mathcal{K}$ and $\mathcal{K}_{\mathcal{S}_i}$. As in [4], we require that the $\mathtt{CKey}$ algorithm, for any $\mathcal{S}_i$, is a group homomorphism from $\mathcal{K}$ into $\mathcal{K}_{\mathcal{S}_i}$.

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.

[9] Although the general definition in [12] allows randomized $\mathtt{CKey}$ algorithm, all our constructions will require this algorithm to be deterministic.

Definition 2. *A constrained pseudo-random function is delegatable [31] if it additionally supports the following algorithm:*

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

To be *correct*, the delegatable constrained PRF must additionally satisfy, for a family of subsets $\{\mathcal{S}_i\}_{i=1}^n$, that, for all $\lambda \in \mathbb{N}$, $pp \leftarrow \texttt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$, $s \leftarrow \texttt{Keygen}()$, $\mathcal{S}_i \subset \mathcal{S}_j \subseteq \mathcal{S}$, and $k_{\mathcal{S}_j} \leftarrow \texttt{CKey}(s, \mathcal{S}_j)$, we have, with overwhelming probability, $\overline{\texttt{CKey}}(k_{\mathcal{S}_j}, \mathcal{S}_i) = \texttt{CKey}(s, \mathcal{S}_i)$.

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.

Pseudo-Randomness (PR). The first property one may expect from a constrained PRF is *pseudo-randomness*, which informally requires that an adversary, even given access to constrained keys, cannot distinguish the PRF evaluation from random, for a new point (not already queried and outside sets of known constrained keys). It is defined by $\texttt{Exp}_{\mathcal{A}}^{pr-b}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ in Figure 1 where the adversary has access to the following oracles:

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

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

Key Pseudo-Randomness (KPR). We note that the previous definition only requires pseudo-randomness for the output of the PRF. As in [4] we extend this property to the constrained keys themselves, leading to a property that we call key pseudo-randomness. However, compared to [4], we additionally require some form of key privacy, in the sense of [31]. In particular, we need that constrained keys issued for subsets of the same size[10] should be indistinguishable.

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

[10] We note that our privacy requirements are weaker than the ones of [9, 11] since we allow the constrained keys to leak the size of the subsets.

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

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

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

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

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

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. $i^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{CKey}, \mathcal{O}\mathtt{Eval}}(\{pp_j\}_{j=1}^t)$
4. $(k_0^1, \ldots, k_0^t) \leftarrow (F_1.\mathtt{CKey}(s, \mathcal{S}_{i^*}), \ldots, F_t.\mathtt{CKey}(s, \mathcal{S}_{i^*}))$
5. $(k_1^1, \ldots, k_1^t) \xleftarrow{\$} \mathcal{K}^t_{\mathcal{S}_{i^*}}$
6. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{CKey}, \mathcal{O}\mathtt{Eval}}(\{pp_j\}_{j=1}^t, (k_b^1, \ldots, k_b^t))$
7. If $\mathcal{O}\mathtt{Eval}$ was queried on $x \in \mathcal{S}_{i^*}$, return 0
8. If $\mathcal{O}\mathtt{CKey}$ was queried on $\mathcal{X}$ such that $\mathcal{X} \cap \mathcal{S}_{i^*}$, return 0
9. Return b^*

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

Combined Key Pseudo-Randomness (CKPR). In practice, divisible e-cash systems require multiple pseudo-random 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 pseudo-random, a solution could be to split the constrained key $k_{\mathcal{S}_i} \leftarrow \mathtt{CKey}(s, \mathcal{S}_i)$ into several parts, each of them being used as pseudo-random values. 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 pseudo-random values by using different PRFs $F_1, \ldots, F_t$ evaluated on the same master key s and the same subset $\mathcal{S}_i$: Let $F_1 \ldots, F_t$ be constrained PRFs $\mathcal{K} \times \mathcal{S} \to \mathcal{Y}$ defined for the same family of subsets $\{\mathcal{S}_i\}_{i=1}^n$ satisfying $\mathcal{K}_{\mathcal{S}_i} = \mathcal{K}_{\mathcal{S}_j}$ $\forall i, j$ such that $|\mathcal{S}_i| = |\mathcal{S}_j|$. We say that the family $(F_1, \ldots, F_t)$ achieves combined key pseudo-randomness if

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

This can be done very easily by constructing each F_i similarly but with different public parameters: let us assume that $F_1.\mathtt{CKey}(s, \mathcal{S}_i) = k^1_{\mathcal{S}_i} = g_1^{\alpha_i \cdot s}\ \forall i$ 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 $F_j.\mathtt{CKey}(s, \mathcal{S}_i) = k^j_{\mathcal{S}_i} = g_j^{\alpha_i \cdot s}$ 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 pseudo-randomness of F_1 and the DDH assumption (see the full version [13] for more details).

Collision Resistance (CR). 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 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 pseudo-randomness. 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 pseudo-randomness 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 $\mathcal{A}$, $\mathtt{Adv}^{cr-k}(\mathcal{A}) = \Pr[\mathtt{Exp}_{\mathcal{A}}^{cr-k}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n) = 1]$ is negligible. We provide in the full version [13] several examples of PRFs achieving these properties.

3 Divisible E-Cash

The syntax and the formal security model are drawn from [20,36]. We nevertheless introduce several changes to make them more generic but also to add some specifications that were previously implicit only.

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.

Collision Resistance 1
$\mathtt{Exp}^{cr-1}_{\mathcal{A}}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$

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. If $(s_1, x_1) = (s_2, x_2)$, return 0
4. Return $\mathtt{Eval}(s_1, x_1) = \mathtt{Eval}(s_2, x_2)$

Collision Resistance 2
$\mathtt{Exp}^{cr-2}_{\mathcal{A}}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$

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. If $k_1 = k_2$, return 0
4. Return $\mathtt{CEval}_{\mathcal{S}_i}(k_1, x) = \mathtt{CEval}_{\mathcal{S}_i}(k_2, x)$

Collision Resistance 3
$\mathtt{Exp}^{cr-3}_{\mathcal{A}}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$ (for Key Homomorphic Constrained PRFs only)

1. $pp \leftarrow \mathtt{Setup}(1^\lambda, \{\mathcal{S}_i\}_{i=1}^n)$
2. $(i, j, k_i, k_j, x) \leftarrow \mathcal{A}(pp)$
3. If $i = j$, return 0
4. If $k_i = 1_{\mathcal{K}_{\mathcal{S}_i}} \vee k_j = 1_{\mathcal{K}_{\mathcal{S}_j}}$, return 0
5. Return $\mathtt{CEval}_{\mathcal{S}_i}(k_i, x) = \mathtt{CEval}_{\mathcal{S}_j}(k_j, x)$

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

- $\mathtt{Setup}(1^\lambda, N)$: 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 λ and N. They are also given as an implicit input to the adversary, we will then omit them.
- $\mathtt{BKeygen}()$: 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.
- $\mathtt{Keygen}()$: 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).
- $\mathtt{Withdraw}(\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk}))$: 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 $\perp$.
- $\mathtt{Spend}(\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, V), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, \mathsf{info}, V))$: This is an interactive protocol between a user $\mathcal{U}$ and a merchant $\mathcal{M}$. Here, info denotes a set of public information associated to the transaction, by the merchant, 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 $\perp$.
- $\mathtt{Deposit}(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (V, \mathsf{info}, Z, \Pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: 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 $\mathtt{SN}_{i_0}, \ldots, \mathtt{SN}_{i_{V-1}}$[11] 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 $\mathtt{SN}_{i_k}$ is also a serial number derived from Z'. Else, $\mathcal{B}$ stores these new serial numbers in L and keeps a copy of $(V, \mathsf{info}, \mathsf{mpk}, Z, \Pi)$.

- $\mathtt{Identify}((V, \mathsf{info}, \mathsf{mpk}, Z, \Pi), (V', \mathsf{info}', \mathsf{mpk}', Z', \Pi'), \mathsf{bpk})$: On the wo 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 $\perp$.
- $\mathtt{CheckDeposit}([(V, \mathsf{info}, \mathsf{mpk}, Z, \Pi), \mu], \mathsf{bpk})$: This deterministic algorithm outputs 1 if $[(V, \mathsf{info}, Z, \Pi), \mu]$ are valid elements deposited by a merchant whose public key is mpk and 0 otherwise.

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 [20]: *anonymity* (user' spendings are anonymous, even with respect to the bank), *exculpability* (honest users cannot be falsely accused, even by the bank) and *traceability* (an author of overspending should be traced back).

However, all these notions (and the corresponding ones in previous papers) fail to capture an important security property for the merchant: he must always be able to clear his transactions, but also, 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[12] 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.

[11] We do not make any assumption on the indices $i_0, \ldots, i_{V-1}$, contrarily to some previous works that assume they are consecutive.

[12] The "correctness for merchant", informally defined in [2], 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 [2] does not ensure that the merchant is the only one able to clear his coins.

For completeness, we therefore add the property of *clearing* (only the recipient merchant can perform the deposit) to the above usual ones. All of them are defined in Figure 3 and make use of the following oracles:

- $\mathcal{O}\mathtt{Add}()$ 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*.
- $\mathcal{O}\mathtt{Corrupt}(\mathsf{upk}/\mathsf{mpk})$ 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*.
- $\mathcal{O}\mathtt{AddCorrupt}(\mathsf{upk}/\mathsf{mpk})$ 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 `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 `Withdraw` protocol. This oracle will be used by $\mathcal{A}$ playing the role of a user whose public key is upk against the bank.
- $\mathcal{O}\mathtt{Spend}(\mathsf{upk}, V)$ is an oracle that executes the user's side of the `Spend` protocol for a value V. This oracle will be used by $\mathcal{A}$ playing the role of the merchant $\mathcal{M}$.
- $\mathcal{O}\mathtt{Receive}(\mathsf{mpk}, V)$ is an oracle that executes the merchant's side of the `Spend` protocol for a value V. This oracle will be used by $\mathcal{A}$ playing the role of a user.
- $\mathcal{O}\mathtt{Deposit}(\mathsf{mpk}, V, \mathsf{info})$ is an oracle that executes the merchant's side of the `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 $\mathcal{O}\mathtt{Spend}$ queries, m_{upk} the number of divisible coins that he has withdrawn and Tr_i the transcript $(V_i, \mathsf{info}_i, \mathsf{mpk}_i, Z_i, \Pi_i)$ 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 $\mathcal{O}\mathtt{AddCorrupt}$ queries, in the traceability, exculpability and anonymity experiments. We therefore do not need to add access to the $\mathcal{O}\mathtt{Receive}$ and $\mathcal{O}\mathtt{Deposit}$ oracles in the latter. We stress that this is not a restriction since the $\mathcal{O}\mathtt{AddCorrupt}$ oracle provides more power to the adversary than the $\mathcal{O}\mathtt{Add}$

and $\mathcal{O}\mathtt{Corrupt}$ 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 $\mathcal{O}\mathtt{Spend}$, $\mathcal{O}\mathtt{Withdraw}_{\mathcal{U}}$ and $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}$ oracles in it.

$\mathtt{Exp}^{tra}_{\mathcal{A}}(1^\lambda, N)$ – 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

$\mathtt{Exp}^{excu}_{\mathcal{A}}(1^\lambda, N)$ – 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 upk not corrupted, then return 1
5. Return 0

$\mathtt{Exp}^{anon-b}_{\mathcal{A}}(1^\lambda, N)$ – 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 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 upk_i has been corrupted for $i \in \{0, 1\}$, then return 0
10. Return $(b = b^*)$

$\mathtt{Exp}^{clear}_{\mathcal{A}}(1^\lambda, N)$ – 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 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

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.

A divisible E-cash system is said to be *traceable*, *exculpable*, *anonymous*, and/or *clearable* if $\mathtt{Succ}^{tra}(\mathcal{A})$, $\mathtt{Succ}^{excu}(\mathcal{A})$, $\mathtt{Adv}^{anon}(\mathcal{A})$, and/or $\mathtt{Succ}^{clear}(\mathcal{A})$,

are respectively negligible for any probabilistic polynomial adversary $\mathcal{A}$, where

$$\mathtt{Succ}^{tra}(\mathcal{A}) = \Pr[\mathtt{Exp}^{tra}_{\mathcal{A}}(1^\lambda, N) = 1] \qquad \mathtt{Succ}^{excu}(\mathcal{A}) = \Pr[\mathtt{Exp}^{excu}_{\mathcal{A}}(1^\lambda, N) = 1]$$
$$\mathtt{Succ}^{clear}(\mathcal{A}) = \Pr[\mathtt{Exp}^{clear}_{\mathcal{A}}(1^\lambda, N) = 1]$$
$$\mathtt{Adv}^{anon}(\mathcal{A}) = |\Pr[\mathtt{Exp}^{anon-1}_{\mathcal{A}}(1^\lambda, N) = 1] - \Pr[\mathtt{Exp}^{anon-0}_{\mathcal{A}}(1^\lambda, N) = 1]|$$

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 $\mathtt{SN}$ must look random has led designers to use pseudo-random functions (PRFs). More specifically, every anonymous divisible e-cash scheme defines $\mathtt{SN}_i$ as $F.\mathtt{Eval}(s, i)$ 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.* [2,16,18,20,21,36]) have thus constructed pseudo-random functions with a special feature: given s and a subset $\mathcal{X} \subseteq [1, N]$, one can compute $k_{\mathcal{X}}$ allowing to evaluate the PRF only on the elements of $\mathcal{X}$. This matches the definition of constrained PRFs, as described above. To spend a value V, the user can now simply reveal a constrained key $k_{\mathcal{X}}$ for a set $\mathcal{X}$ 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 $\mathcal{X}$[13] itself. All these properties are captured by key pseudo-randomness that we defined in Section 2. Eventually, to avoid false positive in the fraud 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 pseudo-random, collision resistant constrained PRF for subsets of $[1, N]$ that smoothly interacts with Non-Interactive Zero-Knowledge (NIZK) proofs. 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 [15, 32]. In these papers, the double spending tag $\mathtt{T}_i$ associated with $\mathtt{SN}_i$ is of the form $\mathtt{ID} \cdot (F'.\mathtt{Eval}(s', i))^R$, where $\mathtt{ID}$ is the "identity" of the spender (usually his public key), F' is a PRF seeded with a

[13] Actually the size of $\mathcal{X}$ can leak as it corresponds to the public amount of the transaction.

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 $\mathtt{T}_i^{(1)}$ and $\mathtt{T}_i^{(2)}$ of the form $\mathtt{ID} \cdot (F'.\mathtt{Eval}(s', i))^{R_1}$ and $\mathtt{ID} \cdot (F'.\mathtt{Eval}(s', i))^{R_2}$. Therefore, by computing:

$$((\mathtt{T}_i^{(1)})^{R_2}/(\mathtt{T}_i^{(2)})^{R_1})^{1/(R_2-R_1)}$$

the bank can directly recover the identity $\mathtt{ID}$ of the defrauder. This idea was adapted in [20,21,36] to the context of divisible e-cash by replacing $F'.\mathtt{Eval}(s', i)$ with a key constrained to the appropriate subset.

However, we have explained in Section 1.2 that this process of identification is problematic and could lead to false accusations against honest user, thus breaking exculpability. Concretely, the problem arises from the fact that the above formula may output $\mathtt{ID}$ while involving tags $\mathtt{T}_i^{(1)}$ and $\mathtt{T}_i^{(2)}$ produced for different identities. Indeed, in the exculpability game, a malicious bank could cooperate with malicious users and merchants to select values $\mathtt{ID}_1$, $\mathtt{ID}_2$, R_1, R_2, s_1 and s_2 such that $((\mathtt{T}_i^{(1)})^{R_2}/(\mathtt{T}_i^{(2)})^{R_1})^{1/(R_2-R_1)} = ((\mathtt{ID}_1 \cdot (F'.\mathtt{Eval}(s_1, i_1))^{R_1})^{R_2}/(\mathtt{ID}_2 \cdot (F'.\mathtt{Eval}(s_2, i_2))^{R_2})^{R_1})^{1/(R_2-R_1)} = \mathtt{ID}$. This means that, in general, this tag construction cannot be used as it is.

To prevent this problem, our generic construction uses four PRFs, 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[14], our first PRF will be used to reveal $k_{\mathcal{S}_i}^{(1)} \leftarrow F_1.\mathtt{CKey}(s, \mathcal{S}_i)$. Likewise, our third PRF will be used to generate an element of the form[15] $\mathtt{ID}^R \cdot k_{\mathcal{S}_i}^3$, with $k_{\mathcal{S}_i}^3 \leftarrow F_3.\mathtt{CKey}(s, \mathcal{S}_i)$. 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 $k_{\mathcal{S}_i}^{(2)} \leftarrow F_2.\mathtt{CKey}(s \cdot id, \mathcal{S}_i)$, where id is some element of $\mathcal{K}$ associated with the public identity $\mathtt{ID}$, and from $\mathtt{ID} \cdot k_{\mathcal{S}_i}^4$ where $k_{\mathcal{S}_i}^4 \leftarrow F_4.\mathtt{CKey}(s, \mathcal{S}_i)$. More specifically,

$$\mathtt{SN}_j = F_1.\mathtt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(1)}, j) || F_2.\mathtt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(2)}, j) \quad \mathtt{T}_{\mathcal{S}_i} = (\mathtt{ID}^R \cdot k_{\mathcal{S}_i}^3, \mathtt{ID} \cdot k_{\mathcal{S}_i}^4).$$

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 $\mathtt{SN}_j$ still ensures that collisions can only occur for spendings involving the same master key, evaluated on the same element $j \in \mathcal{S}$.

[14] 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.

[15] We need to apply the exponent R on the identity itself instead of the constrained key to rely on the correctness of $\mathtt{CEval}$, but the principle is the same.

The last element of the double-spending tag has a more technical purpose, it prevents identification errors in the case where the colliding serial numbers have been generated using different subsets (see Remark 3).

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

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

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:

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

Since $k^3_{\mathcal{S}_{i_1}}$ and $k^3_{\mathcal{S}_{i_2}}$ are derived from the same master key, correctness ensures that $F_3.\mathtt{CEval}_{\mathcal{S}_i}(k^3_{\mathcal{S}_i}, j) = F_3.\mathtt{CEval}_{\mathcal{S}_i}(k^3_{\mathcal{S}_i}, j)$. Therefore:

$$\begin{aligned} &F_3.\mathtt{CEval}_{\mathcal{S}_i}(\mathtt{T}^{(2)}_{\mathcal{S}_i}[1], j) \cdot F_3.\mathtt{CEval}_{\mathcal{S}_i}(\mathtt{T}^{(1)}_{\mathcal{S}_i}[1], j)^{-1} \\ &= F_3.\mathtt{CEval}_{\mathcal{S}_i}(\mathtt{ID}^{R_2}, j) \cdot F_3.\mathtt{CEval}_{\mathcal{S}_i}(\mathtt{ID}^{-R_1}, j) \end{aligned}$$

The bank can then perform an exhaustive search on the set of public identities $\{\mathtt{ID}_i\}$ 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 $\mathtt{ID}^*$:

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

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

$$\begin{aligned} &F_4.\mathtt{CEval}_{\mathcal{S}_{i_1}}(\mathtt{T}^{(1)}_{\mathcal{S}_{i_1}}[2]/(\mathtt{ID}^*), j)/F_4.\mathtt{CEval}_{\mathcal{S}_{i_2}}(\mathtt{T}^{(1)}_{\mathcal{S}_{i_2}}[2]/(\mathtt{ID}^*), j) \\ &= F_4.\mathtt{CEval}_{\mathcal{S}_{i_1}}(\mathtt{ID}/(\mathtt{ID}^*), j)/F_4.\mathtt{CEval}_{\mathcal{S}_{i_2}}(\mathtt{ID}/(\mathtt{ID}^*), j) \end{aligned}$$

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

Remark 3. The use of two elements in the double-spending tag may seem surprising, in particular because the equality

$$\begin{aligned} &F_3.\mathtt{CEval}_{\mathcal{S}_{i_1}}(\mathtt{T}^{(2)}_{\mathcal{S}_{i_2}}[1], j) \cdot F_3.\mathtt{CEval}_{\mathcal{S}_{i_2}}(\mathtt{T}^{(1)}_{\mathcal{S}_{i_2}}[1], j)^{-1} \\ &= F_3.\mathtt{CEval}_{\mathcal{S}_{i_1}}(\mathtt{ID}^{R_2}, j) \cdot F_3.\mathtt{CEval}_{\mathcal{S}_{i_2}}(\mathtt{ID}^{-R_1}, j) \end{aligned}$$

still holds for the right $\mathtt{ID}$ 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 $\mathtt{ID}$, 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.* [16,18,33]). It makes use of a family of two delegatable PRFs (F_1, F_2) and two functions[16] (H, H') such that $H : \mathcal{K} \to \mathbb{G}$ and $H' : \mathcal{K}_{\mathcal{S}_i} \to \mathbb{G}$ 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. $k^{(1)}_{\mathcal{S}_i} \leftarrow F_1.\mathtt{CKey}(s, \mathcal{S}_i)$
2. $k^{(2)}_{\mathcal{S}_i} \leftarrow \mathsf{upk} \cdot F_2.\mathtt{CKey}(s, \mathcal{S}_i)$
3. $\mathtt{T}_{\mathcal{S}_i} \leftarrow \mathsf{upk} \cdot H'(F_1.\mathtt{CKey}(s, \mathcal{S}_{D(\mathcal{S}_i)}))^R$

for some public element R. The first element will be used by the bank to derive the serial numbers $\mathtt{SN}_t \leftarrow F_1.\mathtt{CEval}_{\mathcal{S}_i}(k^{(1)}_{\mathcal{S}_i}, t)\ \forall t \in \mathcal{S}_i$. 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 $(k^{(1)}_{\mathcal{S}_i}, k^{(2)}_{\mathcal{S}_i}, \mathtt{T}_{\mathcal{S}_i})$ and $(k^{(1)}_{\mathcal{S}_j}, k^{(2)}_{\mathcal{S}_j}, \mathtt{T}_{\mathcal{S}_j})$ 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:

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

Collision resistance of F_1 implies that $t_i = t_j$ and that $k^{(1)}_{\mathcal{S}_i}$ and $k^{(1)}_{\mathcal{S}_j}$ 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 $k^{(1)}_{\mathcal{S}_i}$, one can then compute $\mathtt{T}^* \leftarrow H'(F_1.\overline{\mathtt{CKey}}(k^{(1)}_{\mathcal{S}_i}, \mathcal{S}_{D(\mathcal{S}_j)}))$ and thus recover $\mathsf{upk} = \mathtt{T}_{\mathcal{S}_j}/(\mathtt{T}^*)^{R_i}$.
- 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 $\mathsf{upk} \leftarrow (\mathtt{T}^{R_j}_{\mathcal{S}_i}/\mathtt{T}^{R_i}_{\mathcal{S}_j})^{1/(R_j - R_i)}$.

[16] The requirements placed on these functions are specified in the full version [13].

4.3 Discussion

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 1.2, 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 and Gouget [16], several schemes [2, 18, 33]) implicitly followed the second framework[17] 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 [20, 21, 36] followed a different approach that actually matches our first framework. It is then possible to extract from these papers constrained key homomorphic PRFs that achieve key pseudo-randomness. 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 [36] proposed a construction supporting *any* interval $[a, b] \subseteq [1, N]$. 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 the full version [13].

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.

[17] We nevertheless note that the cut-and-choose technique used during withdrawal in [2] is very specific to this work and does not fit our framework.

$\mathtt{Exp}^{\mathtt{euf-cma}}_{\mathcal{A}}(1^\lambda)$ – 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

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) [29]: 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 $\Pr[\mathtt{Exp}^{\mathtt{euf-cma}}_{\mathcal{A}}(1^\lambda) = 1]$ is negligible for any $\mathcal{A}$.

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 be binded to the committed message, but still hiding the latter. This is formally defined by the games $\mathtt{Exp}^{\mathtt{bind}}_{\mathcal{A}}(1^\lambda)$ and $\mathtt{Exp}^{\mathtt{hid}-b}_{\mathcal{A}}(1^\lambda)$ of Figure 5. A commitment scheme is binding if $\Pr[\mathtt{Exp}^{\mathtt{bind}}_{\mathcal{A}}(1^\lambda) = 1]$ is negligible, while it is hiding if $\Pr[\mathtt{Exp}^{\mathtt{hid}-1}_{\mathcal{A}}(1^\lambda) = 1] - \Pr[\mathtt{Exp}^{\mathtt{hid}-0}_{\mathcal{A}}(1^\lambda) = 1]$ is negligible.

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 proof system is defined by the following three algorithms:

Hiding Security Game $\mathtt{Exp}^{\mathtt{hid}-b}_{\mathcal{A}}(1^\lambda)$	**Binding Security Game** $\mathtt{Exp}^{\mathtt{bind}}_{\mathcal{A}}(1^\lambda)$
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 $(b = b^*)$	1. $(\mathsf{ck}) \leftarrow \mathtt{Keygen}(1^\lambda)$ 2. $(m_0, m_1, r_0, r_1) \leftarrow \mathcal{A}(\mathsf{ck})$ 3. If $\mathtt{Commit}(\mathsf{ck}, m_0, r_0) \neq \mathtt{Commit}(\mathsf{ck}, m_1, r_1)$ or $m_0 = m_1$, 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 require 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 $\mathtt{Exp}^{\mathtt{zk}-b}_{\mathcal{A}}(1^\lambda)$, 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 $\Pr[\mathtt{Exp}^{\mathtt{zk}-1}_{\mathcal{A}}(1^\lambda)] - \Pr[\mathtt{Exp}^{\mathtt{zk}-0}_{\mathcal{A}}(1^\lambda)]$ is negligible. It is extractable if $\Pr[\mathtt{Exp}^{\mathtt{ext}}_{\mathcal{A}}(1^\lambda)]$ is negligible.

Zero-Knowledge Game $\mathtt{Exp}^{\mathtt{zk}-b}_{\mathcal{A}}(1^\lambda)$	**Extractability Game** $\mathtt{Exp}^{\mathtt{ext}}_{\mathcal{A}}(1^\lambda)$
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 $(b = b^*)$	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 $(\mathtt{crs}, w, \phi) \notin R$, then return 0 5. Return 1

Fig. 6. Security Game for NIZK Proofs

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 $\mathtt{ID}^R$ of Section 4 that was involved in double-spending tags does not make sense in a lattice setting and would in practice be replaced by $R \cdot \mathtt{ID}$ where R is a matrix and $\mathtt{ID}$ 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 $\mathtt{ID}^{R+R'} = \mathtt{ID}^R \cdot \mathtt{ID}^{R'}$ and that $(R+R') \cdot \mathtt{ID} = R \cdot \mathtt{ID} + R' \cdot \mathtt{ID}$ and so will represent $\mathtt{ID}^R$ and $R \cdot \mathtt{ID}$ by $G(\mathtt{ID}, R)$ where G is a bilinear map (see remark 5 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 5). 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[18] 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 $k_{\mathcal{S}_i}^{(1)} \leftarrow F_1.\mathtt{CKey}(s, \mathcal{S}_i)$, $k_{\mathcal{S}_i}^{(2)} \leftarrow$

[18] 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.

$F_2.\texttt{CKey}(s \cdot G_1(\mathsf{upk}), \mathcal{S}_i)$ 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 $k_{\mathcal{S}_i}^{(3)} = F_3.\texttt{CKey}(s, \mathcal{S}_i)$ and $k_{\mathcal{S}_i}^{(4)} = F_4.\texttt{CKey}(s, \mathcal{S}_i)$.
Finally, it generates a signature $\tau \leftarrow \Sigma.\texttt{Sign}(\mathsf{usk}, (\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}))$ 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. $c = \Gamma.\texttt{Commit}(\mathsf{ck}, [s, \mathsf{upk}], r)$
3. $1 = \Sigma.\texttt{Verify}(\mathsf{pk}_B, c, \sigma_c)$
4. $k_{\mathcal{S}_i}^{(1)} = F_1.\texttt{CKey}(s, \mathcal{S}_i)$
5. $k_{\mathcal{S}_i}^{(2)} = F_2.\texttt{CKey}(s \cdot G_1(\mathsf{upk}), \mathcal{S}_i)$
6. $\mathtt{T}_{\mathcal{S}_i} = (G_3(G_2(\mathsf{upk}), H(\mathsf{mpk}||\mathsf{info})) \cdot F_3.\texttt{CKey}(s, \mathcal{S}_i), G_2(\mathsf{upk}) \cdot F_4.\texttt{CKey}(s, \mathcal{S}_i))$
7. $1 = \Sigma.\texttt{Verify}(\mathsf{upk}, (\mathsf{mpk}, V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}), \tau)$

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.

– $\texttt{Deposit}(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (V, \mathsf{info}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}, \pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: 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 $\mu \leftarrow \Sigma.\texttt{Sign}(\mathsf{msk}, \mathsf{Tr})$. 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) $\Sigma.\texttt{Verify}(\mathsf{mpk}, \mathsf{Tr}, \mu) = 1$ 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 $(V', \mathsf{info}, k_{\mathcal{S}_j}^{(1)}, k_{\mathcal{S}_j}^{(2)}, \mathtt{T}_{\mathcal{S}_j}, \pi')$ along with a signature μ'. All these elements, along with $[\mathsf{Tr}, \mu]$ constitute a proof of double-deposit.
Else, the bank recovers, for all $j \in \mathcal{S}_i$ (see remark 6 below), the serial numbers $\mathtt{SN}_j \leftarrow F_1.\texttt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(1)}, j)||F_2.\texttt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(2)}, j)$. 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

– $\texttt{Identify}((V, \mathsf{info}, \mathsf{mpk}, k_{\mathcal{S}_i}^{(1)}, k_{\mathcal{S}_i}^{(2)}, \mathtt{T}_{\mathcal{S}_i}, \pi), (V', \mathsf{info}', \mathsf{mpk}', k_{\mathcal{S}_j}^{(1)}, k_{\mathcal{S}_j}^{(2)}, \mathtt{T}_{\mathcal{S}_j}, \pi'), \mathsf{bpk})$: 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 $F_1.\texttt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(1)}, x)||F_2.\texttt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}^{(2)}, x) = F_1.\texttt{CEval}_{\mathcal{S}_j}(k_{\mathcal{S}_j}^{(1)}, x')||F_2.\texttt{CEval}_{\mathcal{S}_j}(k_{\mathcal{S}_j}^{(2)}, x')$. If there is no collision, it outputs 0.
Else, it proceeds as in Section 4.1 to identify the defrauder. If $\mathtt{T}_{\mathcal{S}_i}[2] = \mathtt{T}_{\mathcal{S}_j}[2]$, it computes $R = H(\mathsf{mpk}||\mathsf{info})$, $R' = H(\mathsf{mpk}'||\mathsf{info}')$ along with

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

and $F_3.\mathtt{CEval}_{\mathcal{S}_i}(G_3(G_2(\mathsf{upk}), R), x)/F_3.\mathtt{CEval}_{\mathcal{S}_j}(G_3(G_2(\mathsf{upk}), R'), x)$ for all upk until it gets a match. It then returns the corresponding public key upk^* (or $\perp$ if the exhaustive search fails).
Else, $\mathtt{T}_{\mathcal{S}_i}[2] \neq \mathtt{T}_{\mathcal{S}_j}[2]$ and it computes

$$F_4.\mathtt{CEval}_{\mathcal{S}_i}(\mathtt{T}_{\mathcal{S}_i}[2]/G_2(\mathsf{upk}), x)/F_4.\mathtt{CEval}_{\mathcal{S}_j}(\mathtt{T}_{\mathcal{S}_{i_2}}[2]/G_2(\mathsf{upk}), x')$$

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).

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

Remark 4. An example of instantiation of our full construction, in the standard model, is provided in the full version [13] to assess the practical complexity of our framework. 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 [30] if one selects an appropriate digital signature scheme for Σ, as illustrated in our full version where we show that the complexity of our framework is very similar to the one of (unsecure) schemes from the state-of-the-art. 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 [32].

Remark 5. 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 the full version [13], 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 6. Note that, even if the bank does not know the subset $\mathcal{S}_i$, it is always able to recover all the serial numbers $\mathtt{SN}_j \leftarrow \mathtt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}, j)$, for $j \in \mathcal{S}_i$. Indeed, it can generates the list L containing $\mathtt{SN}_k \leftarrow \mathtt{CEval}_{\mathcal{S}}(k_{\mathcal{S}_i}, k)$, 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 [20]. 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 $\overline{\texttt{CEval}}$ that takes as input a constrained key and the size of the corresponding subset and that outputs $\overline{\texttt{CEval}}(k_{\mathcal{S}_i}, |\mathcal{S}_i|) = \{\texttt{CEval}_{\mathcal{S}_i}(k_{\mathcal{S}_i}, x), \forall x \in \mathcal{S}_i\}$.

The security of our construction is stated by the following theorem, proven in the full version [13].

Theorem 7. *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 pseudo-randomness, Γ is computationally hiding and Π is zero-knowledge.*

Remark 8. 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 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 intractability 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 constructions, 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

We thank Benoît Libert for very helpful discussions on the exculpability issue of previous works. This work is supported in part by the European Union PROMETHEUS Project (Horizon 2020 Research and Innovation Program, Grant Agreement no. 780701) and the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

1. Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666. Springer, Heidelberg, August 2011.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. 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.

10. 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.
11. 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.
12. 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.
13. Florian Bourse, David Pointcheval, and Olivier Sanders. Divisible e-cash from constrained pseudo-random functions. *IACR Cryptology ePrint Archive*, 2019:136, 2019.
14. 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.
15. 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.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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.

26. 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.
27. Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.
28. 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.
29. 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.
30. 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.
31. 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.
32. 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.
33. Patrick Märtens. Practical divisible e-cash. *IACR Cryptology ePrint Archive*, 2015:318, 2015.
34. 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.
35. 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.
36. 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.
37. 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.
38. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000.