# Divisible E-Cash in the Standard Model

Malika Izabachène[1] and Benoît Libert[2] *

[1] Ecole Normale Supérieure de Cachan/CNRS/INRIA (France)
[2] Université catholique de Louvain (Belgium)

**Abstract.** Off-line e-cash systems are the digital analogue of regular cash. One of the main desirable properties is anonymity: spending a coin should not reveal the identity of the spender and, at the same time, users should not be able to double-spend coins without being detected. Compact e-cash systems make it possible to store a wallet of $O(2^L)$ coins using $O(L+\lambda)$ bits, where $\lambda$ is the security parameter. They are called *divisible* whenever the user has the flexibility of spending an amount of $2^\ell$, for some $\ell \leq L$, more efficiently than by repeatedly spending individual coins. This paper presents the first construction of divisible e-cash in the standard model (*i.e.*, without the random oracle heuristic). The scheme allows a user to obtain a wallet of $2^L$ coins by running a withdrawal protocol with the bank. Our construction is built on the traditional binary tree approach, where the wallet is organized in such a way that the monetary value of a coin depends on how deep the coin is in the tree.

**Keywords.** E-Cash, provable security, anonymity, non-interactive proofs.

## 1 Introduction

Introduced by Chaum [22, 23] and developed in [24, 20, 25, 40, 29], electronic cash is the digital equivalent of regular money. It allows a user to withdraw a wallet of electronic coins from a bank so that e-coins can be spent to merchants who can then deposit them back to the bank.

The withdrawal, spending and deposit protocols should be designed in such a way that it is infeasible to determine when a particular coin was spent: even if the bank colludes with the merchant, after the deposit protocol, it should be unable to link a received coin to a particular withdrawal protocol. At the same time, users should not be able to covertly double-spend coins: should a cheating user attempt to spend a given coin twice, his identity must be exposed and evidence of his misbehavior must be given. Ideally, dishonest users should be identified without the help of a trusted third party and, as in the off-line scenario [24], the bank should preferably not intervene in the spending protocol between the user and the merchant.

* This author acknowledges the Belgian Fund for Scientific Research for his "Collaborateur scientifique" fellowship.

Related Work. In 2005, Camenisch, Hohenberger and Lysyanskaya [10] described a *compact* e-cash system allowing a user to withdraw a wallet of $2^L$ coins with a computational cost of $O(L+\lambda)$, where $\lambda$ is the security parameter, in the spending and withdrawal protocols. Using appropriate choices [27, 28] of verifiable random functions [34], they also showed how to store a wallet using only $O(L+\lambda)$ bits and additionally described a coin tracing mechanism allowing to trace all the coins of a misbehaving user. The protocol of Camenisch *et al.* was subsequently extended into e-cash systems with coin endorsement [13] or transferability properties [15, 16].

The aforementioned e-cash realizations all appeal to the random oracle model [6] – at least if the amount of interaction is to be minimized in the spending protocol – which is known not to accurately reflect world (see [19] for instance). To fill this gap, Belenkiy, Chase, Kohlweiss and Lysyanskaya [5] described a compact e-cash system with non-interactive spending in the standard model. Their construction cleverly combines multi-block extensions of P-signatures [3] and simulatable verifiable random functions [21] with the Groth-Sahai non-interactive proof systems [31]. Independently, Fuchsbauer, Pointcheval and Vergnaud also used Groth-Sahai proofs [30] to build a transferable fair (*i.e.*, involving a semi-trusted third party) e-cash system in the standard model. More recently, Blazy *et al.* [7] gave a similar construction with stronger anonymity properties.

Divisible E-Cash. In the constructions of [10], users have to run the spending protocol $N$ times if the amount to be paid is the equivalent of $N$ coins. One possible improvement is to use wallets containing coins of distinct monetary values as in [17]. Unfortunately, this approach does not allow to split individual coins of large value. This problem is addressed by divisible e-cash systems where users can withdraw a coin of value $2^L$ that can be spent in several times by dividing the value of that coin. Divisible e-cash makes it possible for users to spend the equivalent of $N = 2^\ell$ (with $0 \leq \ell \leq L$) coins more efficiently than by iterating the spending protocol $2^\ell$ times. Constructions of divisible e-cash were proposed in the 90's [36, 38, 26, 37, 20]. Okamoto provided a practical realization [37] that was subsequently improved in [20]. Unfortunately, these schemes are not fully unlinkable since several spends of a given divisible coin can be linked. To address this concern, Nakanishi and Sugiyama [35] described an unlinkable divisible e-cash system but their scheme requires a trusted third party to uncover the identity of double-spenders. In addition, by colluding with the bank, the merchant can obtain information on which part of the divisible coin the user is spending.

In 2007, Canard and Gouget [14] designed the first divisible e-cash system with full anonymity (and unlinkability) where misbehaving users can be identified without involving a trusted third party. Later on, Au *et al.* [1] came up with a more efficient implementation at the expense of substantially weaker security guarantees. More recently, Canard and Gouget [18] showed how to improve upon the efficiency of their original proposal without sacrificing the original security.

Our Contribution. Prior implementations of truly anonymous divisible e-cash all require the random oracle idealization in their security analysis. In this paper, we describe the first anonymous divisible e-cash in the standard model. Like the scheme of Belenkiy *et al.* [5], our construction relies on the Groth-Sahai non-interactive proof systems [31].

Our scheme is less efficient than the fastest random-oracle-based scheme [18] in several metrics. While the spending phase has constant (*i.e.*, independent of the value $2^L$ of the wallet) communication complexity in [18], our spending protocol requires users to transmit $O(L)$ group elements to the merchant in the worst case. On the other hand, due to the use of bounded accumulators [2], the bank has to set up a public key of size $O(2^L)$ in [18][3] whereas we only need the bank to have a key of size $O(1)$.

Achieving divisibility without resorting to random oracles requires to solve several technical issues. The solutions of Canard and Gouget [14, 18] associate each wallet with a binary tree – where nodes correspond to expandable amounts – combined with cyclic groups of *distinct* but related orders. Since these techniques do not appear compatible with the Groth-Sahai toolbox, we had to find other techniques to split the wallet across the nodes of a binary tree. In particular, we use a different method to authenticate the node corresponding to the spent divided coin in the tree.

As in the first truly anonymous construction of divisible e-cash [14], the communication complexity of our spending algorithm depends on how much the initial amount $2^L$ has to be divided: from an initial tree of value $2^L$, when a coin of value $2^\ell$ has to be spent, the communication cost of the spending phase is $O(L - \ell)$. Hence, the more we want to divide the wallet into small coins, the more expensive the spending phase is.

The downside of our e-cash construction is the complexity of the deposit phase, where the computational workload of the bank depends on the number of previously received coins when it comes to check that the received coin does not constitute a double-spending. Even though the bank can be expected to have significant computational resources (and although the double-spending checks can be performed in parallel by clerks testing a subset of previously spent coins each), this would be a real bottleneck in practice. For this reason, our system is not meant to be a practical solution and should only be seen as a feasibility result. We leave it as an open problem to build such a system with a more efficient deposit procedure from the bank's standpoint: as in previous constructions of compact e-cash (e.g. [10, 5]), the bank should only have to look up the coin's serial number in a table of previously spent coins. It would also be interesting to reduce the communication complexity of the spending phase so as to only transmit a constant number of group elements.

[3] The reason is that, in all known bounded accumulators, the public key has linear size in the maximal number of accumulated values.

## 2 Background and Definitions

### 2.1 Definitions for Divisible E-Cash

An e-cash scheme involves a bank $\mathcal{B}$, many users $\mathcal{U}$ and merchants $\mathcal{M}$ (who can be viewed as special users). All these parties interact together with respect to the following protocols:

**CashSetup**$(\lambda)$**:** takes as input a security parameter $\lambda$ and outputs the public parameters $\mathsf{params}$.

**BankKG**$(\mathsf{params}, L)$**:** generates bank's public and secret parameters $(sk_\mathcal{B}, pk_\mathcal{B})$ that allow $\mathcal{B}$ to issue wallets of up values up to $2^L$ (we assume that $L$ is part of $pk_\mathcal{B}$). It also defines an empty database $\mathrm{DB}_\mathcal{B}$ for later use.

**UserKG**$(\mathsf{params})$**:** generates a user key pair $(sk_\mathcal{U}, pk_\mathcal{U})$. We denote as $\mathcal{H}_\mathcal{U}$ the set of honestly generated public keys.

**Withdraw**$\big(\mathcal{U}(\mathsf{params}, pk_\mathcal{B}, sk_\mathcal{U}), \mathcal{B}(\mathsf{params}, pk_\mathcal{U}, sk_\mathcal{B})\big)$**:** is an interactive protocol between a user $\mathcal{U}$ and the bank $\mathcal{B}$ that allows an honest user to obtain a coin of value $2^L$. The wallet $\mathcal{W}$ comprises the coins, the user's secret key, a signature from the bank on it and some state information $\mathsf{state}$. The bank debits $\mathcal{U}$'s account and stores a piece of tracing information $\mathsf{T}_\mathcal{W}$ in a database $\mathsf{T}$ that allows uncovering the identity of double-spenders.

**Spend**$\big(\mathsf{params}, pk_\mathcal{B}, \mathcal{W}, 2^\ell, pk_\mathcal{M}, \mathtt{info}\big)$**:** is invoked by $\mathcal{U}$ to spend a coin of value $2^\ell$ from his wallet and generates a proof $\Pi$ that the coin is valid. The output is the coin that includes the proof $\Pi$ and some fresh public information $\mathtt{info}$ specifying the transaction.

**VerifyCoin**$\big(\mathsf{params}, pk_\mathcal{M}, pk_\mathcal{B}, coin, v = 2^\ell\big)$**:** allows $\mathcal{M}$ to check the validity of a given coin and output a bit depending on whether the test is successful.

**Deposit**$\big(\mathsf{params}, pk_\mathcal{B}, pk_\mathcal{M}, coin, 2^\ell, \mathrm{DB}_\mathcal{B}\big)$**:** $\mathcal{B}$ updates the database $\mathrm{DB}_\mathcal{B}$ with $\{(coin, \mathsf{flag}, 2^\ell)\}$ where $\mathsf{flag}$ indicates whether $coin$ is a valid coin of value $2^\ell$ and whether a cheating attempt is detected.

- If $coin$ does not verify, $\mathcal{B}$ rejects it and sets $\mathsf{flag}$ = "$\mathcal{M}$" to indicate a cheating merchant.
- If $coin$ verifies, $\mathcal{B}$ runs a double spending detection algorithm, using the database $\mathrm{DB}_\mathcal{B}$ containing already received coins. If an overspent is detected, the bank sets $\mathsf{flag}$ = "$\mathcal{U}$", outputs the two coins and reports the double-spending.
- If the coin passes all the tests, the bank accepts the coin, sets $\mathsf{flag}$ = "$\mathsf{accept}$" and credits the merchant's account.

**Identify**$\big(\mathsf{params}, pk_\mathcal{B}, coin_a, coin_b\big)$**:** the bank retrieves the double-spender's public key $pk_\mathcal{U}$ using its database $\mathrm{DB}_\mathcal{B}$ and the two different coins.

The security model builds on the model of non-interactive compact e-cash from [5]. An e-cash scheme is secure if it provides *Correctness, Anonymity, Balance, Identification* and *Exculpability* simultaneously.

Anonymity. Unlike the model of [14], ours adopts a simulation-based formulation of anonymity (note that simulation-based definitions are often stronger than indistinguishability-based ones). No coalition of banks and merchants should distinguish a real execution of the $\mathsf{Spend}$ protocol from a simulated one. In the security experiment, the adversary is allowed to obtain users' public keys, withdraw and spend coins using the oracles $\mathcal{Q}_{\mathsf{GetKey}}, \mathcal{Q}_{\mathsf{withdraw}}, \mathcal{Q}_{\mathsf{Spend}}$, respectively, which are defined below. Formally, an e-cash system is *anonymous* if there exists a simulator ($\mathsf{SimCashSetup}, \mathsf{SimSpend}$) such that, for any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there is a negligible function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ such that:

$$\begin{aligned}
&|\Pr[\mathsf{params} \leftarrow \mathsf{CashSetup}(\lambda); (pk_{\mathcal{B}}, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{params}) :\\
&\qquad \mathcal{A}_2^{\mathcal{Q}_{\mathsf{Spend}}(\mathsf{params}, pk_{\mathcal{B}}, \cdot, \cdot), \mathcal{Q}_{\mathsf{GetKey}}(\mathsf{params}, \cdot), \mathcal{Q}_{\mathsf{withdraw}}(\mathsf{params}, pk_{\mathcal{B}}, \cdot, \cdot)}(\mathsf{state}) = 1]\\
&\quad - \Pr[(\mathsf{params}, \mathsf{Sim}) \leftarrow \mathsf{SimCashSetup}(\lambda); (pk_{\mathcal{B}}, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{params}) :\\
&\qquad \mathcal{A}_2^{\mathcal{Q}_{\mathsf{SimSpend}}(\mathsf{params}, pk_{\mathcal{B}}, \cdot, \cdot), \mathcal{Q}_{\mathsf{GetKey}}(\mathsf{params}, \cdot), \mathcal{Q}_{\mathsf{withdraw}}(\mathsf{params}, pk_{\mathcal{B}}, \cdot, \cdot)}(\mathsf{state}) = 1]\; | < \mathsf{negl}(\lambda)
\end{aligned}$$

To formalize security against coalition of users, bank and merchants, the anonymity game allows the adversary to generate the bank's public key. It is granted dynamic access to the list of oracles hereafter and has to decide whether it is playing the real game, where the spending oracle is an actual oracle, or the simulation, where the spending oracle is a simulator.

- $\mathcal{Q}_{\mathsf{GetKey}}(\mathsf{params}, j)$: outputs $pk_{\mathcal{U}_j}$. If $\mathcal{U}_j$ does not exist, the oracle generates $(sk_{\mathcal{U}_j}, pk_{\mathcal{U}_j}) \leftarrow \mathsf{UserKG}(\mathsf{params})$ and outputs $pk_{\mathcal{U}_j}$.
- $\mathcal{Q}_{\mathsf{withdraw}}(\mathsf{params}, pk_{\mathcal{B}}, j, f)$: given a wallet identifier $f$, this oracle plays the role of user $j$ – and creates the key pair $(sk_{\mathcal{U}_j}, pk_{\mathcal{U}_j})$ if it does not exist yet – in an execution of $\mathsf{Withdraw}\big(\mathcal{U}(\mathsf{params}, pk_{\mathcal{B}}, sk_{\mathcal{U}_j}), \mathcal{A}(\texttt{states})\big)$, while the adversary $\mathcal{A}$ plays the role of the bank. The oracle then creates a wallet $\mathcal{W}_f$ of value $2^i$.
- $\mathcal{Q}_{\mathsf{Spend}}\big(\mathsf{params}, pk_{\mathcal{B}}, f, v = 2^{\ell}, pk_{\mathcal{M}}, \texttt{info}\big)$: the oracle firstly checks if wallet $\mathcal{W}_f$ has been created via an invocation of $\mathcal{Q}_{\mathsf{withdraw}}(\mathsf{params}, pk_{\mathcal{B}}, j, f)$. If not, the oracle outputs $\perp$. Otherwise, if $\mathcal{W}_f$ contains a sufficient amount, $\mathcal{Q}_{\mathsf{Spend}}$ runs $\mathsf{Spend}\big(\mathsf{params}, pk_{\mathcal{B}}, \mathcal{W}_f, i, v = 2^{\ell}, pk_{\mathcal{M}}, \texttt{info}\big)$ and outputs a coin of value $v$ from the wallet $\mathcal{W}_f$. In any other case (e.g. if the expandable amount of $\mathcal{W}_f$ is less than $2^{\ell}$), it outputs $\perp$.
- $\mathcal{Q}_{\mathsf{SimSpend}}\big(\mathsf{params}, pk_{\mathcal{B}}, f, v = 2^{\ell}, pk_{\mathcal{M}}, \texttt{info}\big)$: if $f$ is not the index of a valid withdrawn wallet obtained from $\mathcal{Q}_{\mathsf{withdraw}}$, the oracle outputs $\perp$. Otherwise, the oracle runs a simulator $\mathsf{SimSpend}$ on input of public elements $\big(\mathsf{params}, pk_{\mathcal{B}}, pk_{\mathcal{M}}, v = 2^{\ell}, \texttt{info}\big)$. Note that $\mathsf{SimSpend}$ does not use the user's wallet or his public key.

Balance. No coalition of users can spend more coins than they withdrew. The adversary is a user and can withdraw or spend coins via oracles defined below. An e-cash system provides the *Balance* property if, for any adversary, every value $L \in \mathsf{poly}(\lambda)$, we have:

$$\begin{aligned}
&\Pr[\mathsf{params} \leftarrow \mathsf{CashSetup}(\lambda); (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{BankKG}(\mathsf{params}, L);\\
&\quad (q_w, n_d) \leftarrow \mathcal{A}^{\mathcal{Q}_{\mathsf{withdraw}}(\mathsf{params}, \cdot, pk_{\mathcal{B}}, \cdot)(), \mathcal{Q}_{\mathsf{deposit}}(\mathsf{params}, pk_{\mathcal{B}}, \mathrm{DB}_{\mathcal{B}})} : q_w \cdot 2^L < n_d] < \mathsf{negl}(\lambda),
\end{aligned}$$

where $n_d$ is the total amount of deposited money after $q_d$ successful calls to oracle $\mathcal{Q}_{\mathsf{deposit}}$ (by successful, we mean that the oracle sets $\mathsf{flag}$ = "$\mathsf{accept}$"), $q_w$ is the number of successful calls to $Q_{\mathsf{Withdraw}}$.

– $\mathcal{Q}_{\mathsf{withdraw}}\big(\mathsf{params}, pk_{\mathcal{U}}, sk_{\mathcal{B}}\big)$: the oracle plays the role of the bank in an execution of the $\mathsf{Withdraw}$ protocol, on input $\big(\mathcal{A}(\mathtt{states}), \mathcal{B}(\mathsf{params}, pk_{\mathcal{U}}, sk_{\mathcal{B}})\big)$, in interaction with the adversary acting as a cheating user. At the end of the protocol, $\mathcal{Q}_{\mathsf{withdraw}}$ stores a piece of tracing information $\mathsf{T}_{\mathcal{W}}$ in a database $\mathsf{T}$.
– $\mathcal{Q}_{\mathsf{deposit}}\big(\mathsf{params}, pk_{\mathcal{B}}, pk_{\mathcal{M}}, coin, v, \mathrm{DB}_{\mathcal{B}}\big)$: this oracle plays the role of the bank while the adversary plays the role of the merchant in the protocol. The oracle initializes the bank database $\mathrm{DB}_{\mathcal{B}}$ at $\emptyset$ and returns the same response as $\mathsf{Deposit}\big(\mathsf{params}, pk_{\mathcal{B}}, pk_{\mathcal{M}}, coin, v, \mathrm{DB}_{\mathcal{B}}\big)$.

Identification. Given two fraudulent but well-formed coins, the bank should identify the double-spender. This property is defined via an experiment where the adversary $\mathcal{A}$ is the double-spender and has access to a $\mathcal{Q}_{\mathsf{withdraw}}$ oracle defined hereafter. Its goal is to deposit a coin twice without being identified by the bank. We denote by $coin_a$ and $coin_b$ the two coins produced by $\mathcal{A}$. Their corresponding entries in $\mathrm{DB}_{\mathcal{B}}$ are of the form $(coin_a, \mathsf{flag}_a, v_a, pk_{\mathcal{M}_a})$ and $(coin_b, \mathsf{flag}_b, v_b, pk_{\mathcal{M}_b})$ with $coin_a = (\mathtt{info}_a; *)$ and $coin_b = (\mathtt{info}_b; *)$, respectively. We also define a predicate $\mathsf{SameCoin}$ that given two coins $coin_a$ and $coin_b$ and their respective values $v_a$ and $v_b$, outputs 1 if the bank detects a double-spending during the deposit of $coin_a$ and $coin_b$: in the context of divisible e-cash, it means that either $coin_a$ and $coin_b$ are the same coin or that one of them, say $coin_a$, is the result of dividing the other one (and thus $v_a$ divides $v_b$). The adversary is successful if its coins satisfy $\mathsf{SameCoin}(coin_a, coin_b, v_a, v_b) = 1$ but the bank fails to identify the user using the database $\mathsf{T}$ of tracing pieces of information that were collected during executions of $\mathcal{Q}_{\mathsf{withdraw}}$. An e-cash scheme provides double-spenders identification if for any adversary $\mathcal{A}$ and any $L \in \mathsf{poly}(\lambda)$,

$$\begin{aligned}\Pr\,[&\mathsf{params} \leftarrow \mathsf{CashSetup}(\lambda); (pk_{\mathcal{B}}, sk_{\mathcal{B}}) \leftarrow \mathsf{BankKG}(\mathsf{params}, L);\\ &\big((coin_a, v_a), (coin_b, v_b)\big) \leftarrow \mathcal{A}^{\mathcal{Q}_{\mathsf{withdraw}}(\mathsf{params},\cdot,sk_{\mathcal{B}},\cdot)}(\mathsf{params}, pk_{\mathcal{B}}) :\\ &(\mathtt{info}_a; pk_{\mathcal{M}_a}) \neq (\mathtt{info}_b; pk_{\mathcal{M}_b}) \wedge\ \mathsf{SameCoin}(coin_a, coin_b, v_a, v_b) = 1\\ &\wedge\ \mathsf{VerifyCoin}(\mathsf{params}, pk_{\mathcal{M}_t}, pk_{\mathcal{B}}, coin_t, v_t) = 1 \text{ for } t \in \{a, b\}\\ &\wedge\ \mathsf{Identify}(\mathsf{params}, pk_{\mathcal{B}}, coin_a, coin_b) \notin \mathsf{T}] < \mathsf{negl}(\lambda)\end{aligned}$$

The oracle $\mathcal{Q}_{\mathsf{withdraw}}$ has the same specification as in the Balance property.

Exculpability. No coalition of bank and merchants interacting with an honest user $\mathcal{U}$ should be able to produce two coins $(coin_a, coin_b)$ such that $\mathsf{Identify}(\mathsf{params}, pk_{\mathcal{B}}, coin_a, coin_b) = pk_{\mathcal{U}}$ while user $\mathcal{U}$ never double-spent. More formally, we define a game with the challenger playing the role of an honest user and the adversary playing the role of the bank and merchants. The adversary $\mathcal{A}$ is given access to oracles $\mathcal{Q}_{\mathsf{GetKey}}, \mathcal{Q}_{\mathsf{withdraw}}, \mathcal{Q}_{\mathsf{Spend}}$ that allow it to obtain users' keys, create wallets and spend coins. The exculpability property holds if, for any

PPT adversary $\mathcal{A}$, we have

$$\begin{aligned}&\Pr\,[\mathsf{params} \leftarrow \mathsf{CashSetup}(\lambda);\ (pk_{\mathcal{B}}, st) \leftarrow \mathcal{A}(\mathsf{params});\\ &\quad (pk_{\mathcal{B}}, coin_a, coin_b, v_a, v_b)\\ &\qquad \leftarrow \mathcal{A}^{\mathcal{Q}_{\mathsf{Spend}}(\mathsf{params}, pk_{\mathcal{B}},\cdot,\cdot), \mathcal{Q}_{\mathsf{GetKey}}(\mathsf{params},\cdot), \mathcal{Q}_{\mathsf{withdraw}}(\mathsf{params},\cdot,\cdot,\cdot)}(\mathsf{params}, st);\\ &\quad \mathsf{SameCoin}(coin_a, coin_b, v_a, v_b) = 1;\\ &\quad pk_{\mathcal{U}} \leftarrow \mathsf{Identify}(\mathsf{params}, coin_a, coin_b) : pk_{\mathcal{U}} \in \mathcal{H}_{\mathcal{U}}] < \mathsf{negl}(\lambda),\end{aligned}$$

where $\mathcal{H}_{\mathcal{U}}$ denotes the set of honest users. Oracles $\mathcal{Q}_{\mathsf{GetKey}}$, $\mathcal{Q}_{\mathsf{withdraw}}$ and $\mathcal{Q}_{\mathsf{Spend}}$ are defined exactly as in the notion of anonymity.

## 2.2 F-Unforgeable Signatures

Since the e-cash construction described in the paper relies on a common reference string, the following algorithms all take as input a set of common public parameters $\mathsf{params}_{\mathsf{GS}}$. To lighten notations, we omit to explicitly write them in the syntax hereafter.

**Definition 1.** *A multi-block signature scheme consists of efficient algorithms* $\Sigma = (\mathsf{SigSetup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *with the following specification.*

**SigSetup**$(\lambda)$**:** *takes as input a security parameter* $\lambda \in \mathbb{N}$ *and outputs* $\mathsf{params}$ *that gives the length* $n \in \mathsf{poly}(\lambda)$ *of message vectors to be signed.*

**Keygen**$(\mathsf{params})$**:** *takes as input the public parameters and outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$.

**Sign**$(\mathsf{sk}, \vec{m})$**:** *is a (possibly randomized) algorithm that takes in a private key* $\mathsf{sk}$ *and a vector* $\vec{m} = (m_1, \ldots, m_n)$ *of messages. It outputs a signature* $\sigma$.

**Verify**$(\mathsf{pk}, \vec{m}, \sigma)$**:** *is a deterministic algorithm that takes as input a public key* $\mathsf{pk}$*, a signature* $\sigma$ *and a message vector* $\vec{m} = (m_1, \ldots, m_n)$*. It outputs* $1$ *if* $\sigma$ *is deemed valid for* $\vec{m}$ *and* $0$ *otherwise.*

**Definition 2 ([5]).** *A multi-block signature scheme* $\Sigma$ *is F-unforgeable, for some injective function* $F(.)$*, if no probabilistic polynomial time (PPT) adversary has non-negligible advantage in the following game:*

1. *The challenger runs* $\mathsf{Setup}$ *and* $\mathsf{Keygen}$ *to obtain a pair* $(\mathsf{pk}, \mathsf{sk})$*, it then sends* $\mathsf{pk}$ *to* $\mathcal{A}$.
2. $\mathcal{A}$ *adaptively queries a signing oracle. At each query* $i$*,* $\mathcal{A}$ *chooses a vector* $\vec{m} = (m_1, \ldots, m_n)$ *and obtains* $\sigma_i = \mathsf{Sign}(\mathsf{sk}, \vec{m})$.
3. *The adversary* $\mathcal{A}$ *outputs a pair* $\big((F(m_1^\star), \ldots, F(m_n^\star)), \sigma^\star\big)$ *and wins if: (a)* $\mathsf{Verify}(\mathsf{pk}, m^\star, \sigma^\star) = 1$*; (b)* $\mathcal{A}$ *did not obtain any signature on the vector* $(m_1^\star, \ldots, m_n^\star)$.

**Definition 3 ([5]).** *A multi-block P-signature combines an F-unforgeable multi-block signature scheme* $\Sigma$ *with a commitment scheme* $(\mathsf{Com}, \mathsf{Open})$ *and three protocols:*

1. *An algorithm* $\mathsf{SigProve}(\mathsf{params}, \mathsf{pk}, \sigma, \vec{m} = (m_1, \ldots, m_n))$ *that generates a series of* $n$ *commitments* $C_\sigma$, $C_{m_1}, \ldots, C_{m_n}$ *and a NIZK proof*

$$\pi \leftarrow \mathsf{NIZPK}\big(m_1 \text{ in } C_{m_1}, \ldots, m_n \text{ in } C_{m_n},\ \sigma \text{ in } C_\sigma \\ \mid \{(F(m_1), \ldots, F(m_n), \sigma) : \mathsf{Verify}(\mathsf{pk}, \vec{m}, \sigma) = 1\}\big)$$

*and the corresponding* $\mathsf{VerifyProof}(\mathsf{params}, \mathsf{pk}, C_{m_1}, \ldots, C_{m_n}, C_\sigma)$ *algorithm.*
2. *A NIZK proof that two commitments open to the same value, i.e., a proof for the relation*

$$R = \{((x, y), (open_x, open_y)) \\ \mid C = \mathsf{Com}(x, open_x)\ \wedge\ D = \mathsf{Com}(y, open_y)\ \wedge\ x = y\}.$$

3. *A protocol* $\mathsf{SigIssue} \rightleftarrows \mathsf{SigObtain}$ *allowing a user to obtain a signature on a committed vector* $\vec{m} = (m_1, \ldots, m_n)$ *without letting the signer learn any information on* $\vec{m}$.

## 2.3 Bilinear Maps and Complexity Assumptions

We consider a configuration of *asymmetric* bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of prime order $p$ with a mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that: (1) $e(g^a, h^b) = e(g, h)^{ab}$ for any $(g, h) \in \mathbb{G}_1 \times \mathbb{G}_2$ and $a, b \in \mathbb{Z}$; (2) $e(g, h) \neq 1_{\mathbb{G}_T}$ whenever $g \neq 1_{\mathbb{G}_1}$ and $h \neq 1_{\mathbb{G}_2}$. Since we rely on the hardness of DDH in $\mathbb{G}_1$ and $\mathbb{G}_2$, we additionally require that no isomorphism be efficiently computable between $\mathbb{G}_2$ and $\mathbb{G}_1$.

**Definition 4.** *The* $q$**-Hidden Strong Diffie-Hellman problem** *(q-HSDH) in* $(\mathbb{G}_1, \mathbb{G}_2)$ *is, given* $(g, u, h, \Omega = h^\omega) \in \mathbb{G}_1^2 \times \mathbb{G}_2^2$ *and tuples* $(g^{1/(\omega+c_i)}, g^{c_i}, h^{c_i}, u^{c_i})$ *with* $c_1, \ldots, c_q \overset{R}{\leftarrow} \mathbb{Z}_p^*$, *finding* $(g^{1/(\omega+c)}, h^c, u^c)$ *such that* $c \neq c_i$ *for* $i = 1, \ldots, q$.

**Definition 5.** *The* $q$**-Decision Diffie-Hellman Inversion problem** *(q-DDHI) in* $(\mathbb{G}_1, \mathbb{G}_2)$ *consists in, given* $(g, g^{(\alpha)}, \ldots, g^{(\alpha^q)}) \in \mathbb{G}_1^{q+1}$ *and* $\eta \in \mathbb{G}_1$, *deciding if* $\eta = g^{1/\alpha}$ *or* $\eta \in_R \mathbb{G}_1$.

**Definition 6 ([3]).** *The* **Triple Diffie-Hellman problem** *(TDH) in* $(\mathbb{G}_1, \mathbb{G}_2)$ *is, given a tuple* $(g, g^a, g^b, h, h^a) \in \mathbb{G}_1^3 \times \mathbb{G}_2^2$, *and pairs* $(c_i, g^{1/a+c_i})_{i=1,\ldots,q}$ *where* $a, b, c_1, \ldots, c_q \overset{R}{\leftarrow} \mathbb{Z}_p^*$, *to find a triple* $(g^{\mu b}, h^{\mu a}, g^{\mu ab})$ *such that* $\mu \neq 0$.

**Definition 7.** *The* **Decision 3-party Diffie-Hellman problem** *(D3DH) in* $(\mathbb{G}_1, \mathbb{G}_2)$ *is, given elements* $(g, g^a, g^b, g^c, h, h^a, h^b, h^c, \Gamma) \in \mathbb{G}_1^4 \times \mathbb{G}_2^4 \times \mathbb{G}_1$, *where* $a, b, c \overset{R}{\leftarrow} \mathbb{Z}_p$, *to decide if* $\Gamma = g^{abc}$ *or* $\Gamma \in_R \mathbb{G}_1$.

## 2.4 Building Blocks

**Non-interactive witness indistinguishable proofs.** Our construction uses Groth-Sahai proofs for pairing product equations (PPE) of the form:

$$\prod_{j=1}^{n} e(\mathcal{A}_j, \mathcal{Y}_j) \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{B}_i) \prod_{i=1}^{m} \prod_{j=1}^{n} e(\mathcal{X}_i, \mathcal{Y}_j)^{\gamma_{i,j}} = t_T,$$

where $\mathcal{X}_i, \mathcal{Y}_j$ are variables in $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and $\mathcal{A}_j \in \mathbb{G}_1, \mathcal{B}_i \in \mathbb{G}_2$ and $t_T \in \mathbb{G}_T$ are constants for $i \in [1, m]$ and $j \in [1, n]$.

A proof system is a tuple of four algorithms ($\mathsf{Setup}_{\mathsf{GS}}, \mathsf{Prove}_{\mathsf{GS}}, \mathsf{VerifyProof}_{\mathsf{GS}}$): $\mathsf{Setup}_{\mathsf{GS}}$ outputs a common reference string (CRS) $crs$, $\mathsf{Prove}_{\mathsf{GS}}$ first generates commitments of variables and constructs proofs that these variables satisfy the statement, and $\mathsf{VerifyProof}_{\mathsf{GS}}$ verifies the proof. GS proofs are witness-indistinguishable and some of these can be made zero-knowledge as shown later. The proofs satisfy correctness, soundness and witness-indistinguishability. *Correctness* requires that a verifier always accepts honestly generated proofs for true statements. *Soundness* guarantees that cheating provers can only prove true statements. *Witness-indistinguishability* requires that an efficient simulator $\mathsf{GSSimSetup}$ should be able to produce a common reference string (CRS) $crs'$ that is computationally indistinguishable from a normal $crs$. When commitments are computed using $crs'$, they are perfectly hiding and the corresponding non-interactive proofs are witness indistinguishable: *i.e.*, they leak no information on the underlying witnesses. *Zero-knowledge* additionally requires the existence of an algorithm $\mathsf{GSSimProve}$ that, given a simulated CRS $crs'$ and some trapdoor information $\tau$, generates a simulated proof of the statement without using the witnesses and in such a way that the proof is indistinguishable from a real proof.

As a building block, we will use a NIZK proof of equality of committed group elements as defined in [3, 5].

If $C_x = \mathsf{GSCom}(x, open_x)$ and $C_y = \mathsf{GSCom}(y, open_y)$ are Groth-Sahai commitments to the group element $x = y \in \mathbb{G}_1$, the NIZK proof can be a proof that committed variables $(x, y, \theta) \in \mathbb{G}_1^2 \times \mathbb{Z}_p$ satisfy the equations $e(x/y, h^\theta) = 1$ and $e(g, h^\theta)e(1/g, h) = 1_{\mathbb{G}_T}$. Using the trapdoor of the CRS, we can trapdoor open to 1 a commitment to 0 and generate fake proofs for the latter relation. Setting $\theta = 0$ and $\theta = 1$, respectively, allows to construct a valid (simulated) witness for each of the two equations. Under the SXDH assumption, commitments cost 2 elements in the group. Thus, if the commitment to $y$ is in $\mathbb{G}_1^2$, the proof above costs 8 elements $\mathbb{G}_1$ and 6 in $\mathbb{G}_2$, 6 multi-exponentiations and 26 pairings (to verify). This includes commitments to $y \in \mathbb{Z}_p$ and $h^\theta$.

**Multi Block P-signatures.** In [5], a multi-block P-signature was proved F-secure under the HSDH and the TDH assumptions. Let $(p, \mathbb{G}_1, \mathbb{G}_2, G_T, e, g, h)$ be parameters for a bilinear map, the public parameters are then defined as $(p, \mathbb{G}_1, \mathbb{G}_2, G_T, e, g, h, \mathsf{params}_{\mathsf{GS}}, e(g, h))$, where $g$ and $h$ are random elements of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively. The public key and the private key are defined to be $\mathsf{pk} = (u, U = g^\beta, \tilde{U} = h^\beta, \{V_i = g^{a_i}, \tilde{V}_i = h^{a_i}\}_{i=1}^n)$ and $\mathsf{sk} = (\beta, \vec{a} = (a_1, \ldots, a_n))$, where $u \xleftarrow{R} \mathbb{G}_1$ and for random scalars $\beta, a_1, \ldots, a_n$. To sign a vector of message $\vec{m} = (m_1, \ldots, m_n)$, the signer chooses $r \xleftarrow{R} \mathbb{Z}_p$ such that $r \neq -(\beta + \sum_{i=1}^n a_i m_i)$ and computes $\sigma = (g^{1/\beta + r + \sum_{i=1}^n a_i m_i}, h^r, u^r)$. Verification of a signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ on some block $\vec{m}$ is done by checking whether

$$e\big(\sigma_1, \tilde{U} \cdot \sigma_2 \cdot \prod_{i=1}^n \tilde{V}_i^{m_i}\big) = e(g, h) \qquad \text{and} \qquad e(u, \sigma_2) = e(\sigma_3, h).$$

As shown in [5], the above scheme can be augmented with the following P-signature protocols.

**SigProve**$(\mathsf{params}, \mathsf{pk}, \sigma, \vec{m})$**:** parse the signature $\sigma$ as $(\sigma_1, \sigma_2, \sigma_3)$ and the vector $\vec{m}$ as $(m_1, \ldots, m_n)$. To commit to an exponent $m_i \in \mathbb{Z}_p$, compute Groth-Sahai commitments of $h^{m_i}$ and $u^{m_i}$ as

$$\begin{aligned}(C_{i,1}, C_{i,2}, C_{i,3}) &= \mathsf{Com}\big(m_i, (open_{m_i,1}, open_{m_i,2}, open_{m_i,3})\big)\\ &= \big(\mathsf{GSCom}(h^{m_i}, open_{m_i,1}),\\ &\qquad \mathsf{GSCom}(u^{m_i}, open_{m_{i,2}}), \mathsf{GSCom}(\tilde{V}_i^{m_i}, open_{m_{i,3}})\big).\end{aligned}$$

Generate an auxiliary variable $\theta = 1 \in \mathbb{Z}_p$ with its own commitment $C_\theta = \mathsf{GSCom}(\theta, open_\theta)$. Then, generate commitments $\{C_{\sigma_\tau}\}_{\tau=1}^3$ to $\{\sigma_\tau\}_{\tau=1}^3$ and give a NIZK proof that

$$\begin{aligned} e(g^\theta, h) &= e\big(\sigma_1, \tilde{U} \cdot \sigma_2 \cdot \prod_{i=1}^n \tilde{V}_i^{m_i}\big),\\ e(u, \sigma_2) &= e(\sigma_3, h),\\ \theta &= 1\\ e(g, \tilde{V}_i^{m_i}) &= e(V_i, h^{m_i}), \qquad e(u, h^{m_i}) = e(u^{m_i}, h) \qquad \text{for } i \in \{1, \ldots, n\}\end{aligned}$$

We denote the complete proof by

$$\pi^{sig} = \big(\{C_{\sigma_\tau}\}_{\tau=1}^3, \pi_1^{sig}, \pi_2^{sig}, \pi_\theta^{sig}, \{\pi_{m_i,1}^{sig}, \pi_{m_i,2}^{sig}\}_{i=1}^n\big).$$

Note that $\pi_1^{sig}$ is a proof for a quadratic equation and requires 4 elements of $\mathbb{G}_1$ and 4 elements of $\mathbb{G}_2$. Other equations are linear: each of $\pi_2^{sig}$ and $\{\pi_{m_i,2}^{sig}\}_{i=1}^n$ demands 2 elements of $\mathbb{G}_1$ and 2 elements of $\mathbb{G}_2$ whereas proofs $\{\pi_{m_i,1}^{sig}\}_{i=1}^n$ only takes two elements of $\mathbb{G}_1$ each since all variables are in $\mathbb{G}_2$. The NIZK property stems from the fact that, on a simulated CRS, a commitment to 0 can be trapdoor opened to 1. For this reason, except for the equation $\theta = 1$ (for which one can simply equivocate the commitment), all other proofs can be simulated using the witnesses $1_{\mathbb{G}_1}$, $1_{\mathbb{G}_2}$ and $\theta = 0$.

**VerifyProof**$(\mathsf{params}, \mathsf{pk}, \pi^{sig}, (C_1, \ldots, C_n))$ Works in the obvious way and returns 1 if and only if the proof $\pi^{sig}$ generated by $\mathsf{SigProve}$ is convincing.

**EqComProve**$(\mathsf{params}, \mathsf{pk}, x, y)$ The protocol for proving that two commitments open to the same value employ the usual technique already used in [3, 5] and is reviewed section 2.4.

**SigIssue**$(\mathsf{sk}, (C_1, \ldots, C_n)) \leftrightarrows$ **SigObtain**$(\mathsf{params}, \mathsf{pk}, \vec{m}, \{(C_i, open_i)\}_{i=1}^n)$ is a secure two-party protocol between the issuer and the receiver where the latter obtains a signature on a committed vector of messages. As suggested in [5], this can be done using the 2-party protocol of [32] for computing a circuit on committed inputs. Another option would be to use the two-party computation protocol from [4] that relies on homomorphic encryption.

**Theorem 1 ([5]).** *If the HSDH and the TDH assumptions hold in* $(\mathbb{G}_1, \mathbb{G}_2)$*, the scheme is F-unforgeable w.r.t. the injective function* $F(m) = (h^m, u^m)$.

## 3 Construction of Divisible E-cash

Known approaches for divisible e-cash [37, 35, 15, 18] make use of a binary tree with $L + 1$ levels (for a monetary value of $2^L$) where each node corresponds to an amount of money which is exactly one of half the amount of its father. Double-spenders are detected by making sure that each user cannot spend a coin corresponding to a node and one of its descendants or one of its ancestors.

In these tree-based constructions, one difficulty is for the user to efficiently prove that the path connecting the spent node to the root is well-formed. In [14, 18], this problem is solved using groups of distinct order: [14] uses a sequence of $L+1$ groups $\mathbb{G}_1, \ldots, \mathbb{G}_{L+1}$ of prime order $p_\nu$ where $\mathbb{G}_\nu$ is a subgroup of $\mathbb{Z}^\star_{p_{\nu-1}}$ for $\nu = 1, \ldots, L + 1$. The solution of [18] uses $L + 2$ bounded accumulators (one for each level of the tree and one for the whole tree) so as to only use two distinct group orders. The use of groups of distinct order (and double discrete logarithms in [14]) is hardly compatible with Groth-Sahai proofs and, in our system we need to find a different technique to prevent users from spending coins associated with a node and one of its ancestors in the same tree.

### 3.1 General Description of the Scheme

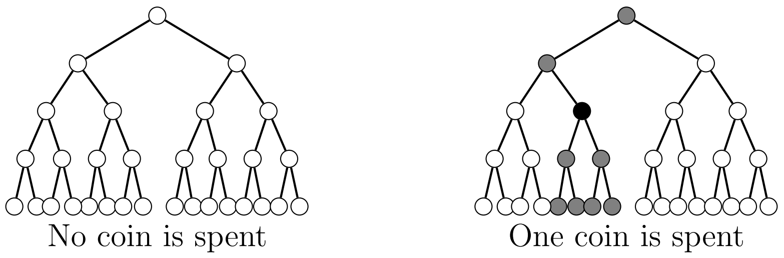

**Fig. 1.** Binary tree for spending one coin in a wallet of $2^4$ coins

Our construction uses the tree-based approach. Each wallet $\mathcal{W}$ consists of a divisible coin of value $2^L$, for some $L \in \mathbb{N}$, and the complexity of the spending phase depends on the depth of the node in the tree $\mathcal{W}$: the deeper the node is, the more expensive the spending phase will be. When an honest user $\mathcal{U}$ with key pairs $(pk_\mathcal{U}, sk_\mathcal{U})$ interacts with the bank $\mathcal{B}$, he obtains a wallet $\mathcal{W} = (s, t, sk_\mathcal{U}, \sigma, \mathsf{state})$ consisting of the bank's signature $\sigma$ on the vector $(s, t, sk_\mathcal{U})$ where $s, t$ are seeds for the Dodis-Yampolskiy PRF [28]. In our notation, $\mathsf{state}$ is a variable indicating the availability of coins.

To spend a coin of value $v = 2^\ell$ (with $\ell \leq L$) in the tree, the user $\mathcal{U}$ determines the next node corresponding to an unspent coin at height $\ell$: the root of the tree is used if the user wants to spend his entire wallet at once whereas the leaves correspond to the smallest expandable amounts. Each node will be assigned a unique label consisting of an integer in the interval $[1, 2^{L+1} - 1]$. A simple assignment is obtained by labeling the root as $x_0 = 1$ and the rightmost leaf as $2^{L+1} - 1$, all other nodes being considered in order of appearance, from the left to the right and starting from the root.

In order to construct a valid coin, the user has to choose a previously unspent node of label $x_{coin}$ at the appropriate level and do the following: (1) Prove his knowledge of a valid signature on committed messages $(s, t, sk_{\mathcal{U}})$ and his knowledge of $sk_{\mathcal{U}}$. (2) Commit to the PRF seeds via commitments to the group elements $(S, T) = (h^s, h^t)$. (3) Commit to the path that connects $x_{coin}$ to the root and prove that commitments pertain to a valid path. (4) Evaluate a coin serial number $Y_{L-\ell} = g^{1/(s+x_{coin})}$ where the input is the label of the node to be spent. (5) Generate NIZK proofs that everything was done consistently. (6) Add some material that makes it impossible to subsequently spend an ancestor or a descendant of $x_{coin}$ without being detected.

At step (1), we use the multi-block P-signature scheme to sign the block $(s, t, sk_{\mathcal{U}})$. Using the proof produced by $\mathsf{SigProve}$ in the P-signature, we can efficiently prove knowledge of a signature on committed inputs in NIZK.

The trickiest problem to solve is actually (6). If $\{x_0, \ldots, x_{L-\ell}\}$ denotes the path from the root $x_0$ to $x_{L-\ell} = x_{coin}$, for each $j \in \{0, \ldots, L - \ell\}$, we include in the coin a pair $(T_{j,1}, T_{j,2})$ where $T_{j,1} = h^{\delta_{j,1}}$, for some random $\delta_{j,1} \xleftarrow{R} \mathbb{Z}_p$, and $T_{j,2} = e(Y_j, T_{j,1})$, where $Y_j = g^{1/(s+x_j)}$ is the value of the PRF for the label $x_j$. In addition, $\mathcal{U}$ must add a NIZK proof that the pair $(T_{j,1}, T_{j,2})$ was correctly calculated. By doing so, at the expense of $n_s$ pairing evaluations at each deposit (where $n_s$ denotes the number of previously spent coins), the bank will be able to detect whether a spent node is in the path connecting a previously spent node to the root. At the same time, if $\mathcal{U}$ does not overspend at any time, the coins he spends remain computationally unlinkable.

By itself, the pair $(T_{j,1}, T_{j,2})$ only renders cheating attempts evident. In order to expose the public key $pk_{\mathcal{U}}$ of double spenders, $\mathcal{U}$ is required to add a pair $(T_{j,3}, T_{j,4}) = (h^{\delta_{j,2}}, pk_{\mathcal{U}} \cdot e(Y_j, T_{j,3}))$ at each node of the path: by doing so, $pk_{\mathcal{U}}$ is exposed if $\mathcal{U}$ subsequently spends a node above $x_{coin}$ in the path. However, we have to consider a second kind of double-spending, where the two coins involve the same tree node $x_{coin} = x_{L-\ell}$. To deal with this case, we require $\mathcal{U}$ to additionally use the seed $t$ of his wallet and the merchant's data $R$ and compute another security tag $Z_{L-\ell} = g^{sk_{\mathcal{U}}} g^{R/(t+x_{L-\ell})}$. The latter will be used to identify cheating users in the same way as in [10].

Finally, in order to obtain the exculpability property (and prevent a dishonest bank from wrongly accusing the user of double-spending coins), we need to add yet another pair of the form $(h^{\delta_{j,3}}, e(g_0, h^{sk_{\mathcal{U}}}) \cdot e(Y_j, h^{\delta_3}))$, where $g_0 \in \mathbb{G}_1$ is part of the CRS, in such a way that framing the user requires to compute $e(g_0, h^{sk_{\mathcal{U}}})$ and solve a (computational) Bilinear Diffie-Hellman instance.

In order to solve problem (5), we need to generate non-interactive proofs for a number of pairing-product equations. Since the notion of anonymity requires to build a simulator that emulates the prover without knowing any witness, it means that we need NIZK proofs for pairing product equations on multiple occasions. Fortunately, the specific equations fall into the category of equations for which NIZK proofs are possible at the cost of introducing extra variables. For this reason, we will have to introduce auxiliary variables for each pairing product equations.

**Example.** Suppose that, in his wallet $L = 4$, $\mathcal{U}$ uses the seed $s$ to spend the amount of $v = 2^2$. The left part of Figure 1 represents the state of the wallet when no coin has been spent. In the rightmost tree, the black node indicates the target node $x_{coin}$ of value $v$ and greyed nodes are those that cannot be spent any longer once the black node was spent.

## 3.2 Construction

We now describe our divisible e-cash system where the withdrawal protocol allows users to obtain a wallet of a divisible coin of value $2^L$.

**$\mathsf{CashSetup}(\lambda)$:** chooses bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ of order $p > 2^\lambda$ and generators $g, g_0 \overset{R}{\leftarrow} \mathbb{G}_1$, $h \overset{R}{\leftarrow} \mathbb{G}_2$. It also generates a Groth-Sahai common reference string $\mathsf{params}_{GS} = \{g, h, \vec{u}_1, \vec{u}_2, \vec{v}_1, \vec{v}_2\}$ for the perfectly soundness setting. The algorithm also selects a collision-resistant hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$. The output is $\mathsf{params} := \{(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T),\ g_0,\ \mathsf{params}_{GS},\ H\}$.

**$\mathsf{BankKG}(\mathsf{params}, L)$:** runs $\mathsf{SigSetup}(\lambda, n)$ with $n = 3$ to obtain a key pair $(\mathsf{sk}, \mathsf{pk})$ for the P-signature of section 2.4. The bank's key pair is defined to be $(sk_\mathcal{B}, pk_\mathcal{B}) = (\mathsf{sk}, \mathsf{pk})$ and $pk_\mathcal{B}$ consists of

$$pk_\mathcal{B} = \big(u, U = g^\beta, \tilde{U} = h^\beta, \{V_i = g^{a_i}, \tilde{V}_i = h^{a_i}\}_{i=1}^3,\ L\big).$$

**$\mathsf{UserKG}(\mathsf{params})$:** the user $\mathcal{U}$ defines his key pair as $(sk_\mathcal{U}, pk_\mathcal{U} = e(g,h)^{sk_\mathcal{U}})$ for a random $sk_\mathcal{U} \overset{R}{\leftarrow} \mathbb{Z}_p$.

**$\mathsf{Withdraw}\big(\mathcal{U}(\mathsf{params}, pk_\mathcal{B}, sk_\mathcal{U}), \mathcal{B}(\mathsf{params}, pk_\mathcal{U}, sk_\mathcal{B})\big)$:** $\mathcal{U}$ and $\mathcal{B}$ run the following interactive protocol:

1. The user $\mathcal{U}$ first picks $s', t' \overset{R}{\leftarrow} \mathbb{Z}_p$ at random and computes perfectly hiding commitments $C_{s'} = \mathsf{Com}(s', open_{s'})$, $C_{t'} = \mathsf{Com}(t', open_{t'})$ and $C_{sk_\mathcal{U}} = \mathsf{Com}(sk_\mathcal{U}; open_{sk_\mathcal{U}})$. The user sends $(C_{s'}, C_{t'}, C_{sk_\mathcal{U}})$ to $\mathcal{B}$ and provides interactive witness indistinguishable proofs that he knows how to open $(C_{s'}, C_{t'})$. In addition, he provides an interactive zero-knowledge[4] proof that $C_{sk_\mathcal{U}}$ is a commitment to the private key $sk_\mathcal{U}$ that was used to generate $pk_\mathcal{U}$.
2. If the proofs verifies, $\mathcal{B}$ picks $(s'', t'') \leftarrow \mathbb{Z}_p^2$ which are sent to $\mathcal{U}$.
3. The user $\mathcal{U}$ sets $s = s' + s''$ and $t = t' + t''$, updates commitments $C_{s'}$ and $C_{t'}$ into commitments $C_s = \mathsf{Com}(s, open_s)$ and $C_t = \mathsf{Com}(t, open_t)$. The user sends $(C_s, C_t)$ to the bank with a proof that these commitments were properly calculated.

[4] The zero-knowledge property will be needed in the proof of weak exculpability.

4. $\mathcal{U}$ and $\mathcal{B}$ jointly run the protocol

$$\begin{aligned}&\mathsf{SigIssue}(\mathsf{params}, \mathsf{sk}, (C_s, C_t, C_{sk_{\mathcal{U}}}))\\ &\qquad\qquad \leftrightarrows \mathsf{SigObtain}(\mathsf{params}, \mathsf{pk}, (s, t, sk_{\mathcal{U}}), (open_s, open_t, open_{sk_{\mathcal{U}}}))\end{aligned}$$

in such a way that $\mathcal{U}$ eventually obtains $\mathcal{B}$'s signature $\sigma$ on $(s, t, sk_{\mathcal{U}})$. The user $\mathcal{U}$ stores the wallet $\mathcal{W} = (s, t, sk_{\mathcal{U}}, \sigma, \mathsf{state})$, where $\mathsf{state} = \emptyset$.
5. $\mathcal{B}$ records a debit of value $v = 2^L$ on $\mathcal{U}$'s account. It stores the transcript of the protocol and the tracing information $pk_{\mathcal{U}}$ in its database $\mathsf{T}$.

**Spend**$\big(\mathsf{params}, pk_{\mathcal{B}}, \mathcal{W} = (s, t, sk_{\mathcal{U}}, \sigma, \mathsf{state}), 2^\ell, pk_{\mathcal{M}}, \mathtt{info}\big)$**:** Let us assume that $\mathcal{U}$ wants to spend a coin of value $2^\ell$ for the wallet $\mathcal{W}$ of initial value $2^L$. Using $\mathsf{state}$, $\mathcal{U}$ determines the label $x_{coin} \in [1, 2^{L+1} - 1]$ of the first node corresponding to an unspent coin at height $\ell$ in the tree associated with the wallet. Let $\{x_0, x_1, \ldots, x_{L-\ell}\}$ denote the path connecting node $x_{coin} = x_{L-\ell}$ to the root $x_0 = 1$ of the tree. The user $\mathcal{U}$ computes $S = h^s$ and $T = h^t$ and conducts the following steps.

1. $\mathcal{U}$ has to prove that he knows a signature $\sigma$ on the committed vector $(s, t, sk_{\mathcal{U}}) \in \mathbb{Z}_p^3$. To this end, he first generates commitments and proofs $\big(\{C_{S,i}\}_{i=1}^3, \{C_{T,i}\}_{i=1}^3, \{C_{\mathcal{U},i}\}_{i=1}^3, \pi^{sig}\big) \leftarrow \mathsf{SigProve}(\mathsf{params}, \mathsf{pk}, \sigma, (s, t, sk_{\mathcal{U}}))$. The output of $\mathsf{SigProve}$ includes $\{C_{\mathcal{U},i}\}_{i=1}^3$, which are commitments to $(L_{\mathcal{U},1}, L_{\mathcal{U},2}, L_{\mathcal{U},3}) = (h^{sk_{\mathcal{U}}}, u^{sk_{\mathcal{U}}}, \tilde{V}_3^{sk_{\mathcal{U}}})$, and $\{C_{S,i}, C_{T,i}\}_{i=1}^3$, that contain $(L_{S,1}, L_{S,2}, L_{S,3}) = (h^s, u^s, \tilde{V}_1^s)$ and $(L_{T,1}, L_{T,2}, L_{T,3}) = (h^t, u^t, \tilde{V}_2^t)$, respectively. In addition, $\mathcal{U}$ computes $C_{K_{\mathcal{U}}} = \mathsf{GSCom}(h^{sk_{\mathcal{U}}}, open'_{\mathcal{U}})$ as a commitment to $K_{\mathcal{U}} = h^{sk_{\mathcal{U}}}$ and generates a NIZK proof $\pi_{K_{\mathcal{U}}} \leftarrow \mathsf{EqComProve}(L_{\mathcal{U},1}, K_{\mathcal{U}})$ that $C_{K_{\mathcal{U}}}$ and $C_{\mathcal{U},1}$ are commitment to the same value. This amounts to prove that
$$e(L_{\mathcal{U},1}/K_{\mathcal{U}}, h^\theta) = 1_{\mathbb{G}_T} \quad \text{and} \quad \theta = 1, \qquad (1)$$
for some variable $\theta \in \mathbb{Z}_p$ contained in $C_\theta = \mathsf{GSCom}(\theta, open_\theta)$ and that will be re-used in subsequent steps of the spending protocol.
2. For $j = 0$ to $L - \ell$ do the following.
   a. If $j > 0$, generate a commitment $C_{X_j} = \mathsf{GSCom}(h^{x_j}, open_{x_j})$ to $X_j = h^{x_j}$ and a proof that $x_j = 2x_{j-1} + b_j$, for some bit $b_j \in \{0, 1\}$. To this end, generate the commitments $C_{b_j} = \mathsf{GSCom}(g^{b_j}, open_{b_j})$ and $C'_{b_j} = \mathsf{GSCom}(h^{b_j}, open'_{b_j})$ as well as a NIZK proof $\pi_{x_j} \leftarrow \mathsf{EqComProve}(C_{X_j}, C'_{X_j})$ that $C_{X_j}$ and $C'_{X_j} = C^2_{X_{j-1}} \cdot C'_{b_j}$ open to the same value. To prove that $b_j \in \{0, 1\}$, $\mathcal{U}$ generates a NIZK proof $(\pi_{b_j,1}, \pi_{b_j,2})$ for the pairing-product equations $e(g^{b_j}, h) = e(g, h^{b_j})$ and $e(g^{b_j}, h^{b_j}) = e(g^{b_j}, h)$, which guarantee that $b_j^2 = b_j$, so that $b_j \in \{0, 1\}$.

   b. If $j < L - \ell$, generate a commitment $C_{Y_j} = \mathsf{GSCom}(Y_j, open_{Y_j})$ to the PRF value $Y_j = g^{1/(s+x_j)}$ as well as a NIZK proof $\pi_{Y_j}$ that it satisfies $e(Y_j, L_{S,1} \cdot X_j) = e(g, h)$, where $X_j = h^{x_j}$. This consists of a commitment $C_{\Phi_{Y_j}}$ to a variable $\Phi_{Y_j} \in \mathbb{G}_1$ and a proof that $e(\Phi_{Y_j}, L_{S,1} \cdot X_j) = e(g,h)$ and $e(Y_j/\Phi_{Y_j}, h^\theta) = 1_{\mathbb{G}_T}$. Then, pick $\delta_{j,1}, \delta_{j,2}, \delta_{j,3} \xleftarrow{R} \mathbb{Z}_p$ and compute

$$\begin{aligned} T_{j,1} &= h^{\delta_{j,1}}, & T_{j,2} &= e(Y_j, h)^{\delta_{j,1}} \\ T_{j,3} &= h^{\delta_{j,2}}, & T_{j,4} &= pk_{\mathcal{U}} \cdot e(Y_j, h)^{\delta_{j,2}}, \\ T_{j,5} &= h^{\delta_{j,3}}, & T_{j,6} &= e(g_0, h^{sk_{\mathcal{U}}}) \cdot e(Y_j, h)^{\delta_{j,3}}. \end{aligned}$$

Generate NIZK proofs $(\pi_{j,T_1}, \pi_{j,T_3}, \pi_{j,T_5})$ that $(Y_j, K_{\mathcal{U}})$ satisfy

$$\begin{aligned} T_{j,2} &= e(Y_j, T_{j,1}) \\ T_{j,4} &= e(g, K_{\mathcal{U}}) \cdot e(Y_j, T_{j,3}) \\ T_{j,6} &= e(g_0, K_{\mathcal{U}}) \cdot e(Y_j, T_{j,5}). \end{aligned} \tag{2}$$

These proofs require new commitments $\{C_{\Phi_{j,k}}\}_{k=1,3,5}$, $C_{\Phi'_{Y_j}}$ and $C_{\Phi''_{Y_j}}$ to auxiliary variables $\{\Phi_{j,k}\}_{k=1,3,5}$, $\Phi'_{Y_j}, \Phi''_{Y_j} \in \mathbb{G}_1$ respectively and proofs for relations

$$\begin{aligned} T_{j,2} &= e(Y_j, \Phi_{j,1}), & e(Y_j/\Phi_{Y'_j}, h^\theta) &= 1_{\mathbb{G}_T}, \\ T_{j,4} &= e(g, K_{\mathcal{U}}) \cdot e(\Phi_{Y'_j}, \Phi_{j,3}) & e(Y_j/\Phi_{Y''_j}, h^\theta) &= 1_{\mathbb{G}_T}, \\ T_{j,6} &= e(g_0, K_{\mathcal{U}}) \cdot e(\Phi_{Y''_j}, \Phi_{j,5}), & \{e(g^\theta, T_{j,k}/\Phi_{j,k}) &= 1_{\mathbb{G}_T}\}_{k\in\{1,3,5\}}. \end{aligned}$$

c. If $j = L - \ell$, compute the serial number $Y_{L-\ell} = g^{1/(s+x_{L-\ell})}$ and generate a NIZK proof $\pi_{Y_{L-\ell}}$ that $e(Y_{L-\ell}, L_{S,1} \cdot X_{L-\ell}) = e(g,h)$. This proof consists of a commitment $C_{\Phi_{Y_{L-\ell}}}$ to $\Phi_{Y_{L-\ell}} \in \mathbb{G}_1$ and proofs for equations

$$e(\Phi_{Y_{L-\ell}}, L_{S,1} \cdot X_{L-\ell}) = e(g,h), \qquad e(Y_{L-\ell}/\Phi_{Y_{L-\ell}}, h^\theta) = 1_{\mathbb{G}_T}.$$

Compute $Z_{L-\ell} = g^{sk_{\mathcal{U}}} \cdot g^{R/(t+x_{L-\ell})}$, where $R = H(\texttt{info}, pk_{\mathcal{M}}) \in \mathbb{Z}_p$, and a NIZK proof $\pi_{Z_{L-\ell}}$ that $Z_{L-\ell}$ is well-formed. This requires new Groth-Sahai commitments $C_{W_{L-\ell}}, C_{\Phi_{W_{L-\ell}}}$ to auxiliary variables $W_{L-\ell} = g^{1/(t+x_{L-\ell})}$, $\Phi_{W_{L-\ell}} = g^{1/(t+x_{L-\ell})}$ and a proof that:

$$\begin{aligned} e(g, K_{\mathcal{U}}) \cdot e(W_{L-\ell}, h^R) &= e(Z_{L-\ell}, h), \\ e(W_{L-\ell}, L_{T,1} \cdot X_{L-\ell}) &= e(g,h) \\ e(W_{L-\ell}/\Phi_{W_{L-\ell}}, h^\theta) &= 1_{\mathbb{G}_T}. \end{aligned}$$

Finally, update $\mathsf{state}$ into $\mathsf{state}' = \mathsf{state} \cup \{(x_{coin})\}$ and output the coin

$$\begin{aligned} coin = \Big( &\{C_{S,i}\}_{i=1}^3, \{C_{T,i}\}_{i=1}^3, \{C_{\mathcal{U},i}\}_{i=1}^3,\ C_{K_{\mathcal{U}}},\ \pi_{K_{\mathcal{U}}},\ \pi^{sig}, \\ &\{C_{X_j},\ C_{b_j},\ C'_{b_j},\ \pi_{x_j},\ \pi_{b_j,1}\ \pi_{b_j,2}\}_{j=1}^{L-\ell}, \\ &\{(T_{j,1}, T_{j,2}, T_{j,3}, T_{j,4}, T_{j,5}, T_{j,6}),\ C_{Y_j},\ C_{\Phi_{Y_j}},\ C'_{\Phi_{Y_j}},\ C''_{\Phi_{Y_j}}, \\ &\{C_{\Phi_{j,k}}\}_{k\in\{1,3,5\}},\ \pi_{Y_j},\ \pi_{j,T_1},\ \pi_{j,T_3},\ \pi_{j,T_5}\}_{j=0}^{L-\ell-1}, \\ &Y_{L-\ell},\ Z_{L-\ell},\ C_{\Phi_{Y_{L-\ell}}}, C_{W_{L-\ell}},\ C_{\Phi_{W_{L-\ell}}},\ \pi_{Y_{L-\ell}},\ \pi_{Z_{L-\ell}},\ \texttt{info} \Big) \end{aligned}$$

**VerifyCoin**$(\mathsf{params}, pk_{\mathcal{M}}, pk_{\mathcal{B}}, v = 2^{\ell}, coin)$**:** parse *coin* as above. Return 1 iff all proofs verify.

**Deposit**$(\mathsf{params}, pk_{\mathcal{B}}, pk_{\mathcal{M}}, coin, 2^{\ell}, \mathrm{DB}_{\mathcal{B}})$**:** parse *coin* as above and perform the same checks as VerifyCoin. Then, define $\mathrm{DB}'_{\mathcal{B}} = \mathrm{DB}_{\mathcal{B}} \cup \{(coin, \mathsf{flag}, 2^{\ell}, pk_{\mathcal{M}})\}$ where the value of flag depends on whether *coin* is a valid coin of value $2^{\ell}$ and whether a cheating attempt is detected.

- If *coin* does not properly verify, $\mathcal{B}$ sets $\mathsf{flag} =$ "$\mathcal{M}$" to indicate a cheating merchant.
- If *coin* properly verifies, the bank $\mathcal{B}$ runs the following test. For each entry $(coin_s, \mathsf{flag}_s, 2^{\ell_s}, pk_{\mathcal{M}_s}) \in \mathrm{DB}_{\mathcal{B}}$, where $s = 1$ to $|\mathrm{DB}_{\mathcal{B}}|$, $\mathcal{B}$ parses $coin_s$ as above. If $(\mathtt{info}_s, pk_{\mathcal{M}_s}) = (\mathtt{info}, pk_{\mathcal{M}})$, $\mathcal{B}$ sets $\mathsf{flag} =$ "$\mathcal{M}$". Otherwise, from $coin_s$, it extracts the path $\{(T_{s,j,1}, T_{s,j,2}, T_{s,j,3}, T_{s,j,4})\}_{j=0}^{L-\ell_s-1}$, the serial number $Y_{L-\ell_s} \in \mathbb{G}_1$ and the tag $Z_{L-\ell_s} \in \mathbb{G}_1$. It also parses *coin* to extract the path $\{(T_{j,1}, T_{j,2}, T_{j,3}, T_{j,4})\}_{j=0}^{L-\ell-1}$, the serial number $Y_{L-\ell} \in \mathbb{G}_1$ and the tag $Z_{L-\ell}$. If $\ell < \ell_s$ and $T_{L-\ell_s,2} = e(Y_{L-\ell_s}, T_{L-\ell_s,1})$, $\mathcal{B}$ sets $\mathsf{flag} =$ "$\mathcal{U}$", outputs $coin_s$ and *coin* and reports a double-spending. Likewise, if $\ell > \ell_s$ and $T_{s,L-\ell,2} = e(Y_{L-\ell}, T_{s,L-\ell,1})$, $\mathcal{B}$ also sets $\mathsf{flag} =$ "$\mathcal{U}$" and outputs $coin_s$ and *coin*. Finally, if $\ell = \ell_s$, $\mathcal{B}$ sets $\mathsf{flag} =$ "$\mathcal{U}$" if and only if $Y_{L-\ell} = Y_{L-\ell_s}$.
- If *coin* verifies and no double-spending is detected, $\mathcal{B}$ sets $\mathsf{flag} =$ "accept" and credits the account of $pk_{\mathcal{M}}$ by the amount of $2^{\ell}$.

After the above tests, the updated database $\mathrm{DB}'_{\mathcal{B}}$ supersedes $\mathrm{DB}_{\mathcal{B}}$.

**Identify**$(\mathsf{params}, pk_{\mathcal{B}}, coin_a, coin_b)$**:** on input of fraudulent coins $coin_a$ and $coin_b$, the bank $\mathcal{B}$ can identify the double-spender as follows.

1. Extract $\mathtt{info}_a$, $\{(T^{(a)}_{j,1}, T^{(a)}_{j,2}, T^{(a)}_{j,3}, T^{(a)}_{j,4})\}_{j=0}^{L-\ell_a-1}$, $(Y^{(a)}_{L-\ell_a}, Z^{(a)}_{L-\ell_a}) \in \mathbb{G}_1^2$ from $coin_a$. Also, parse $coin_b$ to retrieve $\mathtt{info}_b$, $\{(T^{(b)}_{j,1}, T^{(b)}_{j,2}, T^{(b)}_{j,3}, T^{(b)}_{j,4})\}_{j=0}^{L-\ell_b-1}$ and $(Y^{(b)}_{L-\ell_b}, Z^{(b)}_{L-\ell_b}) \in \mathbb{G}_1^2$.
2. If $\ell_b > \ell_a$, recover $pk_{\mathcal{U}}$ as $pk_{\mathcal{U}} = T^{(a)}_{L-\ell_b,4}/e(Y^{(b)}_{L-\ell_b}, T^{(a)}_{L-\ell_b,3})$. If $\ell_b < \ell_a$, $pk_{\mathcal{U}}$ can be obtained as $pk_{\mathcal{U}} = T^{(b)}_{L-\ell_a,4}/e(Y^{(a)}_{L-\ell_a}, T^{(b)}_{L-\ell_a,3})$. In the case $\ell_a = \ell_b$, we must have $Y_{\ell_a} = Y_{\ell_b}$. Then, $\mathcal{B}$ computes $R_a = H(\mathtt{info}_a, pk_{\mathcal{M}_a})$, $R_b = H(\mathtt{info}_b, pk_{\mathcal{M}_b})$ and then $\kappa = (Z^{(a)}_{L-\ell_a}/Z^{(b)}_{L-\ell_b})^{1/(R_a-R_b)}$, which allows recovering $g^{sk_{\mathcal{U}}} = Z^{(a)}_{L-\ell_a}/\kappa^{R_a}$ and thus $pk_{\mathcal{U}} = e(g^{sk_{\mathcal{U}}}, h)$.

The security of the scheme relies on the collision-resistance of $H$ and the intractability assumptions recalled in Section 2.3. More precisely, we state the following theorem for which a proof is given in the full version of the paper.

**Theorem 2.** *Assuming that $H$ is a collision-resistant hash function and that the SXDH, D3DH, TDH, D3DH, $q_w$-HSDH and the $2^{L+2}$-DDHI assumptions where $q_w$ denotes the number of $\mathcal{Q}_{\mathsf{withdraw}}$ queries all hold in $(\mathbb{G}_1, \mathbb{G}_2)$, our e-cash scheme provides anonymity, balance, identification and weak-exculpability.*

The most difficult part of the security proof is the proof of anonymity. More precisely, when it comes to build a simulator, we need to simulate NIZK proofs for pairing product equations of the form (2), which is non-trivial. Indeed, as noted in [31], this is only known to be possible when the target element of the equation (which lives in $\mathbb{G}_T$) can be written as a pairing of known elements of $\mathbb{G}_1$ and $\mathbb{G}_2$. The problem is that, in equations like (2), some pairing values have to be gradually replaced by uniformly random values of $\mathbb{G}_T$. To deal with this problem, we appeal to the D3DH assumption in a similar way to [33]. Namely, the D3DH input element $\varGamma$, which is either $g^{abc}$ or a random element of $\mathbb{G}_1$, is available as a "pre-image" of the target pairing value and makes it possible to simulate proofs for pairing product equations at the expense of introducing auxiliary variables.

## References

1. M. H. Au, W. Susilo, Y. Mu. Practical Anonymous Divisible E-Cash from Bounded Accumulators. In *Financial Cryptography 2008*, LNCS 5143, pp. 287–301 , 2008.
2. M. H. Au, Q. Wu, W. Susilo, Y. Mu. Compact E-Cash from Bounded Accumulator. In *CT-RSA'07*, LNCS 4377, pp. 178–195, 2007.
3. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In *TCC'08*, LNCS 4948, pages 356–374, 2008.
4. M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *Crypto'09*, LNCS 5677, pp. 108–125, 2009
5. M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. Compact E-Cash and Simulatable VRFs Revisited. In *Pairing'09*, LNCS 5671, pp. 114–131, 2009.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS'93*, pp. 62–73, 1993.
7. O. Blazy, S. Canard, G. Fuchsbauer, A. Gouget, H. Sibert, J. Traoré. Achieving Optimal Anonymity in Transferable E-Cash with a Judge. In *Africacrypt 2011*, LNCS 6737, pp. 206–223, 2011.
8. D. Boneh, C. Gentry, B. Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In *Crypto'05*, *LNCS* 3621, pp. 258–275, 2005.
9. X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In *PKC'07* , LNCS 4450, pp. 1–15, 2007.
10. J. Camenisch, S. Hohenberger, A. Lysyanskaya. Compact E-Cash. In *Eurocrypt'05*, LNCS 3494, pp. 302–321, 2005.
11. J. Camenisch, S. Hohenberger, A. Lysyanskaya. Balancing Accountability and Privacy Using E-Cash. In *SCN'06*, LNCS 4116, pp. 141–155, 2006.
12. J. Camenisch, M. Kohlweiss, C. Soriente. An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In *PKC'09*, *LNCS* 5443, pp. 481–500, 2009.
13. J. Camenisch, A. Lysyanskaya, M. Meyerovich. Endorsed E-Cash In *IEEE Security & Privacy'07*, pp. 101–115, 2007.
14. S. Canard, A. Gouget. Divisible E-Cash Systems Can Be Truly Anonymous. In *Eurocrypt'07*, LNCS 4515, pp. 482–497, 2007.
15. S. Canard, A. Gouget, J. Traoré. Improvement of Efficiency in (Unconditional) Anonymous Transferable E-Cash. In *Financial Cryptography 2008*, LNCS 5143, pp 202–214, 2008.

16. S. Canard, A. Gouget. Anonymity in Transferable E-cash. In *ACNS'08*, LNCS 5037, pp. 207–223, 2008.
17. S. Canard, A. Gouget, E. Hufschmitt. A Handy Multi-coupon System. In *ACNS'06*, LNCS 3989, pp. 66–81, 2006.
18. S. Canard, A. Gouget. Multiple Denominations in E-cash with Compact Transaction Data. In *Financial Cryptography 2010*, LNCS 6052, pp. 82–97, 2010.
19. R. Canetti, O. Goldreich, S. Halevi. The Random Oracle Methodology, Revisited. In *STOC'98*, pp. 209–218, ACM Press, 1998.
20. A.-H. Chan, Y. Frankel, Y. Tsiounis. Easy Come - Easy Go Divisible Cash. In *Eurocrypt'98*, *LNCS* 1403, pp. 561–575, 1998.
21. M. Chase, A. Lysyanskaya. Simulatable VRFs with Applications to Multi-theorem NIZK. In *Crypto'07*, *LNCS* 4622, pp. 303–322, 2007.
22. D. Chaum. Blind Signatures for Untraceable Payments. In *Crypto'82*, pp. 199–203, 1982.
23. D. Chaum. Blind Signature Systems. In *Crypto'83*, p. 153, 1983.
24. D. Chaum, A. Fiat, M. Naor. Untraceable Electronic Cash. In *Crypto'88*, *LNCS* 403, pp. 319–327, 1988.
25. D. Chaum, T. Pedersen. Transferred Cash Grows in Size. In *Eurocrypt'92*, *LNCS* 658, pp. 390–407, 1992.
26. S. D'Amiano, G. Di Crescenzo. Methodology for Digital Money based on General Cryptographic Tools. In *Eurocrypt'94*, *LNCS* 950, pp. 156–170, 1994.
27. Y. Dodis. Efficient Construction of (Distributed) Verifiable Random Functions. In *PKC'03*, *LNCS* 2567, pp. 1–17, 2003.
28. Y. Dodis, A. Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *PKC'05*, *LNCS* 3386, pp. 416–431, 2005.
29. M. K. Franklin, M. Yung. Secure and Efficient Off-Line Digital Money. In *ICALP'93*, *LNCS* 700, pp. 265–276, 1993.
30. G. Fuchsbauer, D. Pointcheval, D. Vergnaud. Transferable Constant-Size Fair E-Cash. In *CANS'09*, *LNCS* 5888, pp. 226–247, 2009.
31. J. Groth and A. Sahai. Efficient non-interactive proof systems for bilinear groups. In *Eurocrypt'08*, LNCS 4965, pp. 415–432, 2008.
32. S. Jarecki, V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In *Eurocrypt'07*, LNCS 4515, pp. 97–114, 2007.
33. B. Libert, D. Vergnaud. Group Signatures with Verifier-Local Revocation and Backward Unlinkability in the Standard Model. In *CANS'09*, *LNCS* 5888, pp. 498-517, 2009.
34. S. Micali, M.-O. Rabin, S. Vadhan. Verifiable Random Functions. In *FOCS'99*, pp. 120–130, 1999.
35. T. Nakanishi, Y. Sugiyama. Unlinkable Divisible Electronic Cash. In *ISW'00*, LNCS 1975, pp. 121–134, 2000.
36. T. Okamoto. K. Ohta. Universal Electronic Cash. In *Crypto'91*, LNCS 576, pp. 324–337, 1991.
37. T. Okamoto. An Efficient Divisible Electronic Cash Scheme. In *Crypto'95*, *LNCS* 963, pp. 438–451, 1991.
38. J.-C. Pailles. New Protocols for Electronic Money. In *Auscrypt'92*, *LNCS* 718, pp. 263–274, 1992.
39. T. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Crypto'91*, *LNCS* 576, pp. 129–140, 1991.
40. Y. Tsiounis. Efficient Electronic Cash: New Notions and Techniques. PhD Thesis, Northeaster University, Boston, 1997.