Multiple Denominations in E-cash with Compact Transaction Data[⋆]

Sébastien Canard[1] and Aline Gouget[2]

[1] Orange Labs R&D, 42 rue des Coutures, F-14066 Caen, France.
[2] Gemalto, 6, rue de la Verrerie, F-92190 Meudon, France.

Abstract. We present a new construction of *divisible* e-cash that makes use of 1) a new generation method of the binary tree of keys; 2) a new way of using bounded accumulators. The transaction data sent to the merchant has a constant number of bits while spending a monetary value 2^ℓ. Moreover, the spending protocol does not require complex zero-knowledge proofs of knowledge such as proofs about double discrete logarithms. We then propose the first strongly anonymous scheme with standard unforgeability requirement and realistic generation parameters while improving the efficiency of the spending phase.

1 Introduction

In e-cash systems, users withdraw coins from a bank and use them to pay merchants (preferably without involving the bank during this protocol). Finally, merchants deposit coins to the bank. An e-cash system must prevent both a user from double-spending, and a merchant from depositing twice a coin. The anonymity of honest users should be protected whereas the identity of cheaters must be recovered preferably without using a trusted third party.

Divisible e-cash aims at improving the efficiency of both the withdrawal protocol and the spending of multiple denominations. The underlying idea is to efficiently withdraw a single divisible coin equivalent to 2^L unitary coins. The user can spend this coin by dividing its monetary value, e.g. by sub-coins of monetary value 2^ℓ, $0 \leq \ell \leq L$. In this paper, we revisit the divisible e-cash approach by targeting the most demanding security model while providing a realistic parameter generation algorithm and an efficient spending protocol.

1.1 Related Work

A generic construction of divisible e-cash schemes which fulfill the classical properties of anonymity and strongly unlinkability without using a trusted third party to revoke the identity of cheaters has been proposed in [9]. The wallet is represented by a binary tree such that each internal node corresponds to an amount, i.e. 2^{L-i} if the node's distance to the root is i, $0 \leq i \leq L$. Each node in the

[⋆] This work has been financially supported by the French Agence Nationale de la Recherche and the TES Cluster under the PACE project.

Appeared in R. Sion (Ed.): Financial Cryptography 2010, preproceedings.

tree is related to a *key* such that the key of a child can be computed from the key of one of its ascendants. The main efficiency bottleneck of the practical instantiation given in [9] is that the user has to prove during the spending phase the correctness from the tree root to the target node without revealing none of the $L - \ell$ intermediate values. As one node key is derived from its parents using a modular exponentiation, the user must prove, for each intermediate value and using proofs about double discrete logarithms, that they satisfy a certain relation. Each such proof is expensive and requires $\Omega(k)$ group elements to be communicated in order to guarantee $1/2^k$ soundness error. Moreover, the construction of the binary tree used in [9] (and previously used in [11]) is difficult to instantiate in practice [1]. Indeed, this construction necessitates to manage $L + 2$ groups $G_0, G_1, \cdots, G_{L+1}$ with prime order $p_0, p_1, \cdots, p_{L+1}$ respectively, such that for all $1 \leq i \leq L{+}1$, G_{i+1} is a subgroup of $\mathbb{Z}_{p_i}^*$. One possibility is to take $p_i = 2 \times p_{i-1} + 1$ for all $1 \leq i \leq L + 1$. Using prime number theory, it is possible to show that the probability to generate such prime numbers is approximately 2^{-95} for 1024 bits prime numbers and $L = 10$, which is unpractical.

A very efficient variant of this scheme based on bounded accumulators has been proposed in [1]. Its main drawback is that it does not fulfills the classical security property of unforgeability. Indeed, it is possible for a malicious user to withdraw a divisible coin of monetary value $L2^L$ whereas the legitimate value is 2^L by cheating in the construction of the binary tree of keys during the withdrawal protocol. Next, the user can spend $L2^L$ coins without being detected and identified. The solution proposed by the authors is that the bank will use the cut-and-choose method during the withdrawal protocol by flipping a coin b and executing the withdrawal protocol correctly if $b = 1$ and asking the user to reveal her binary tree that is finally dropped if $b = 0$. If the revealed tree is correct, the user is honest and the withdrawal protocol is repeated again from the beginning. If the user is a cheater, a fine of value $2L2^L$ is deducted from the user's account. This drawback may be considered as unacceptable from the bank point of view even if the bank should not loose money "on average".

1.2 Our Contribution

We revisit the divisible e-cash approach by targeting both the most demanding security model and the effective possibility to instantiate an e-cash system from a theoretical method. We introduce a new construction based on algebraic objects to generate the binary tree without any previously mentioned problems (impracticability of the key generation [9] and unusual security model [1]). We introduce a new technique to prove the validity of the spending. We show that it is possible to prove that one node key is derived from its father, which is impossible in the proposal of [1]. This enables us to prove that one node key is derived from only its father and we do it only once instead of $(L-\ell)$ times in the scheme proposed in [9] for spending 2^ℓ coins from a divisible coin of 2^L coins. Next, we prove the remainder of the paths from the spent node to the leaves using a variant of the accumulator technique from [1]. In our construction, the spender only sends to the merchant a constant number of bits to spend 2^ℓ coins.

Appeared in R. Sion (Ed.): Financial Cryptography 2010, preproceedings.

2 Preliminaries

2.1 Construction of the binary tree of keys

In the following, any divisible coin of monetary value 2^L is assigned to a binary tree with $L+2$ levels, as done in [9]. The value of the tree root (at level 0) is 2^L. Any other internal node in the tree has a value corresponding to half of the amount of its parent node. The leaves (at level $L+1$) have no monetary value. For every level i, $0 \leq i \leq L+1$, the 2^i nodes are assigned keys denoted by $k_{i,j}$ with $0 \leq j \leq 2^i - 1$. The following rule must be satisfied in order to protect over-spending (c.f. Definition 3): *when a node n is used, none of descendant and ancestor nodes of n can be used, and no node can be used more than once.*

We now describe a new way to generate the binary tree of keys based on algebraic objects which can be efficiently generated. Let q, p, P be three primes such that p is of size l_p, q is of size l_q and divides $p-1$, and $P = 2p+1$. We denote by $\mathcal{G}_q$ (resp. $\mathcal{G}_p$) the subgroup of $\mathbb{Z}_p^*$ (resp. $\mathbb{Z}_P^*$) of order q (resp. p) and g_0, g_1 are two generators of $\mathcal{G}_q$. The keys of the tree are computed from the root to the leaves as follows. Given a key $k_{i,j}$ with $0 \leq i \leq L$ and $0 \leq j \leq 2^i - 1$ of an internal node, the two keys related to its two direct descendants are computed as follows: $k_{i+1,2j} = g_0^{k_{i,j} \pmod q} \pmod p$ and $k_{i+1,2j+1} = g_1^{k_{i,j} \pmod q} \pmod p$.

2.2 Discrete Log Relation Sets

Roughly speaking, a Zero Knowledge Proof of Knowledge (ZKPK) is an interactive protocol during which a prover proves to a verifier that he knows a set of secret values $\alpha_1, \ldots, \alpha_q$ that verify a given relation R, without revealing the secret values; we denote it by $\textsc{Pok}(\alpha_1, \ldots, \alpha_q : \mathsf{R}(\alpha_1, \ldots, \alpha_q))$. In the following, the secret values are discrete logarithms in relations constructed over a group either of prime or unknown order. These constructions should verify the soundness and zero-knowledge properties [8, 4]. The relation R can be a proof of knowledge of a discrete logarithm denoted by $\textsc{Pok}(\alpha : y = g^\alpha)$, a proof of knowledge of a representation, denoted by $\textsc{Pok}(\alpha_1, \ldots, \alpha_q : y = g_1^{\alpha_1} \ldots g_q^{\alpha_q})$, or a proof of equality of discrete logarithms, denoted by $\textsc{Pok}(\alpha : y = g^\alpha \wedge z = h^\alpha)$. Note that, contrary to [9], we do not use the complex proof of knowledge of double-discrete logarithms. We apply the Fiat-Shamir heuristic [10] to turn it into a signature on some message m: $\textsc{Sok}(\alpha_1, \ldots, \alpha_q : \mathsf{R}(\alpha_1, \ldots, \alpha_q))(m)$.

2.3 Signature Schemes with Additional Features

Camenisch and Lysyanskaya [7] have proposed various unforgeable signature schemes based on Pedersen's commitment scheme to which they add some specific protocols. There is first an efficient protocol between a user $\mathcal{U}$ and a signer $\mathcal{S}$ that permits $\mathcal{U}$ to obtain from $\mathcal{S}$ a signature σ of some commitment C on values $(x_1, \ldots, x_\ell)$ unknown from $\mathcal{S}$. $\mathcal{S}$ computes $\mathrm{CL}\textsc{Sign}(C)$ and $\mathcal{U}$ obtains $\sigma = \textsc{Sign}(x_1, \ldots, x_\ell)$ and second, an efficient proof of knowledge of a signature of some committed values.

The Extended Special Signature (ESS+) scheme introduced in [2,1] is a variant of the Camenisch-Lysyanskaya (CL) signature scheme that allows to sign a block of messages, one of them being an element in a cyclic group. The user obtains a signature $\sigma = \textsc{Sign}(X, x_1, \ldots, x_\ell)$ where X is an element of the multiplicative group while the x_i's are exponents. In our divisible e-cash system, we may use the signature scheme described in [1] together with the following zero-knowledge proof of knowledge: $\textsc{Pok}(X, x_1, \cdots, x_\ell, \sigma : \sigma = \textsc{Sign}(X, x_1, \cdots, x_\ell))$, which is unforgeable under the AWSM assumption [2]. Note that any signature scheme with the same features can also be used.

2.4 Bounded Accumulators

An accumulator scheme Acc is a method which permits to accumulate a large set of objects in a single short value. It next provides evidence that a given object is contained in the accumulator by producing a related witness. We denote by $x \in \mathsf{Acc}$ or $(x, w) \in \mathsf{Acc}$ that the value x is accumulated in Acc, possibly with the witness w. It is possible to prove (in a zero-knowledge manner) that one (secret) value is truly accumulated in a given accumulator such that the computation and the verification of the proof do not depend on the number of accumulated values. The main accumulator schemes are described in [6,12,5]. As noticed in [2], the proposal given in [12] (and this is also the case for [5]) is bounded in the sense that it should not be possible to accumulate more than a given number s of objects, this number being stated at the key generation process. In our scheme, the payer needs to prove that she knows a secret accumulator Acc certified by some authorities in which several revealed values $x_1, \cdots, x_\ell$ are accumulated, that is $\textsc{Pok}(w_1, \cdots, w_\ell, \mathsf{Acc}, \sigma : (x_1, w_1) \in \mathsf{Acc} \wedge \cdots \wedge (x_\ell, w_\ell) \in \mathsf{Acc} \wedge \sigma = \textsc{Sign}(\mathsf{Acc}))$, also denoted $\textsc{Pok}(\mathsf{Acc}, \sigma : (x_1, \cdots, x_\ell) \in \mathsf{Acc} \wedge \sigma = \textsc{Sign}(\mathsf{Acc}))$. This new feature obviously not introduce any new flaw in the accumulator scheme. In Appendix A, we describe a construction such that this proof does not depend on the number ℓ of revealed values.

3 Model for Divisible E-cash

3.1 Procedures for Divisible E-cash

Three types of actors are involved in a divisible e-cash system: the bank $\mathcal{B}$, the user $\mathcal{U}$ and the merchant $\mathcal{M}$. We denote by λ the security parameter. The monetary value of a divisible coin is fixed to 2^L. A divisible e-cash system $\mathcal{S}$ can be defined by the following polynomial-time procedures:

- $\textsc{Setup}(1^\lambda)$ is a probabilistic algorithm which outputs the parameters of the system param. In the following, param and 1^λ are implicitly in the input of all algorithms and protocols;
- $\textsc{BKeyGen}()$ is a probabilistic algorithm which outputs $(\mathsf{bsk}, \mathsf{bpk})$ as the secret and public keys of the bank, respectively. A database cdb of all spent coins is initialized to the empty set ϵ;

- UKeyGen() is a probabilistic algorithm which outputs $(\mathsf{usk}, \mathsf{upk})$ as the secret and public keys of the user, respectively. Note that the same algorithm is executed by the merchants to get $(\mathsf{msk}, \mathsf{mpk})$;
- Withdraw$[\mathcal{B}(\mathsf{bsk}) \longleftrightarrow \mathcal{U}(\mathsf{usk}, \mathsf{bpk})]$ is a protocol which permits $\mathcal{U}$ to withdraw a divisible coin co, while the bank outputs its view viewW;
- Spend$[\mathcal{U}(\mathsf{usk}, \mathsf{co}, \mathsf{bpk}, \ell) \longleftrightarrow \mathcal{M}(\mathsf{msk}, \mathsf{bpk})]$ is a protocol which permits $\mathcal{U}$ to spend a value 2^ℓ from the divisible coin co to the merchant $\mathcal{M}$. The user outputs a new state for co and $\mathcal{M}$ outputs the received coin rco;
- Deposit$[\mathcal{M}(\mathsf{msk}, \mathsf{rco}, \mathsf{bpk}) \longleftrightarrow \mathcal{B}(\mathsf{bsk}, \mathsf{mpk}, \mathsf{cdb})]$ is a protocol which permits $\mathcal{M}$ to deposit a coin rco to the bank. The bank outputs either 1 and the monetary value 2^ℓ, or executes the algorithm Identify if the database cdb already contains the serial number in rco. This procedure is sometimes written Deposit(rco) for simplicity.
- Identify$(\mathsf{rco}, \mathsf{cdb})$ (or Identify(rco)) for short) is an algorithm which outputs the public key upk of a fraudulent player (either a user or a merchant) together with a proof π_G;
- VerifyGuilt$(\mathsf{rco}, \mathsf{cdb}, \mathsf{upk}, \pi_G)$ is an algorithm which outputs 1 if π_G is a valid proof that the player's public key upk has made a fraud during the spending of the coin rco, and 0 otherwise.

In the following, it is assumed that if an honest user runs a Withdraw protocol with an honest bank, then neither will output an error message. If an honest user runs a Spend protocol with an honest merchant, then the merchant always accept the coin.

3.2 Security Properties

The adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ in order to break a security property. The adversary $\mathcal{A}$ has access to the procedures of the system and to the parameters param. In addition, two oracles are defined in order to add and corrupt users: AddU() and CorruptU(j), where j is related to the user public key usk_j. In the following, the execution of $\mathcal{A}$ with access to the oracle Xxxx and with input e is denoted by $\mathcal{A}^{\mathrm{Xxxx}}(\mathsf{e})$.

Unforgeability. It guarantees that no coalition of players can deposit more coins than they have withdrawn from the bank.

```
Experiment $Exp^{\mathsf{unforge}}_{\mathcal{S},\mathcal{A}}(\lambda)$:

  – $(\mathsf{param}) \longleftarrow$ Setup$(\lambda)$,
  – $(\mathsf{bsk}, \mathsf{bpk}) \longleftarrow$ BKeyGen(), $(\mathsf{mpk}) \longleftarrow \mathcal{A}()$
  – $\mathsf{dc} \longleftarrow 0$, $\mathsf{sp} \longleftarrow 0$, cont $\longleftarrow$ true,
  – while (cont == true),
        $b \longleftarrow$ Withdraw$[\mathcal{C}(\mathsf{bsk}) \longleftrightarrow \mathcal{A}(\mathsf{bpk})]$
        if $(b == 1)$, then $\mathsf{dc} \longleftarrow \mathsf{dc} + 1$
        $(\mathsf{rco}, \ell) \longleftarrow \mathcal{A}(\mathsf{bpk})$
        if (Deposit$(\mathsf{rco})$ == 1), then $\mathsf{sp} \longleftarrow \mathsf{sp} + 2^\ell$
        cont $\longleftarrow \mathcal{A}()$
  – if $2^L \cdot \mathsf{dc} < \mathsf{sp}$ return 1
  – return 0
```

The success probability of $\mathcal{A}$ is defined by $Succ^{\mathsf{unforge}}_{\mathcal{S},\mathcal{A}}(\lambda) = \Pr\left[Exp^{\mathsf{unforge}}_{\mathcal{S},\mathcal{A}}(\lambda) = 1\right]$.

Definition 1 (Unforgeability). *A system $\mathcal{S}$ is* unforgeable *if for any polynomial-time adversary $\mathcal{A}$, the success probability $Succ^{\mathsf{unforge}}_{\mathcal{S},\mathcal{A}}(\cdot)$ is negligible.*

Anonymity. It guarantees that the bank, even helped by malicious users, cannot learn anything about a spending other than what is available from side information from the environment. In the following experiment, b is a bit.

```
Experiment $Exp^{\mathsf{anon}-b}_{\mathcal{S},\mathcal{A}}(\lambda)$:

  – $(\mathsf{param}) \longleftarrow \textsc{Setup}(\lambda)$, $(\mathsf{bpk}) \longleftarrow \mathcal{A}()$
  – $(\mathsf{upk}_0, \mathsf{upk}_1) \longleftarrow \mathcal{A}^{\textsc{Withdraw},\textsc{Spend},\textsc{AddU},\textsc{CorruptU}}()$
  – $\mathsf{rco} \longleftarrow \textsc{Spend}[\mathcal{C}(\mathsf{usk}_b) \longleftrightarrow \mathcal{A}()]$
  – return $b' \longleftarrow \mathcal{A}^{\textsc{Withdraw},\textsc{Spend},\textsc{AddU},\textsc{CorruptU}}(\mathsf{rco})$
```

The advantage of $\mathcal{A}$ for the anonymity experiment is defined by:

$$Adv^{\mathsf{anon}}_{\mathcal{S},\mathcal{A}}(\lambda) = \Pr\left[Exp^{\mathsf{anon}-1}_{\mathcal{S},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[Exp^{\mathsf{anon}-0}_{\mathcal{S},\mathcal{A}}(\lambda) = 1\right].$$

Definition 2 (Anonymity). *A system $\mathcal{S}$ is* anonymous *if for any polynomial-time adversary $\mathcal{A}$, the adversary advantage $Adv^{\mathsf{anon}}_{\mathcal{S},\mathcal{A}}(\cdot)$ is negligible.*

Identification of double-spenders. From the bank's point of view, no collection of users should be able to double-spend a coin without revealing one of their identities.

```
Experiment $Exp^{\mathsf{idds}}_{\mathcal{S},\mathcal{A}}(\lambda)$:

  – $(\mathsf{param}) \longleftarrow \textsc{Setup}(\lambda)$, $(\mathsf{bsk}, \mathsf{bpk}) \longleftarrow \textsc{BKeyGen}()$,
  – $\mathsf{rco} \longleftarrow \mathcal{A}^{\textsc{Withdraw},\textsc{Spend},\textsc{AddU},\textsc{CorruptU}}(\mathsf{bpk})$
  – if $(\textsc{Deposit}(\mathsf{rco}) == 0 \land \textsc{VerifGuilt}(\mathsf{rco}, \textsc{Identify}(\mathsf{rco})) == 0)$ return 1
  – return 0
```

The success probability of $\mathcal{A}$ is defined by $Succ^{\mathsf{idds}}_{\mathcal{S},\mathcal{A}}(\lambda) = \Pr\left[Exp^{\mathsf{idds}}_{\mathcal{S},\mathcal{A}}(\lambda) = 1\right]$.

Definition 3 (Identification of double spender). *A system $\mathcal{S}$* identifies double-spender *if for any polynomial-time adversary $\mathcal{A}$, the success probability $Succ^{\mathsf{idds}}_{\mathcal{S},\mathcal{A}}(\cdot)$ is negligible.*

Exculpability. It guarantees that the bank, even cooperating with malicious users, cannot falsely accuse honest users from having double-spent a coin. In the experiment, $\mathcal{CU}$ is the set of corrupted users.

```
Experiment $Exp^{\mathsf{exculp}}_{\mathcal{S},\mathcal{A}}(\lambda)$:

  – $(\mathsf{param}) \longleftarrow \textsc{Setup}(\lambda)$, $(\mathsf{bpk}) \longleftarrow \mathcal{A}()$
  – cont $\longleftarrow$ true, $\mathcal{CU} \longleftarrow \emptyset$
  – while (cont == true),
        $(j, \text{cont}) \longleftarrow \mathcal{A}^{\textsc{Withdraw},\textsc{Spend},\textsc{AddU},\textsc{CorruptU}}(\mathcal{CU})$, $\mathcal{CU} \longleftarrow \mathcal{CU} \cup \{\mathsf{upk}_j\}$
  – $\mathsf{rco} \longleftarrow \mathcal{A}^{\textsc{Withdraw},\textsc{Spend},\textsc{AddU},\textsc{CorruptU}}()$
  – if $(\textsc{Identify}(\mathsf{rco}) = (\mathsf{upk}, \pi_G) \land \textsc{VerifyGuilt}(\mathsf{rco}, \mathsf{upk}, \pi_G) = 1 \land \mathsf{upk} \notin \mathcal{CU})$ return 1
  – return 0
```

The success probability of $\mathcal{A}$ is defined by $Succ^{\mathsf{exculp}}_{\mathcal{S},\mathcal{A}}(\lambda) = \Pr\left[Exp^{\mathsf{exculp}}_{\mathcal{S},\mathcal{A}}(\lambda) = 1\right]$.

Definition 4 (Exculpability). *A system $\mathcal{S}$ is* exculpable *if for any polynomial-time adversary $\mathcal{A}$, the success probability $Succ^{\mathsf{exculp}}_{\mathcal{S},\mathcal{A}}(\cdot)$ is negligible.*

4 Description of Our Divisible E-cash Construction

4.1 Setup and Key Generation Procedures

A divisible coin has a value set to 2^L, where L is a positive integer. As in [9], a divisible coin in our system is represented by a binary tree of $L+2$ levels and the leaves have no value.

Let λ be the security parameter. Let q, p, P be three primes such that q divides $p-1$ and $P = 2p+1$. The size of p (resp. q) is denoted by l_p (resp. l_q). We denote by $\mathcal{G}_q$ (resp. $\mathcal{G}_p$) the subgroup of $\mathbb{Z}_p^*$ (resp. $\mathbb{Z}_P^*$) or order q (resp. p); g_0 and g_1 (resp. G and H) are two generators of $\mathcal{G}_q$ (resp. $\mathcal{G}_p$). Finally, let $\mathcal{H} : \{0,1\}^* \longrightarrow \mathbb{Z}_p^*$ be a collision-resistant hash function.

The parameters of $L+2$ bounded accumulators $\mathsf{Acc}, \mathsf{Acc}_1, \cdots, \mathsf{Acc}_{L+1}$ on the cyclic group $\mathcal{G}_p$ are generated during the Setup procedure (see Appendix A). The accumulator Acc is bounded to $2^{L+2}-2$ in order to accumulate all the keys of nodes in the tree from level 1 to level $L+1$ (and thus the key of the root is not accumulated in Acc). The accumulator Acc_i is bounded to 2^i in order to accumulate all the keys of nodes at level i, with $i \in [1, L+1]$. Then, it is not possible to accumulate more than 2^i keys of value 2^{L-i} (see Figure 1).

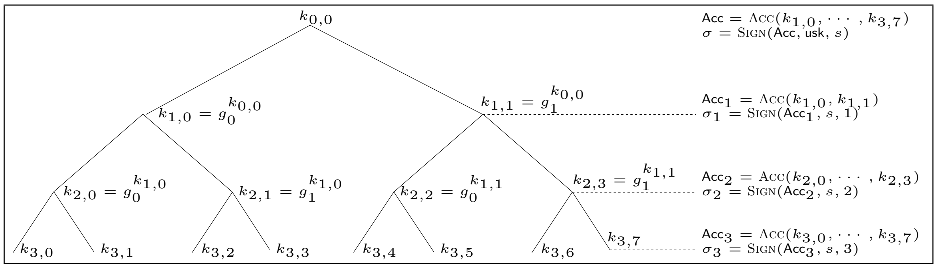
$k_{0,0}$
$k_{1,0} = g_0^{k_{0,0}}$
$k_{1,1} = g_1^{k_{0,0}}$
$k_{2,0} = g_0^{k_{1,0}}$
$k_{2,1} = g_1^{k_{1,0}}$
$k_{2,2} = g_0^{k_{1,1}}$
$k_{2,3} = g_1^{k_{1,1}}$
$k_{3,0}$
$k_{3,1}$
$k_{3,2}$
$k_{3,3}$
$k_{3,4}$
$k_{3,5}$
$k_{3,6}$
$k_{3,7}$
$\mathsf{Acc} = \textsc{Acc}(k_{1,0}, \cdots, k_{3,7})$
$\sigma = \textsc{Sign}(\mathsf{Acc}, \mathsf{usk}, s)$
$\mathsf{Acc}_1 = \textsc{Acc}(k_{1,0}, k_{1,1})$
$\sigma_1 = \textsc{Sign}(\mathsf{Acc}_1, s, 1)$
$\mathsf{Acc}_2 = \textsc{Acc}(k_{2,0}, \cdots, k_{2,3})$
$\sigma_2 = \textsc{Sign}(\mathsf{Acc}_2, s, 2)$
$\mathsf{Acc}_3 = \textsc{Acc}(k_{3,0}, \cdots, k_{3,7})$
$\sigma_3 = \textsc{Sign}(\mathsf{Acc}_3, s, 3)$

Fig. 1. Our new binary tree for a coin of monetary value 2^2 with $L = 2$

The algorithm BKeyGen is performed by the bank in order to generate a key pair $(\mathsf{bsk}, \mathsf{bpk})$ for the signature scheme ESS+ [1] on the group $\mathcal{G}_p$. The algorithm UKeyGen is executed by any user and merchant of the system. It consists in randomly choosing a secret $\mathsf{usk} \in \mathbb{Z}_p^*$ (resp. $\mathsf{msk} \in \mathbb{Z}_p^*$) and computing $\mathsf{upk} = G^{\mathsf{usk}}$ (resp. $\mathsf{mpk} = G^{\mathsf{msk}}$). Moreover, any user public key is assumed to be certified by an authority and the bank can be convinced that it belongs to a known identified user.

4.2 The Withdrawal Protocol

The withdrawal phase is a protocol between the user $\mathcal{U}$ (on input usk and bpk) and the bank $\mathcal{B}$ (on input bsk), which permits $\mathcal{U}$ to withdraw a coin of value

2^L. In a nutshell, $\mathcal{U}$ computes the keys $k_{0,0}, \cdots, k_{L+1,2^{L+1}-1}$ of the binary tree and next the accumulators $\mathsf{Acc}, \mathsf{Acc}_1, \cdots, \mathsf{Acc}_{L+1}$. Next, $\mathcal{B}$ produces $L+2$ ESS+ signatures on the messages $(\mathsf{Acc}, \mathsf{usk}, s)$, $(\mathsf{Acc}_1, s, 1)$, $\cdots$, $(\mathsf{Acc}_{L+1}, s, L+1)$. These signatures give a proof of the interaction between the user knowing usk and the bank. The secret value s is used to link together the $L+2$ signatures of a given withdrawal protocol knowing that the user and bank contribute randomness to the value s.

An important remark is that it is not necessary for the bank to check if the tree of keys is well-formed or if the accumulators are well-formed since they are bounded using appropriate values. As explained in the following, if the user cheats in the construction of the tree of keys or in the construction of the accumulated values, he won't be able to correctly execute the spending protocol, as the merchant will be able to make all validity checks. More formally, the protocol works as follows:

- $\mathcal{U}$ chooses at random the key root $k_{0,0} \in \mathbb{Z}_p^*$ and computes the keys of the full tree: given a key node $k_{i,j}$ with $0 \leq i \leq L$ and $0 \leq j \leq 2^i - 1$, the keys related to its two direct descendants are $k_{i+1,2j} = g_0^{k_{i,j} \pmod q} \pmod p$ and $k_{i+1,2j+1} = g_1^{k_{i,j} \pmod q} \pmod p$. The keys are stored in a table tr;
- $\mathcal{U}$ accumulates the keys $k_{i,j}$, for all $1 \leq i \leq L+1$ and $0 \leq j \leq 2^i - 1$, in Acc and sends it to $\mathcal{B}$. Next $\mathcal{U}$ and $\mathcal{B}$ interact using the ESS+ interactive protocol in order to get a signature σ on $(\mathsf{Acc}, \mathsf{usk}, s)$, where $s \in \mathbb{Z}_p^*$ is a secret value only known by the user and jointly generated by both $\mathcal{U}$ and $\mathcal{B}$. For this purpose, the user commits to the values usk and s' and the bank modifies s' to $s = s' + s''$ (without learning any information about s'), produces the commitment to Acc, usk and s and signs this commitment. Note that $\mathcal{B}$ can verify that usk is related to a known public key upk (*i.e.* by making $\mathcal{U}$ produces a proof of knowledge of usk such that $\mathsf{upk} = G^{\mathsf{usk}}$).
- for every i such that $1 \leq i \leq L+1$, $\mathcal{U}$ accumulates in Acc_i the keys $k_{i,j}$, with $j \in [0; 2^i - 1]$. Next, $\mathcal{U}$ sends $\mathsf{Acc}_1, \cdots, \mathsf{Acc}_{L+1}$ to $\mathcal{B}$ and interacts with $\mathcal{B}$ using the ESS+ interactive protocol in order to get the signatures $\sigma, \sigma_1, \cdots, \sigma_{L+1}$ on $(\mathsf{Acc}_1, s, 1)$, $\cdots$, $(\mathsf{Acc}_{L+1}, s, L+1)$, respectively. For this purpose, the user commits s' and proves that this is the same as in the previous step. The bank verifies the proof, again modifies s' to $s = s' + s''$, produces the commitment on Acc_i, s and i and signs it. Note that one single commitment to s' by the user is enough to obtain all the signatures.

At the end of this protocol, $\mathcal{U}$ outputs a coin $\mathsf{co} = \{\mathsf{tr}, \mathsf{Acc}, \mathsf{Acc}_1, \cdots, \mathsf{Acc}_{L+1}, \sigma, \sigma_1, \cdots, \sigma_{L+1}, \mathsf{spc}\}$, where $\mathsf{spc} \leftarrow \epsilon$ will contain information on spent nodes. Note that tr can be either erased or kept to avoid the re-computation of the key nodes during the spending phase.

4.3 The Spending Phase

We suppose that a user $\mathcal{U}$, with keys $(\mathsf{usk}, \mathsf{upk})$ and with a coin $\mathsf{co} = \{\mathsf{tr}, \mathsf{Acc}, \mathsf{Acc}_1, \cdots, \mathsf{Acc}_{L+1}, \sigma, \sigma_1, \cdots, \sigma_{L+1}, \mathsf{spc}\}$ wants to spend a value $2^\ell \leq 2^L$ to a merchant $\mathcal{M}$ (with input msk and bpk). Informally, $\mathcal{U}$ chooses an unspent node j_0 in the tree at level $L - \ell$ and computes 1) a serial number S as the concatenation of the keys $k_{L-\ell+1,2j_0}$ and $k_{L-\ell+1,2j_0+1}$ of the two descendant nodes and 2) the security tag using the key $k_{L-\ell,j_0}$. In addition, $\mathcal{U}$ produces a proof of validity of the accumulators and the ESS+ signatures coming from the withdrawal protocol. More formally, the protocol works as follow:

- $\mathcal{U}$ receives $\mathcal{M}$'s public key mpk and the proof π that $\mathcal{M}$ knows msk. Next, $\mathcal{U}$ and $\mathcal{M}$ can compute $R = \mathcal{H}(\mathsf{mpk}\|\mathit{info})$ where info is a pre-determined public information including the monetary value 2^ℓ and e.g. current time;
- $\mathcal{U}$ chooses in the binary tree an unspent node j_0 at level $L - \ell$ and finds in tr (or recompute) the corresponding key $k_{L-\ell,j_0}$ and its two direct descendants $k_{L-\ell+1,2j_0}$ and $k_{L-\ell+1,2j_0+1}$;
- the serial number is then formed as $S = k_{L-\ell+1,2j_0}\|k_{L-\ell+1,2j_0+1}$ and the security tag of this spending is $T = \mathsf{upk} \cdot H^{R \cdot k_{L-\ell,j_0}}$;
- finally $\mathcal{U}$ produces the signature of knowledge:

$$\begin{aligned}
&\mathit{\Pi} = \textsc{Sok}\big(\mathsf{usk}, k_{L-\ell,j_0}, s, \mathsf{Acc}, \sigma, \mathsf{Acc}_{L-\ell+1}, \sigma_{L-\ell+1} :\\
&T = G^{\mathsf{usk}} \cdot (H^R)^{k_{L-\ell,j_0}} \wedge k_{L-\ell+1,2j_0} = g_0^{k_{L-\ell,j_0}} \wedge k_{L-\ell+1,2j_0+1} = g_1^{k_{L-\ell,j_0}} \wedge\\
&(k_{L-\ell+1,2j_0}, k_{L-\ell+1,2j_0+1}, \ldots, k_{L+1,2^{\ell+1}j_0}, \cdots, k_{L+1,2^{\ell+1}(j_0+1)-1}) \in \mathsf{Acc} \wedge\\
&(k_{L-\ell+1,2j_0}, k_{L-\ell+1,2j_0+1}) \in \mathsf{Acc}_{L-\ell+1} \wedge \sigma = \textsc{Sign}(\mathsf{Acc}, u, s)\\
&\sigma_{L-\ell+1} = \textsc{Sign}(\mathsf{Acc}_{L-\ell+1}, s, L - \ell + 1)\big)(\mathsf{mpk}\|\mathit{info}\|R\|S\|T)
\end{aligned}$$

The spent coin is $\mathsf{rco} = \{\ell, S, T, \mathit{\Pi}, R\}$. Its validity is checked by $\mathcal{M}$ by computing all the descendant keys of S before performing the verification of $\mathit{\Pi}$ (see below). Moreover, the merchant, using the parameters used in the proof of knowledge $\mathit{\Pi}$, can check that the accumulator $\mathsf{Acc}_{L-\ell+1}$ has been signed with the value $L-\ell+1$, that the used parameters correspond to the ones of the right bounded accumulator. The divisible coin of $\mathcal{U}$ is updated as $\mathsf{co} = \{\mathsf{tr}, \mathsf{Acc}, \mathsf{Acc}_1, \cdots, \mathsf{Acc}_{L+1}, \sigma, \sigma_1, \cdots, \sigma_{L+1}, \mathsf{spc} = \mathsf{spc} \cup \{(L-\ell, j_0)\}\}$.

The proof $\mathit{\Pi}$ is done non-interactively by using usual zero-knowledge proofs of knowledge (see Appendix B) and the Fiat-Shamir heuristic [10], in the random oracle model, using $m = \mathsf{mpk}\|\mathit{info}\|R\|S\|T$ as a message. It proves that

- the security tag T is correctly computed from usk, R and $k_{L-\ell,j_0}$;
- the serial number S is correctly computed from $k_{L-\ell,j_0}$;
- all the descendants of the node $k_{L-\ell,j_0}$ are accumulated in Acc;
- $k_{L-\ell+1,2j_0}$ and $k_{L-\ell+1,2j_0+1}$ are accumulated in $\mathsf{Acc}_{L-\ell+1}$;
- the values Acc, usk and s (resp. $\mathsf{Acc}_{L-\ell+1}$, s and $L - \ell + 1$) are signed by the bank in σ (resp. $\sigma_{L-\ell+1}$).

Note that the spender only needs to prove that $k_{L-\ell+1,2j_0}$ and $k_{L-\ell+1,2j_0+1}$ are correctly derived from $k_{L-\ell,j_0}$. This is not necessary for the other descendant nodes. In fact, the receiver can easily compute all the descendant of $k_{L-\ell+1,2j_0}$ and $k_{L-\ell+1,2j_0+1}$. As they are all accumulated into Acc and both $k_{L-\ell+1,2j_0}$ and $k_{L-\ell+1,2j_0+1}$ are accumulated in $\mathsf{Acc}_{L-\ell+1}$, it is enough to prove that the spent coin is correct.

As shown in Appendix B, this proof is done in constant time. It does not depend on the monetary value 2^ℓ which is spent, except when the spender needs to develop a polynomial of degree 2^ℓ, which is quite immediate in practice. Moreover, as the merchant can compute all the keys from the ones used in the serial number to the ones of the leaves, the user does not need to send them to the merchant. Thus, the transaction data sent to the merchant has a constant number of bits while spending a monetary value of 2^ℓ.

4.4 Deposit and Detection of Frauds

The deposit of a coin $\mathsf{rco} = \{\ell, S, T, \Pi, R\}$ with value ℓ is done by $\mathcal{M}$ which sends it to $\mathcal{B}$ with a signature of the deposit request. First $\mathcal{B}$ verifies the correctness of Π. If it is correct, $\mathcal{B}$ computes the keys related to the $2^{\ell+1}$ leaves of $S = k_{L-\ell+1,2j_0} \| k_{L-\ell+1,2j_0+1}$ at level L in the tree. If at least one of these keys is already in its database cdb, $\mathcal{B}$ executes the procedure of double-spender identification. Else, $\mathcal{B}$ adds the $2^{\ell+1}$ leaf keys in cdb and outputs 1. $\mathcal{B}$ could store only 2^ℓ keys, i.e. it will always store leaves that are "right" (or left) child.

In case of a double-spending detection, the bank $\mathcal{B}$, given two coins $\mathsf{rco}_1 = \{\ell_1,\ S_1,\ T_1,\ \Pi_1\ ,R_1\}$ and $\mathsf{rco}_2 = \{\ell_2,\ S_2,\ T_2,\ \Pi_2,\ R_2\}$, tests if $R_1 = R_2$ which means that $\mathcal{M}$ is a cheater since the hash function $\mathcal{H}$ is collision-resistant. The proof of the cheat consists in publishing both deposits (including two signatures of $\mathcal{M}$ on the deposit request for the same coin). Else $\mathcal{B}$ will identify a user public key using the same technique as described in [9]. We distinguish two cases:

1. if $\ell_1 = \ell_2 = \ell$, then the same node key $k_{L-\ell,j_0}$ has been used in both T_1 and T_2. Thus, $\mathcal{B}$ computes $\mathsf{upk} = \left(T_1^{R_2}/T_2^{R_1}\right)^{\frac{1}{R_2-R_1}}$;
2. if $\ell_1 \neq \ell_2$ (*e.g.* $\ell_1 < \ell_2$), then from S_2, $\mathcal{B}$ can compute $k_{L-\ell_1,j_0}$ such that $T_1 = \mathsf{upk} \cdot H^{R_1 \cdot k_{L-\ell_1,j_0}}$ and thus retrieve $\mathsf{upk} = T_1/H^{R_1 \cdot k_{L-\ell_1,j_0}}$;

Finally, $\mathcal{B}$ outputs the proof π_G based on the two entries in the database cdb.

4.5 Efficiency Considerations

We compare the efficiency of the strongly anonymous divisible e-cash schemes of the state-of-the-art [9, 1] with our new proposal. We give in Table 1 the computation cost of the binary tree, the time complexity of the withdrawal and spending phases and the size of the divisible coin, where Exp is a modular exponentiation, and Dev(i) is the time needed to develop a polynomial of degree i. We differentiate in our comparison both types of secure divisible e-cash systems depending on the security model, i.e. classical model with truly unforgeability [9] and unusual model with a statistical balance assumption [1]. We do not include the complexity of the deposit phase and the size of the database which are similar in the three schemes.

Based on Table 1, we can conclude that our new proposal is significantly more efficient that the one of Canard-Gouget [9], regarding the spending phase, with the same security level. Our new proposal is little bit less efficient than the Au *et al.* one [1] but with a better security result. We consequently obtain the best trade off between previous approaches, considering efficiency and security.

Divisible e-cash scheme	Au *et al.* [1]	Canard-Gouget [9]	this paper
Model choice	Statistical balance	Unforgeability	
Binary tree computation	$(2^{L+1}-2)\textsc{Exp}$	$(2^{L+2}-2)\textsc{Exp}$ (not necessarily computed)	$(2^{L+2}-2)\textsc{Exp}$
Divisible coin storage size	$(2^{L+1}-2)\lvert p\rvert + (L+1)\textsc{Sign}$ $+(L+1)\textsc{Acc}$	$2\lvert p\rvert + (1)\textsc{Sign}$	$(2^{L+2}-2)\lvert p\rvert + (L+2)\textsc{Sign}$ $+(L+2)\textsc{Acc}$
Computational complexity of withdraw	$(L+1)\textsc{Sign}$ $+(L+1)\textsc{Acc}$	$(1)\textsc{Sign}$	$(L+2)\textsc{Sign}$ $+(L+2)\textsc{Acc}$
Computational complexity of spending 2^ℓ	$\textsc{Exp} + \textsc{Sok}\big(\textsc{Sign}$ $+\textsc{Acc} + 2\textsc{Exp}\big)$	$(i+3)\textsc{Exp} + \textsc{Sok}\big(\textsc{Sign}$ $+\mathcal{O}((L-\ell)t)\textsc{Exp}\big)$	$\textsc{Exp} + \textsc{Dev}(2^\ell)+$ $\textsc{Sok}\big(2\textsc{Sign} + 2\textsc{Acc} + 3\textsc{Exp}\big)$
Spending transfer size	$2\lvert p\rvert + \textsc{Sok}\big(\textsc{Sign}$ $+\textsc{Acc} + 2\textsc{Exp}\big)$	$3\lvert p\rvert + \textsc{Sok}\big(\textsc{Sign}$ $+\mathcal{O}((L-\ell)t)\textsc{Exp}\big)$	$2\lvert p\rvert + \lvert P\rvert + \textsc{Sok}\big(2\textsc{Sign}$ $+2\textsc{Acc} + 3\textsc{Exp}\big)$

Table 1. Efficiency comparison between related work and our proposal

4.6 Security Theorem

We give the statement of security for our new proposed scheme.

Theorem 1. *In the random oracle model, our divisible e-cash scheme fulfills (i) the unforgeability under the assumptions that ESS is unforgeable and the bounded accumulator scheme fulfills the bound property; (ii) the anonymity under the zero-knowledge property of ZKPK and the DDH assumption; (iii) the identification of double-spender under the unforgeability of ESS and (iv) the exculpability under the one-more discrete logarithm assumption.*

Proof. We consider the four properties.
– **Anonymity**: we use a reduced "game proof technique" from Shoup. We denote by ϵ the probability that $\mathcal{A}$ succeeds in linking the spending protocol V_1 to a spending or withdrawal protocol.

During V_1, $\mathcal{A}$ gets a serial number S, a security tag T, a zero-knowledge proof of knowledge $\mathit{\Pi}$ and a value R. It is obvious that R does not reveal any Shannon information on the user identity. Under the zero-knowledge property of the proof of knowledge $\mathit{\Pi}$, in the random oracle model, the value $\mathit{\Pi}$ does not help $\mathcal{A}$ in winning the anonymity experiment. Thus, V_1 is replaced by V_2 in which $\mathcal{A}$ gets only S and T. The difference of probability between V_1 and V_2 is the probability of success of $\mathcal{A}$ in breaking the zero-knowledge property of $\mathit{\Pi}$, namely $Succ^{\mathsf{zk}}_{\mathit{\Pi},\mathcal{A}}(\lambda)$.

In V_2, we focus on T. The secret key $k_{L-\ell,j_0}$ is used only in the computation of T and Π. Indeed, knowing $T = \mathsf{upk} \cdot H^{Rk}$ and $T' = \mathsf{upk}' \cdot H^{R'k'}$, $\mathcal{A}$ has to decide if the same upk is embedded in both T and T'. This is assumed to be infeasible under the discrete logarithm (DL) assumption. Thus, V_2 is replaced by V_3 in which $\mathcal{A}$ gets only S. The difference of probability between V_2 and V_3 is the probability of success of $\mathcal{A}$ in breaking the DL problem, namely $Succ_{\mathcal{A}}^{\mathsf{dl}}(\lambda)$.

Finally, from S, $\mathcal{A}$ needs to decide whether or not two different node keys k and k' are related to the same root key k_0. This is assumed to be infeasible under a stronger variant of the Decisional Diffie-Hellman assumption (see [9] for details). We denote by $Adv_{\mathcal{A}}^{\mathsf{ddh}}(\lambda)$ the corresponding success probability.

We conclude that $Adv_{\mathcal{DCS},\mathcal{A}}^{\mathsf{anon}}(\lambda) \leq Succ_{\Pi,\mathcal{A}}^{\mathsf{zk}}(\lambda) + Adv_{\mathcal{A}}^{\mathsf{ddh}}(\lambda)$.

– **Unforgeability**: we use a reduction to either the unforgeability of the signature scheme or to the bound property of the accumulator scheme. Let $\mathcal{A}$ be an adversary that breaks the unforgeability property in polynomial time τ with a probability of success equal to ϵ. We interact with the black box adversary $\mathcal{A}$ by flipping at random a coin and playing one among two games.

Game 1. We construct a machine $\mathcal{M}_1$ that breaks the existential unforgeability of ESS+ with access to a signing oracle Sign:

- In the Setup procedure, $\mathcal{M}_1$ sets the group defined for the signature scheme as $\mathcal{G}_p$ and $\mathcal{A}$ is given the public key $\mathsf{bpk} = \mathsf{spk}$ of the signature scheme.
- when $\mathcal{A}$ ask for a withdrawal, dc is incremented by 1 and $\mathcal{M}_1$, playing the role of the bank, interacts with $\mathcal{A}$ by interacting with the oracle Sign to obtain ESS+ signatures. Each time, $\mathcal{M}_1$ stores in askdb the signature and the corresponding messages;
- when $\mathcal{A}$ asks for a spending protocol of 2^ℓ coins, sp is incremented by 2^ℓ and $\mathcal{M}_1$, playing the role of the merchant, rewinds $\mathcal{A}$ during its computation of the zero-knowledge proof of knowledge by using standard techniques (and in the random oracle model) in order to extract and store in recdb the signatures and the corresponding messages used by $\mathcal{A}$.
- at any time of the unforgeability experiment, $\mathcal{A}$ sets cont = false and with probability ϵ, we have $2^L \cdot \mathsf{dc} < \mathsf{sp}$. In case there is one entry in $\mathsf{recdb} \setminus \mathsf{askdb}$. This entity is obviously a forgery.

Game 2. We construct a machine $\mathcal{M}_2$ that breaks the bound property of the accumulator scheme:

- In the Setup procedure, $\mathcal{M}_2$ sets the group defined for the signature scheme as $\mathcal{G}_p$ and $\mathcal{A}$ is given the public key $\mathsf{bpk} = \mathsf{spk}$ of the signature scheme. Next, $\mathcal{M}_2$ generates the $L+2$ accumulators. $\mathcal{M}_2$ maintains a database accdb containing all accumulators generated by $\mathcal{A}$ together with the bound of the corresponding bounded accumulator and the obtained accumulated values;
- when $\mathcal{A}$ ask for a withdrawal, $\mathcal{M}_2$ plays the role of the bank;

- when $\mathcal{A}$ asks for a spending protocol of 2^ℓ coins, sp is incremented by 2^ℓ and $\mathcal{M}_2$, playing the role of the merchant, rewinds $\mathcal{A}$ during its computation of the zero-knowledge proof of knowledge, using standard technique (and in the random oracle model) in order to extract the accumulators. Next, $\mathcal{M}_2$ stores in accdb the accumulator (if not already present in the database) and the corresponding accumulated values;
- at any time of the unforgeability experiment, $\mathcal{A}$ sets cont = false and with probability ϵ, we have $2^L \cdot \mathsf{dc} < \mathsf{sp}$. In case there is one entry in accdb such that there are more accumulated values than the bound for this accumulator, $\mathcal{M}_2$ breaks the bound property of the accumulator scheme.

As a consequence, $Succ^{\mathsf{unforge}}_{\mathcal{DCS},\mathcal{A}}(\lambda) = \frac{1}{2}\big(Succ^{\mathsf{unforge}}_{\mathcal{S}ign,\mathcal{A}}(\lambda) + Succ^{\mathsf{bound}}_{\mathcal{A}cc,\mathcal{A}}(\lambda)\big)$.

– **Identification of Double Spender**: we use a reduction to the unforgeability of the signature scheme. Let $\mathcal{A}$ be an adversary breaking the identification of double spender in polynomial time τ with a probability of success ϵ. We consider a black box adversary and construct a machine $\mathcal{M}$ which breaks the unforgeability of the signature scheme. We can access to the group $\mathcal{G}_p$, the public key spk and interact with a signing oracle:

- In the Setup procedure, $\mathcal{M}$ sets the group defined for the signature scheme as $\mathcal{G}_p$ and $\mathcal{A}$ is given the public key $\mathsf{bpk} = \mathsf{spk}$ of the signature scheme.
- when $\mathcal{A}$ ask for a withdrawal, dc is incremented by 1 and $\mathcal{M}_1$, playing the role of the bank, interacts with $\mathcal{A}$ by interacting with the Sign oracle to obtain ESS+ signatures. Each time, $\mathcal{M}$ stores in askdb the signature and the corresponding messages;
- when $\mathcal{A}$ asks for a spending protocol, $\mathcal{M}$, playing the role of the merchant, rewinds the adversary $\mathcal{A}$ during its computation of the zero-knowledge proof of knowledge to extract and store in recdb the signatures and the corresponding messages used by $\mathcal{A}$;
- at any time of the experiment, $\mathcal{A}$ outputs one spent coin rco and, with probability ϵ, the Deposit and the VerifyGuilt procedures output 0. Thus, $\mathcal{M}$ takes on input rco and the other coin with the same serial number and extracts, in $\Pi \in \mathsf{rco}$ and using standard techniques, both signatures and the corresponding messages. Necessarily, one of the two signatures is not an output of the signing oracle since the signed upk is not detected by the Identify algorithm. Thus, $\mathcal{M}$ has produced a forge on the signature scheme.

We have $Succ^{\mathsf{idds}}_{\mathcal{DCS},\mathcal{A}}(\lambda) = Succ^{\mathsf{unforge}}_{\mathcal{S}ign,\mathcal{A}}(\lambda)$.

– **Exculpability**: suppose that an adversary $\mathcal{A}$ succeeded in breaking the exculpability property. That means that there are two valid spends with the same serial number S (or, either S can be computed from S', or S' can be computed from S) and two different proofs Π and Π' and two different correct randoms R and R'. As spendings are correct, the proofs include that both T and T' are well formed. Thus, since the user is honest, $\mathcal{A}$ has faked T or T'.

We now use $\mathcal{A}$ to break the one-more discrete logarithm problem [3]. Given $l+1$ values, we have to find the discrete logarithm of all these values, and we can ask a discrete logarithm oracle at most l times. We first associate each value to the public key of one user (assuming there are at most l users) and we ask the oracle each time $\mathcal{A}$ corrupt a user. It is possible to simulate all withdrawals and spends using standard techniques (in the random oracle model). $\mathcal{A}$ finally outputs two correctly formed T and T' and the associated proofs of validity. Thus, T and T' are both formed from the same public key of a honest user.

From the two proofs of validity, we can extract the user secret key and thus break the one-more discrete logarithm. Indeed, since the user is honest, this discrete logarithm has not been requested to the oracle. We consequently have $Succ^{\mathsf{exculp}}_{\mathcal{DCS},\mathcal{A}}(\lambda) = Succ^{\mathsf{omdl}}_{\mathcal{A}}(\lambda)$, which concludes the proof. □

References

1. M.H. Au, W. Susilo, and Y. Mu. Practical anonymous divisible e-cash from bounded accumulators. In *Financial Cryptography'08*, volume to appear of *LNCS*. Springer, 2008.
2. M.H. Au, Q. Wu, W. Susilo, and Y. Mu. Compact e-cash from bounded accumulator. In *CT-RSA'07*, volume 4377 of *LNCS*, pages 178–195. Springer, 2007.
3. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme. *J. Cryptology*, 16(3):185–215, 2003.
4. J. Camenisch, A. Kiayias, and M. Yung. On the portability of generalized schnorr proofs. In *EUROCRYPT'09*, volume 5479 of *LNCS*, pages 425–442. Springer, 2009.
5. J. Camenisch, M. Kohlweiss, and C. Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credential. In *PKC'09*, volume to appear of *LNCS*. Springer, 2009.
6. J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Crypto'02*, volume 2442 of *LNCS*, pages 61–76. Springer, 2002.
7. J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Crypto'04*, LNCS, pages 56–72. Springer, 2004.
8. S. Canard, I. Coisel, and J. Traoré. Complex zero-knowledge proofs of knowledge are easy to use. In *ProvSec'07*, volume 4784 of *LNCS*, pages 122–137. Springer, 2007.
9. S. Canard and A. Gouget. Divisible e-cash systems can be truly anonymous. In *Eurocrypt'07*, volume 4515 of *LNCS*, pages 482–497. Springer, 2007.
10. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.
11. T. Nakanishi and Y. Sugiyama. Unlinkable divisible electronic cash. In *ISW'00*, volume 1975 of *LNCS*, pages 121–134. Springer, 2000.
12. L. Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA'05*, volume 3376 of *LNCS*, pages 275–292. Springer, 2005.

A Bounded Accumulator with Additional Procedure

We give a concrete example of accumulator based on [12]. Let $\mathcal{G}$ and $\tilde{\mathcal{G}}$ be two cyclic groups of prime order p, G (resp. $\tilde{G}$) is a generator of $\mathcal{G}$ (resp. $\tilde{\mathcal{G}}$). Let $\mathcal{G}_T$ be a multiplicative group of order p. We refer a bilinear structure for the groups $(\mathcal{G}, \tilde{\mathcal{G}}, \mathcal{G}_T)$. Let e be a bilinear map $e : \mathcal{G} \times \tilde{\mathcal{G}} \longrightarrow \mathcal{G}_T$.

Using [12], the number of values accumulated in a single accumulator is limited to a fixed number which is here denoted by s. In our construction, we use a new proof of knowledge, denoted $\textsc{Pok}(\mathsf{Acc} : (k_1, \cdots, k_\ell) \in \mathsf{Acc})$, to prove that several values known by the verifier are accumulated in a secret accumulator.

Let $U_0 \in \mathcal{G}$, $\tilde{V}_0 \in \tilde{\mathcal{G}}$, $\alpha \in \mathbb{Z}_p^*$, $U_i = U_0^{\alpha^i}$ for all $i \in \{1, \cdots, s\}$ and $\tilde{V}_1 = \tilde{V}_0^{\alpha}$. The values $(k_1, \cdots, k_s)$ are accumulated in Acc as $\mathsf{Acc} = U_0^{\prod_{i=1}^{s}(k_i+\alpha)}$. We denote by W the value $U_0^{\prod_{j \notin [1,\ell]}(k_j+\alpha)}$. Thus, we have $\mathsf{Acc} = W^{\prod_{j \in [1,\ell]}(k_j+\alpha)}$. Let $\mathsf{P}(\alpha) = \prod_{j \in [1,\ell]}(k_j + \alpha) = \sum_{j_1}^{\ell} \mathsf{p}_j \alpha^j$ where the p_j's are product of the k_i's and the degree of P is ℓ. More precisely, we assume that there exists a public function $\mathsf{F}(\{k_1, \cdots, k_\ell\}) = \{\mathsf{p}_1, \cdots, \mathsf{p}_\ell\}$. We need to introduce some additional public parameters: $\tilde{V}_i = \tilde{V}_0^{\alpha^i}$ for all $i \in \{2, \cdots, s\}$. Then , we have $e(\mathsf{Acc}, \tilde{V}_0) = e(W^{\prod_{j \in [1,\ell]}(k_j+\alpha)}, \tilde{V}_0) = e(W, \tilde{V}_0^{\prod_{j \in [1,\ell]}(k_j+\alpha)}) = e(W, \tilde{V}_0 \prod_{j=1}^{\ell} \tilde{V}_j^{\mathsf{p}_j})$. In our setting, the values k_i's are public, and thus it is also the case for the values p_j's. Our ZKPK is then: $\textsc{Pok}\big(\mathsf{Acc}, W : e(\mathsf{Acc}, v_0) = e(W, \tilde{V}_0 \prod_{j=1}^{\ell} \tilde{V}_j^{\mathsf{p}_j})\big)$.

B Proof of Validity of a Spending

The first part of the proof, consisting in proving that one knows usk and $k_{L-\ell,j_0}$ such that $T = G^{\mathsf{usk}} \cdot (H^R)^{k_{L-\ell,j_0}}$, $k_{L-\ell+1,2j_0} = g_0^{k_{L-\ell,j_0}}$ and $k_{L-\ell+1,2j_0+1} = g_1^{k_{L-\ell,j_0}}$ is simply done by using standard discrete logarithm based proof of knowledge. The next part of the proof consists in proving that several values known by the verifier are accumulated in a secret accumulator which is signed by the bank, using the ESS+ scheme. We describe the case of $\mathsf{Acc}_{L-\ell+1}$; the case of Acc is similar.

We first need a proof that the tuple $(k_{L-\ell+1,2j_0}, k_{L-\ell+1,2j_0+1}) \in \mathsf{Acc}_{L-\ell+1}$. Everyone can compute $\{\mathsf{p}_1, \mathsf{p}_2\} = \mathsf{F}(k_{L-\ell+1,2j_0}, k_{L-\ell+1,2j_0+1})$. Since there exists the public relation $e(\mathsf{Acc}_{L-\ell+1}, \tilde{V}_0) = e(W, \tilde{P})$ with $\tilde{P} = \tilde{V}_0 \tilde{V}_1^{\mathsf{p}_1} \tilde{V}_2^{\mathsf{p}_2}$, the second part of the proof of knowledge is then $\textsc{Pok}\big(s, \mathsf{Acc}_{L-\ell+1}, \sigma_{L-\ell+1} : \sigma_{L-\ell+1} = \textsc{Sign}(\mathsf{Acc}_{L-\ell+1}, u) \wedge e(\mathsf{Acc}_{L-\ell+1}, \tilde{V}_0) = e(W, \tilde{P})\big)$.

The signature $\sigma_{L-\ell+1}$ is an ESS+ signature on the message $M = (\mathsf{Acc}_{L-\ell+1}, s, L-\ell+1)$. Thus, $\sigma_{L-\ell+1}$ is composed by the elements $\varSigma_1 = X(\mathsf{Acc}_{L-\ell+1} G_0^a)^c$, $\varSigma_2 = \big(G_1 G_2^a G_3^b H_1^s H_2^{L-\ell+1}\big)^{\frac{1}{x+c}}$ and $\tilde{\varSigma}_3 = \tilde{H}^c$ where $a, b, c \in_R \mathbb{Z}_p^*$ and X, G_0, G_1, G_2, G_3, H_1, H_2, $\tilde{H}$ are public. These values verify the following relations: $e(\varSigma_2, \tilde{\varSigma}_3 \tilde{Z}) = e(G_1, \tilde{H}) e(G_2, \tilde{H})^a e(G_3, \tilde{H})^b e(H_1, \tilde{H})^s e(H_2, \tilde{H})^{L-\ell+1}$ and

$e(\Sigma_1, \tilde{H}) = Y e(\mathsf{Acc}_{L-\ell+1} G_0^a, \tilde{\Sigma}_3)$. The second part of the proof is finally

$$\begin{aligned}&\textsc{Pok}\big(s, \mathsf{Acc}_{L-\ell+1}, \Sigma_1, \Sigma_2, \tilde{\Sigma}_3 : \\ &e(\mathsf{Acc}_{L-\ell+1}, \tilde{V}_0) = e(W, \tilde{P}) \wedge e(\Sigma_1, \tilde{H}) = Y e(\mathsf{Acc}_{L-\ell+1} G_0^a, \tilde{\Sigma}_3) \wedge \\ &e(\Sigma_2, \tilde{\Sigma}_3 \tilde{Z}) = e(G_1, \tilde{H}) e(G_2, \tilde{H})^a e(G_3, \tilde{H})^b e(H_1, \tilde{H})^s e(H_2, \tilde{H})^{L-\ell+1}\big).\end{aligned}$$

This proof is generated using standard techniques and the results in [2, 1].

Appeared in R. Sion (Ed.): Financial Cryptography 2010, preproceedings.