This is the Full Version of the Extended Abstract that appears
in the Proceedings of the 18th International Conference on Practice and Theory in Public-Key Cryptography (PKC '15)
(30 March – 1 April 2015, Gaithersburg, Maryland, USA)
Jonathan Katz Ed., Springer-Verlag, LNCS ????, pages ???–???.

Divisible E-Cash Made Practical

Sébastien Canard[1], David Pointcheval[2], Olivier Sanders[1,2] and Jacques Traoré[1]

[1] Orange Labs, Applied Crypto Group, Caen, France
[2] École normale supérieure, CNRS & INRIA, Paris, France

Abstract. Divisible E-cash systems allow users to withdraw a unique coin of value 2^n from a bank, but then to spend it in several times to distinct merchants. In such a system, whereas users want anonymity of their transactions, the bank wants to prevent, or at least detect, double-spending, and trace the defrauders. While this primitive was introduced two decades ago, quite a few (really) anonymous constructions have been introduced. In addition, all but one were just proven secure in the random oracle model, but still with either weak security models or quite complex settings and thus costly constructions. The unique proposal, secure in the standard model, appeared recently and is unpractical. As evidence, the authors left the construction of an efficient scheme secure in this model as an open problem.
In this paper, we answer it with the first efficient divisible E-cash system secure in the standard model. It is based on a new way of building the coins, with a unique and public global tree structure for all the coins. Actually, we propose two constructions: a very efficient one in the random oracle model and a less efficient, but still practical, in the standard model. They both achieve constant time for withdrawing and spending coins, while allowing the bank to quickly detect double-spendings by a simple comparison of the serial numbers of deposited coins to the ones of previously spent coins.

1 Introduction

Electronic Cash (E-cash), introduced by Chaum [Cha82,Cha83], is the digital analogue of regular money. It allows users to withdraw coins from a bank and to spend them to merchants, in an anonymous way, thus perfectly emulating conventional cash transactions.

Unfortunately, with E-cash, as any digital data, coins can easily be duplicated, and thus spent several times. It is therefore essential to be able to detect double-spending and even to identify the defrauders. As for group signatures [BMW03,BSZ05], one solution could be to give to a specific entity the ability of revoking anonymity for any transaction of his choice. However, such an approach (called *fair* E-cash [BGK95,SPC95]) weakens the anonymity of the scheme because, ideally, user's privacy should be guaranteed as long as the user is honest. Moreover, such an entity should be trusted by all the users or distributed among a large group of authorities, which makes the tracing procedure, in case of fraud, quite costly.

E-cash systems achieve their ultimate goal when the user's side of the protocol is implemented on a mobile device (*e.g.* a smartphone). However, the limited power of such devices along with the strong time constraints of electronic transactions require very efficient withdrawal and spending procedures. Moreover, even if the bank is more powerful than the users, it has to centralize a huge number of transactions, and thus double-spending detection should be made as efficient as possible. Reconciling security requirements with efficiency is therefore the main challenge when designing E-cash systems.

1.1 Related Work

Compact E-Cash. Camenisch, Hohenberger and Lysyanskaya [CHL05] described the first compact E-cash system (later extended to systems supporting additional features [CLM07,CGT08,CG08]), allowing users to withdraw wallets with 2^n coins at once. Each coin is associated with a unique serial number, allowing the bank to efficiently detect double-spending. Unfortunately, while the withdrawal of many coins can be done at once, the spending procedure is done coin by coin, which is a major drawback for concrete use. Indeed, in order to provide a good granularity, one must use coins of one cent, and thus transactions often involve thousands of coins. An alternative could be the use of coins with several denominations, but then one should use several systems in parallel for each value, and in addition anonymity would be more difficult to achieve since users would withdraw different kinds

of coins. Then, the bank could classify the users according to their withdrawals and then infer where users spend their money from the coins the merchants deposit.

Divisible E-Cash Systems. The purpose of divisible E-cash systems is to address this problem of splitting coins of large values. As above, users withdraw a large coin of value 2^n (or withdraw 2^n coins at once), but can spend it in several times by dividing it (or spend several coins at once): more concretely, one can spend a coin of value 2^ℓ, for any $0 \leq \ell \leq n$, at once, instead of spending 2^ℓ unitary coins, which is clearly much more efficient.

Since their introduction, many schemes have been proposed [OO91,Oka95,CFT98,NS00] but they only achieved quite weak levels of anonymity. Indeed, transactions involving the same coin (from the same withdrawal) were all linkable, except with [NS00], which however still reveals which part of the coin is spent (which is not compatible with the highest security notion) and in addition requires a trusted authority to recover spenders' identities.

Canard and Gouget [CG07] introduced the first truly anonymous E-cash system. Unfortunately, it makes use of complex zero-knowledge proofs of knowledge (ZKPK) and of groups of different but related orders, whose generation requires a huge computational power. Despite its inefficiency (pointed out in [ASM08,CG10]) this system was a proof of concept: a "truly" anonymous divisible E-cash system is possible. Au, Susilo, and Mu [ASM08] proposed a more efficient scheme but at the cost of an unconventional security model where the bank is only ensured that it will not loose money on average (provided that it can legally impose fines on users, which is not necessarily the case). Canard and Gouget [CG10] later proposed another construction, but still with groups of different orders, leading to rather inefficient ZKPK. All these schemes were proven secure in the random oracle model (ROM) [BR93]. More recently, Izabachène and Libert [IL12] provided the first construction with security proven in the standard model. However their construction is rather inefficient, especially the deposit phase whose computational cost for the bank depends on the number of *previously* deposited coins with a pairing computation between every new coin and every past coin. Such a downside makes the scheme impractical, leading the authors to leave the construction of an efficient scheme secure in the standard model as an open problem.

1.2 Our Contribution

In this paper, we address this open problem, with the first *really* efficient divisible E-cash system. It can be designed either in the ROM or the standard model. Our main contribution is a new way for building the serial numbers. As noticed in [IL12], the use of serial numbers is indeed the best approach for the bank to quickly detect double-spending.

In previous solutions [CG07,ASM08,CG10], every divisible coin is associated with a binary tree whose nodes correspond to expendable amounts. When a user withdraws a coin, he selects a random number k_ϵ associated with the root of the tree and then computes, for each internal node s, the corresponding number k_s, using k_ϵ and a one-way function. The user then obtains signatures on these numbers (or on elements accumulating them) from the bank and defines the coin to be the binary tree along with the signatures. However, to ensure that the user will not spend more than the amount he has withdrawn, he will have to prove (either during the spending or the withdrawal protocol) that the tree (and so the numbers k_s) is well-formed. Unfortunately, the construction from [ASM08] is not compatible with any zero-knowledge proof construction because the numbers k_s are computed using a hash function, modeled as a random oracle. The authors therefore proposed a *cut-and-choose* method to detect cheaters which leads to the problem mentioned above. Canard and Gouget [CG07,CG10] used groups of different orders which are compatible with zero-knowledge proofs in the ROM (although they are rather inefficient) but not in the standard model since the Groth-Sahai [GS08] methodology does no longer work in this setting.

In our construction, we use a totally different approach: instead of using one tree by coin, we define, in the public parameters, one single tree which will be common to all the coins. The key point

of this solution is obvious: users no longer have to prove that the tree is well-formed. Moreover, it considerably alleviates the withdrawal procedure since the bank no longer has to certify each tree.

We will use bilinear groups (*i.e.* a set of three cyclic groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p along with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$), which are compatible with Groth-Sahai proofs. In a nutshell, our system works as follows: it uses a unique tree T of depth n (for coins of value 2^n), where each leaf f is associated with an element $\chi_f \in \mathbb{G}_T$ and each internal node s is associated with an element $g_s \in \mathbb{G}_1$. In addition to these group elements, the public parameters also contain, for each leaf f and any node s on the path to f, an element $\widetilde{g}_{s \mapsto f} \in \mathbb{G}_2$ such that $e(g_s, \widetilde{g}_{s \mapsto f}) = \chi_f$ (a setup algorithm will efficiently generate these parameters). When a user withdraws a coin, he gets a certificate on some random scalar $x \in \mathbb{Z}_p$, which will implicitly define all the serial numbers associated with this coin as χ_f^x for each leaf f. To spend a node s of height ℓ in the tree, corresponding to a value of 2^ℓ, the user can compute $t_s \leftarrow g_s^x$ and prove that it is well-formed: such a proof can easily be done in either the ROM or the standard model. Informally, the unlinkability property follows from the fact that it is hard, given g_s^x and $g_{s'}^x$ for two nodes s and s', to decide whether they were computed using the same x (and thus belong to the same tree) under the XDH assumption. However, using the elements $\widetilde{g}_{s \mapsto f}$, the bank will be able to recover the 2^ℓ serial numbers by computing $e(t_s, \widetilde{g}_{s \mapsto f}) = \chi_f^x$ for each f in the subtree below s. A double-spending means that two transactions involve two nodes s and s' with non-disjoint subtrees: a common leaf f is in both subtrees issued from s and s', and so the bank will detect a collision between the serial numbers since $e(t_s, \widetilde{g}_{s \mapsto f}) = \chi_f^x = e(t_{s'}, \widetilde{g}_{s' \mapsto f})$.

Of course, several problems have to be addressed to fulfill all the security requirements, but the above key idea allows to design a system with constant cost for both the withdrawal and spending protocols, which can be proven secure in either the random oracle and the standard models.

1.3 Organization

In Section 2, we review some classical definitions and notations. Section 3 describes the security model for divisible E-cash. We provide a high level description of our construction in Section 4, and a more detailed presentation in Section 5. Eventually, security proofs are given in Section 6.

2 Preliminaries

Bilinear Groups. Bilinear groups are a set of three cyclic groups $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p along with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ with the following properties:

1. for all $g \in \mathbb{G}_1, \widetilde{g} \in \mathbb{G}_2$ and $a, b \in \mathbb{Z}_p$, $e(g^a, \widetilde{g}^b) = e(g, \widetilde{g})^{a \cdot b}$;
2. for $g \neq 1_{\mathbb{G}_1}$ and $\widetilde{g} \neq 1_{\mathbb{G}_2}$, $e(g, \widetilde{g}) \neq 1_{\mathbb{G}_T}$;
3. e is efficiently computable.

Computational Assumptions. Our construction will rely on several computational assumptions that have been considered reasonable in well chosen groups:

- the DL **assumption** holds in the group $\mathbb{G}$ if it is hard, given $(g, g^x) \in \mathbb{G}^2$, to output x;
- the XDH **assumption** holds in bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ if it is hard, given $(g, g^x, g^y, g^z) \in \mathbb{G}_1^4$, to decide whether $z = x \cdot y$ or z is random;
- the $q-\mathsf{SDH}$ **assumption** [BB08] holds in a group $\mathbb{G}$ if it is hard, given a tuple $(g, g^x, g^{x^2}, \ldots, g^{x^q}) \in \mathbb{G}^{q+1}$, to output a pair $(m, g^{\frac{1}{m+x}})$.

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

- the key generation algorithm $\Sigma.\mathtt{Keygen}$ which outputs a pair of signing and verification keys $(\mathsf{sk}, \mathsf{pk})$ – we assume that sk always contains pk;
- the signing algorithm $\Sigma.\mathtt{Sign}$ algorithm which, on input the signing key sk and a message m, outputs a signature σ;

– and the verification algorithm $\Sigma.\texttt{Verify}$ which, on input m, σ and pk, outputs 1 if σ is a valid signature on m under pk and 0 otherwise.

The standard security notion for a signature scheme is *existential unforgeability under chosen message attacks* (EUF-CMA) [GMR88] which means that it is hard, even given access to a signing oracle, to output a valid pair (m, σ) for a message m never asked to the oracle. In this paper we will also use two weaker different security notions for signature schemes. The former is the security against selective chosen message attacks, which limits the oracle queries to be asked before having seen the key pk. The latter is a *strong* one-time security notion where the adversary can ask only one query to the signing oracle, but *strong* means that an output with a new signature on an already signed message is also considered a forgery. In our instantiation in Appendix A, we use a deterministic one-time signature, and thus one-time security is equivalent to strong one-time security.

3 Divisible E-cash System

3.1 Syntax

As in [CG07,CG10], a divisible e-cash system is defined by the following algorithms, that involve at least three entities: the bank $\mathcal{B}$, a user $\mathcal{U}$ and a merchant $\mathcal{M}$. Although not necessary, it is often easier to assume that the $\texttt{Setup}$ algorithm is run by a trusted entity (we refer to Remark 3 in Section 4 for more discussion).

- $\texttt{Setup}(1^k, V)$: On inputs a security parameter k and an integer V, this probabilistic algorithm outputs the public parameters $p.p.$ for divisible coins of global value V. We assume that $p.p.$ are implicit to the other algorithms, and that they include k and V. They are also an implicit input to the adversary, we will then omit them.
- $\texttt{BKeygen}()$: This probabilistic algorithm executed by the bank $\mathcal{B}$ outputs a key pair $(\mathsf{bsk}, \mathsf{bpk})$. It also sets L as an empty list, that will store all deposited coins. We assume that bsk contains bpk.
- $\texttt{Keygen}()$: This probabilistic algorithm executed by a user $\mathcal{U}$ (resp. a merchant $\mathcal{M}$) outputs a key pair $(\mathsf{usk}, \mathsf{upk})$ (resp. $(\mathsf{msk}, \mathsf{mpk})$). We assume that usk (resp. msk) contains upk (resp. mpk).
- $\texttt{Withdraw}(\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk}))$: This is an interactive protocol between the bank $\mathcal{B}$ and a user $\mathcal{U}$. At the end of this protocol, the user gets a divisible coin C of value V or outputs $\perp$ (in case of failure) while the bank stores the transcript $\texttt{Tr}$ of the protocol execution or outputs $\perp$.
- $\texttt{Spend}(\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, \mathsf{mpk}, v), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, v))$: This is an interactive protocol between a user $\mathcal{U}$ and a merchant $\mathcal{M}$. At the end of the protocol the merchant gets a master serial number Z of value v (the amount of the transaction they previously agreed on) along with a proof of validity Π or outputs $\perp$. $\mathcal{U}$ either updates C or outputs $\perp$.
- $\texttt{Deposit}(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (v, Z, \Pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: This is an interactive protocol between a merchant $\mathcal{M}$ and the bank $\mathcal{B}$. $\mathcal{B}$ checks that Π is valid on v and Z and that (v, z, Π) has never been deposited (corresponding to the case of a cheating merchant). $\mathcal{B}$ then recovers the m (for some $m \geq v$) serial numbers $z_1, \ldots, z_m$ corresponding to this transaction and checks whether, for some $1 \leq i \leq m$, $z_i \in L$. If none of the serial numbers is in L, then the bank credits $\mathcal{M}$'s account of v, stores (v, Z, Π) and appends $\{z_1, \ldots, z_m\}$ to L. Else, there is at least an index $i \in \{1, \ldots, m\}$ and a serial number z' in L such that $z' = z_i$. The bank then recovers the tuple (v', Z', Π') corresponding to z' and publishes $[(v, Z, \Pi), (v', Z', \Pi')]$.
- $\texttt{Identify}((v_1, Z_1, \Pi_1), (v_2, Z_2, \Pi_2), \mathsf{bpk})$: On inputs two different valid transcripts (v_1, Z_1, Π_1) and (v_2, Z_2, Π_2), this deterministic algorithm outputs a user's public key upk if there is a collision between the serial numbers derived from Z_1 and from Z_2, and $\perp$ otherwise.

It is worthy to note that the $\texttt{Identify}$ algorithm does not require knowledge of any secret element and can thus be run by anyone. So, there is no need for a $\texttt{VerifyGuilt}$ algorithm (as provided in [CG07,CG10]) since any entity can be convinced of the culpability of a user by recovering his public key upk from the transcripts published by the bank.

$\mathtt{Exp}_{\mathcal{A}}^{anon-b}(1^k, V)$

1. $p.p. \leftarrow \mathtt{Setup}(1^k, V)$
2. $\mathsf{bpk} \leftarrow \mathcal{A}()$
3. $(v, \mathsf{upk}_0, \mathsf{upk}_1, \mathsf{mpk}) \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Add}, \mathcal{O}\mathtt{Corrupt}, \mathcal{O}\mathtt{AddCorrupt}, \mathcal{O}\mathtt{Withdraw}_{\mathcal{U}}, \mathcal{O}\mathtt{Spend}}()$
4. If upk_0 or upk_1 is not registered, then return 0
5. If $c_{\mathsf{upk}_i} > m_{\mathsf{upk}_i} \cdot V - v$ for $i \in \{0, 1\}$, then return 0
6. $(v, Z, \Pi) \leftarrow \mathtt{Spend}(\mathcal{C}(\mathsf{usk}_b, C, \mathsf{mpk}, v), \mathcal{A}())$
7. $b^* \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Add}, \mathcal{O}\mathtt{Corrupt}, \mathcal{O}\mathtt{AddCorrupt}, \mathcal{O}\mathtt{Withdraw}_{\mathcal{U}}, \mathcal{O}\mathtt{Spend}^*}()$
8. If upk_0 or upk_1 has been corrupted, then return 0
9. Return $(b = b^*)$

Fig. 1. Anonymity Security Game

3.2 Security Model

Besides the usual *correctness* property (informally meaning that an honest user running a `Withdraw` protocol with an honest bank will receive a divisible coin accepted by any honest merchant), a secure e-cash system must achieve several security properties, defined through games between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. Our security model makes use of the following oracles.

- $\mathcal{O}\mathtt{Add}()$ is an oracle used by $\mathcal{A}$ to register a new honest user (resp. merchant). The challenger runs the `Keygen` algorithm, stores usk (resp. msk) and returns upk (resp. mpk) to $\mathcal{A}$. In this case, upk (resp. mpk) is said *honest*.
- $\mathcal{O}\mathtt{Corrupt}(\mathsf{upk}/\mathsf{mpk})$ is an oracle used by $\mathcal{A}$ to corrupt an honest user (resp. merchant) whose public key is upk (resp. mpk). The challenger then returns the corresponding secret key usk (resp. msk) to $\mathcal{A}$ along with the secret values of every coin withdrawn by this user. From now on, upk (resp. mpk) is said *corrupted*.
- $\mathcal{O}\mathtt{AddCorrupt}(\mathsf{upk}/\mathsf{mpk})$ is an oracle used by $\mathcal{A}$ to register a new corrupted user (resp. merchant) whose public key is upk (resp. mpk). In this case, upk (resp. mpk) is said *corrupted*. The adversary could use this oracle on a public key already registered (during a previous $\mathcal{O}\mathtt{Add}$ query) but for simplicity, we reject such case as it will gain nothing more than using the $\mathcal{O}\mathtt{Corrupt}$ oracle on the same public key.
- $\mathcal{O}\mathtt{Withdraw}_{\mathcal{U}}(\mathsf{upk})$ is an oracle that executes the user's side of the `Withdraw` protocol. This oracle will be used by $\mathcal{A}$ playing the role of the bank against the user with public key upk.
- $\mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}(\mathsf{upk})$ is an oracle that executes the bank's side of the `Withdraw` protocol. This oracle will be used by $\mathcal{A}$ playing the role of a user whose public key is upk against the bank.
- $\mathcal{O}\mathtt{Spend}(\mathsf{upk}, v)$ is an oracle that executes the user's side of the `Spend` protocol for a value v. This oracle will be used by $\mathcal{A}$ playing the role of the merchant $\mathcal{M}$.

In our experiments, we denote users by their public keys upk, the value spent by user upk during $\mathcal{O}\mathtt{Spend}$ queries by c_{upk}, and the number of divisible coins withdrawn by this user by m_{upk}. This means that the total amount available by a user upk is $m_{\mathsf{upk}} \cdot V$.

Anonymity. Informally, anonymity requires that the bank, even helped by malicious users and merchants, cannot learn anything about a spending other than what is available from side information from the environment. We define the anonymity experiments $\mathtt{Exp}_{\mathcal{A}}^{anon-b}(1^k, V)$ as described on Figure 1. After the challenge phase, the $\mathcal{O}\mathtt{Spend}$ queries are restricted to avoid $\mathcal{A}$ trivially wins: $\mathcal{A}$ then has access to a $\mathcal{O}\mathtt{Spend}^*$ oracle that is the same as the $\mathcal{O}\mathtt{Spend}$ oracle except that it cannot be asked on upk_i if $c_{\mathsf{upk}_i} > m_{\mathsf{upk}_i} \cdot V - v$, for $i \in \{0, 1\}$. Otherwise one can easily deduce which user has spent v during the challenge phase.

We define $\mathtt{Adv}_{\mathcal{A}}^{anon}(1^k, V)$ as $\Pr[\mathtt{Exp}_{\mathcal{A}}^{anon-1}(1^k, V)] - \Pr[\mathtt{Exp}_{\mathcal{A}}^{anon-0}(1^k, V)]$. A divisible e-cash system is *anonymous* if, for any probabilistic polynomial adversary $\mathcal{A}$, this advantage is negligible. Of course, the adversary must choose the users involved in the challenge phase among the registered users, and

$\mathtt{Exp}^{tra}_{\mathcal{A}}(1^k, V)$

1. $p.p. \leftarrow \mathtt{Setup}(1^k, V)$
2. $(\mathsf{bsk}, \mathsf{bpk}) \leftarrow \mathtt{BKeygen}()$
3. $[(v_1, Z_1, \Pi_1), \ldots, (v_u, Z_u, \Pi_u)] \xleftarrow{\$} \mathcal{A}^{\mathcal{O}\mathtt{Add}, \mathcal{O}\mathtt{Corrupt}, \mathcal{O}\mathtt{AddCorrupt}, \mathcal{O}\mathtt{Withdraw}_{\mathcal{B}}, \mathcal{O}\mathtt{Spend}}(\mathsf{bpk})$
4. If $\sum_{i=1}^{u} v_i > m \cdot V$ and $\forall i \neq j, \mathtt{Identify}((v_i, Z_i, \Pi_i), (v_j, Z_j, \Pi_j)) = \perp$, then return 1
5. Return 0

Fig. 2. Traceability Security Game

never corrupted, and cannot ask them to spend more than withdrawn, hence restrictions in steps 4, 5, and 8 respectively.

Remark 1. The scheme from [NS00] achieves an *unlinkability* property, meaning that it is hard to link two spendings from the same coin. This protocol makes use of a tree for the global coin, and each transcript reveals which part of the tree (*i.e.* which node) is spent. In some cases, this property can be enough (we describe informally in Section 4.2 a protocol fulfilling this property) but we stress that a scheme revealing the spent nodes in a tree structure is not anonymous (according to our above model) even if it is unlinkable (and our main scheme given in Section 5 is anonymous in the sense of the above definition). Indeed, to break the anonymity of such a scheme, the adversary can make one $\mathcal{O}\mathtt{Withdraw}_{\mathcal{U}}(\mathsf{bsk}, \mathsf{usk}_i)$ and then $(V-1)$ $\mathcal{O}\mathtt{Spend}(\mathsf{upk}_i, 1)$ queries, for each user ($i \in \{0, 1\}$). Therefore, it will only remain one unspent node s_{upk_i} for each user. If the nodes are randomly selected among the unspent nodes during a spending (which is even worse if this is deterministic or chosen by the adversary), with overwhelming probability, the two unspent nodes will not be the same for the two users upk_0 and upk_1: $s_{\mathsf{upk}_0} \neq s_{\mathsf{upk}_1}$. The node involved in the challenge phase will then reveal the user identity.

Traceability. Informally, traceability requires that no coalition of malicious users can spend more than they have withdrawn, without revealing their identity. We define the traceability experiment $\mathtt{Exp}^{tra}_{\mathcal{A}}(1^k, V)$ as described on Figure 2. We denote by m the total number of coins withdrawn during the entire experiment. It is assumed that $\{(v_1, Z_1, \Pi_1), \ldots, (v_u, Z_u, \Pi_u)\}$ is a set of different and valid transcripts (else, we do not consider the invalid or duplicated ones when computing the sum $v = \sum v_i$). We define $\mathtt{Adv}^{tra}_{\mathcal{A}}(1^k, V)$ as $\Pr[\mathtt{Exp}^{tra}_{\mathcal{A}}(1^k, V) = 1]$. A divisible e-cash system ensures the *traceability* property if, for any probabilistic polynomial adversary $\mathcal{A}$, this advantage is negligible.

Remark 2. The E-cash systems from [CG07,ASM08,CG10,IL12] considered the *balance* property, requiring that no coalition of users can spend (and then later accepted for deposit) more than they have withdrawn, and the *identification* property, requiring that no coalition of users can double-spend a coin without revealing their identity. We argue that traceability is enough. Indeed, an adversary against the *balance* property must produce $[(v_1, Z_1, \Pi_1), \ldots, (v_u, Z_u, \Pi_u)]$ (with $\sum_{i=1}^{u} v_i > m \cdot V$) that the bank accepts as valid, not duplicated and not double-spent. This adversary can therefore be easily converted into an adversary against our traceability experiment.

Similarly, an adversary against the *identification* property must produce two valid transcripts (v_1, Z_1, Π_1) and (v_2, Z_2, Π_2) which are detected as a double-spending but such that the $\mathtt{Identify}$ algorithm does not output one name from the collusion: either this is a name outside the collusion, we deal with this case below, with the exculpability, or $\mathtt{Identify}((v_1, Z_1, \Pi_1), (v_2, Z_2, \Pi_2)) = \perp$. By legally spending all the remaining parts of this coin, one breaks the traceability property too.

Exculpability. Informally, exculpability requires that the bank, even cooperating with malicious users and merchants, cannot falsely accuse honest users of having double-spent a coin. We define the exculpability experiment $\mathtt{Exp}^{excu}_{\mathcal{A}}(1^k, V)$ as described on Figure 3. We emphasize that any adversary

$\mathtt{Exp}^{excu}_{\mathcal{A}}(1^k, V)$

1. $p.p. \leftarrow \mathtt{Setup}(1^k, V)$
2. $\mathsf{bpk} \leftarrow \mathcal{A}()$
3. $[(v_1, Z_1, \Pi_1), (v_2, Z_2, \Pi_2)] \leftarrow \mathcal{A}^{\mathcal{O}\mathtt{Add}, \mathcal{O}\mathtt{Corrupt}, \mathcal{O}\mathtt{AddCorrupt}, \mathcal{O}\mathtt{Withdraw}_{\mathcal{U}}, \mathcal{O}\mathtt{Spend}}()$
4. If $\mathtt{Identify}((v_1, Z_1, \Pi_1), (v_2, Z_2, \Pi_2), \mathsf{bpk}) = \mathsf{upk}$ and upk not corrupted, then return 1
5. Return 0

Fig. 3. Exculpability Security Game

able to spend a coin of a honest user can easily break the exculpability property, by simple making a double-spending in the name of this honest user. We define $\mathtt{Adv}^{excu}_{\mathcal{A}}(1^k, V)$ as $\Pr[\mathtt{Exp}^{excu}_{\mathcal{A}}(1^k, V) = 1]$. A divisible e-cash system is *exculpable* if, for any probabilistic polynomial adversary $\mathcal{A}$, this advantage is negligible.

4 Our Construction: Intuition

Our construction makes use of a binary tree, as in most previous works [CG07,ASM08,CG10]. The main difference is the way the tree is built. In the previous systems, each user constructs his own tree by selecting elements $k_{i,j}$ associated with the nodes of the tree and then has to get certificates on all of them (during the `Withdraw` protocol) and more importantly, must prove (either during the `Withdraw` protocol or the `Spend` one) that these elements are well-formed. This latter proof led to complex systems with either unconventional security properties [ASM08] or costly operations in groups of different orders [CG07,CG10].

In our system, there is only one tree, which is part of the public parameters. It allows us to avoid proving its well-formedness and so to achieve a better efficiency while working with zero-knowledge proofs compatible with the Groth-Sahai methodology [GS08]. In the following we first describe our `Setup` algorithm and then give a high level description of our divisible e-cash system.

4.1 Setup

Notation. Let $\mathcal{S}_n$ be the set of bitstrings of size smaller than or equal to n and $\mathcal{F}_n$ be the set of bitstrings of size exactly n. We then define, $\forall s \in \mathcal{S}_n$, the set $\mathcal{F}_n(s)$ as $\{f \in \mathcal{F}_n : s \text{ is a prefix of } f\}$. For every $s \in \mathcal{S}_n$, $|s|$ denotes the length of s.

Intuitively, since we will make use of a tree of depth n for coins of value $V = 2^n$, as illustrated on Figure 4, each node of the tree (or its path from the root) will refer to an element $s \in \mathcal{S}_n$, the root to the empty string ϵ, and each leaf to an element of $\mathcal{F}_n$. For any node $x \in \mathcal{S}_n$, $\mathcal{F}_n(s)$ contains all the leaves in the subtree below s.

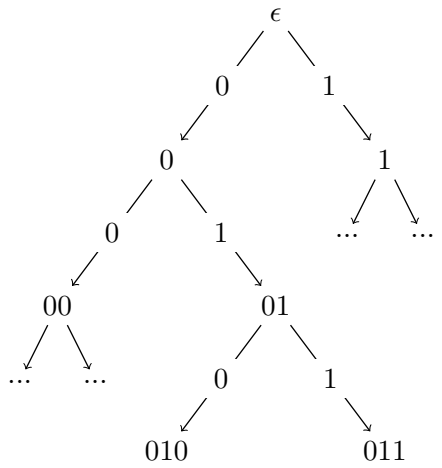

Fig. 4. Divisible coin

Public Parameters. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be the description of bilinear groups of prime order p, g, h, u_1, u_2, w (resp. $\widetilde{g}$) be generators of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$), $G = e(g, \widetilde{g})$ is thus a generator of $\mathbb{G}_T$, and $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$ be a collision-resistant hash function. In addition, a trusted authority generates

– for each $s \in \mathcal{S}_n$, $r_s \xleftarrow{\$} \mathbb{Z}_p$ and $(g_s, h_s) \leftarrow (g^{r_s}, h^{r_s})$;
– for each $f \in \mathcal{F}_n$, $l_f \xleftarrow{\$} \mathbb{Z}_p$;
– for each $s \in \mathcal{S}_n$, for each $f \in \mathcal{F}_n(s)$, $\widetilde{g}_{s\mapsto f} \leftarrow \widetilde{g}^{l_f/r_s}$.

The public parameters $p.p.$ are set as the bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, with the generators $g, h, u_1, u_2, w, \widetilde{g}$, and $\mathbb{G}$, the hash function H, as well as all the above elements $\{(g_s, h_s), s \in \mathcal{S}_n\}$ and $\{\widetilde{g}_{s\mapsto f}, s \in \mathcal{S}_n, f \in \mathcal{F}_n(s)\}$. In addition, according to the setting, either the random oracle model or the standard model, we also have

– another hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$, that will be modeled by a random oracle;
– or a CRS for the Groth-Sahai [GS08] proofs and a one-time signature scheme Σ_{ots} (such as the one from [BB08]).

It is worthy to note that users and merchants need to know the groups and the generators, but then just $\{(g_s, h_s), s \in \mathcal{S}_n\}$ (along with $\mathcal{H}$, or CRS and Σ_{ots}). The set $\{\widetilde{g}_{s\mapsto f}, (s, f) \in \mathcal{S}_n \times \mathcal{F}_n\}$ is only used by the bank, while $\{l_f, f \in \mathcal{F}_n\}$ is not useful anymore in the system.

Remark 3. An entity knowing the random scalars (r_s, l_f) used to generate the public parameters will be able to break the anonymity of our scheme. This problem already appears when generating the CRS from Groth-Sahai proofs (whose construction is not specified in [IL12]). To avoid the need of a trusted entity (although this last one would intervene only during the `Setup` phase) the public parameters can be cooperatively generated by the bank and a set of users. For example,

– one party can first generate $a_s, c_f \xleftarrow{\$} \mathbb{Z}_p$ for all $s \in \mathcal{S}_n$ and $f \in \mathcal{F}_n$, compute $A_s \leftarrow g^{a_s}$ and $\widetilde{A}_{s\mapsto f} \leftarrow \widetilde{g}^{c_f/a_s}$, and prove knowledge of a_s and c_f;
– the second party will then select random $b_s, d_f \xleftarrow{\$} \mathbb{Z}_p$, compute $B_s \leftarrow A_s^{b_s}$ and $\widetilde{B}_{s\mapsto f} \leftarrow \widetilde{A}_s^{d_f/b_s}$, and prove knowledge of b_s and d_f.

If the proofs are valid then $g_s \leftarrow B_s$ and $\widetilde{g}_{s\mapsto f} \leftarrow \widetilde{B}_{s\mapsto f}$.

4.2 High Level Description

The main challenge when designing a divisible e-cash system is to describe an efficient way for the bank to recover all the serial numbers of the spent coins without endangering the anonymity of the honest users. For the sake of clarity, we describe our construction in three stages.

– In the first one, we describe a system fulfilling a weak anonymity property (called *unlinkability*, see Remark 1, in Section 3.2) meaning that no one can decide, given two transcripts of the `Spend` protocol, whether they were produce by using the same global coin but where the spent part (*i.e.* the node) of the coin can be revealed;
– In the second stage, we show how to increase the level of anonymity, reaching a stronger version of unlinkability (that we call *strong unlinkability*) meaning that it is now impossible to decide which node was spent. However, the level of the node is inevitably revealed since it corresponds to the amount paid by the user.
– Eventually, it remains to explain how to recover the identity of a double-spender without the help of a trusted entity. This is done in our third stage where we describe the construction of a security tag which, added to the scheme from the second stage, leads to an anonymous divisible e-cash system in the sense of the definition given in Section 3.2.

All our three stages use the same idea: each divisible coin is associated with a secret scalar $x \xleftarrow{\$} \mathbb{Z}_p$, known to the user only, and certified by the bank during the `Withdraw` protocol.

Unlinkability. To spend a value of $v = 2^\ell$, the user first selects an unspent node s at level $n - \ell$, computes $t_s \leftarrow g_s^x$ and proves in a zero-knowledge way that t_s is well formed and that x is certified. This can be efficiently performed in either model: the random oracle model or standard model.

Since we consider here the *unlinkability* property, the user can reveal the chosen s so that $\mathcal{B}$ can compute, for each $f \in \mathcal{F}_n(s)$, the 2^ℓ elements $S_f(x) \leftarrow e(t_s, \widetilde{g}_{s\mapsto f}) = e(g_s, \widetilde{g}_{s\mapsto f})^x = G^{x \cdot l_f}$ which actually are the 2^ℓ serial numbers. Indeed, these elements only depend on (i) the spent leaves and (ii) the secret of the coin, which makes the bank able to detect double-spending (*i.e.* two identical spent leaves, which equivalently means that two nodes s and s' are such that one of them is the prefix of the other): the bank simply checks whether for some f, there are s and s' such that $e(t_s, \widetilde{g}_{s\mapsto f}) = G^{x \cdot l_f} = e(t_{s'}, \widetilde{g}_{s'\mapsto f})$.

For honest users, the unlinkability property follows from the fact that it is hard, knowing g_s, $g_{s'}$ and g_s^x, to decide whether an element of $\mathbb{G}_1$ is random or equal to $g_{s'}^x$ under the XDH assumption.

Strong Unlinkability. We now want to leak less information about the node s: actually, just its level can be leaked, since it corresponds to the value of the transaction. To address this issue, the bank will now certify every element g_s, using a different key for each level of the tree (and so according to the value), and publish all certificates. To prove that t_s is well-formed, the user now has to prove that he used a certified value g_s of the appropriate level, which is still possible in either the random oracle model or the standard model, with a slight increase of the size of the proof. Since the bank does not know the exact node s, but only its level $n - \ell$, given t_s, it now needs to compute and stores all the elements $e(t_s, \widetilde{g}_{s'\mapsto f})$ for every leaf f and every node s' of the same level $n - \ell$. Of course, some of these elements (and maybe most of them) do not correspond to any valid serial number, but the point is that all the 2^ℓ valid serial numbers will be among them, making the bank able to detect double spendings.

Remark 4. One has to take care of additional false positive cases for a leaf f: for two distinct coins whose associated secrets are x_1 and x_2 respectively ($x_1 \neq x_2$), there exist four nodes $r_{s_1}, r_{s'_1}$ and $r_{s'_2}, r_{s'_2}$ such that $e(g^{x_1 r_{s_1}}, \widetilde{g}^{y_f / r_{s'_1}}) = e(g^{x_2 r_{s_2}}, \widetilde{g}^{y_f / r_{s'_2}})$, and thus $x_1 r_{s_1} r_{s'_2} = x_2 r_{s'_1} r_{s_2}$. For randomly chosen x's, this happens with negligible probability.

Anonymity. Once a double-spending is detected, the procedure for recovering the user's identity depends on the kind of opening we want. In the case of fair e-cash systems, an opening authority uses, as for group signatures schemes, the knowledge of some trapdoor to recover the identity of the user from any valid transaction. Such system can be used in association with the above strongly unlinkable solution to provide identification of the double spender. However, to reach the true anonymity property we must avoid such a powerful authority and then allow anyone to recover this identity from any double-spent coins, and only in case of fraud.

Let $\mathsf{usk} \in \mathbb{Z}_p$ be the secret key of a user and $\mathsf{upk} \leftarrow g^{\mathsf{usk}}$ his public key. When spending a node s, each user will also compute and send a security tag $v_s \leftarrow \mathsf{upk}^r \cdot h_s^x$, where r is deterministically obtained by hashing some public information *info* related to the transaction (amount, date, public key of the merchant, etc). Of course, it will also have to prove that this tag is well formed (x and upk are certified and h_s corresponds to the same node as g_s).

If the bank detects a double-spending, it means that there are two transcripts containing (t_s, v_s) and $(t_{s'}, v_{s'})$ such that there exists $f \in \mathcal{F}_n$ which is a descendant of both s and s'. Therefore, we have both $e(t_s, \widetilde{g}_{s\mapsto f}) = e(t_{s'}, \widetilde{g}_{s'\mapsto f}) = G^{x \cdot l_f}$ and $e(h_s, \widetilde{g}_{s\mapsto f}) = e(h_{s'}, \widetilde{g}_{s'\mapsto f}) = e(h, \widetilde{g})^{l_f}$. Anyone can then compute, from the involved transcripts and the public parameters, $T \leftarrow e(v_s, \widetilde{g}_{s\mapsto f})$ and $T' \leftarrow e(v_{s'}, \widetilde{g}_{s'\mapsto f})$. Using the bilinearity of the pairing we get:

$$\begin{aligned} T \cdot T'^{-1} &= e(\mathsf{upk}^r, \widetilde{g}_{s\mapsto f}) \cdot e(h_s^x, \widetilde{g}_{s\mapsto f}) \cdot e(\mathsf{upk}^{-r'}, \widetilde{g}_{s'\mapsto f}) \cdot e(h_{s'}^x, \widetilde{g}_{s'\mapsto f})^{-1} \\ &= e(\mathsf{upk}, \widetilde{g}_{s\mapsto f}^r \cdot \widetilde{g}_{s'\mapsto f}^{-r'}). \end{aligned}$$

It remains to check, for each registered upk_i, whether $T \cdot T'^{-1} = e(\mathsf{upk}_i, \widetilde{g}^{r}_{s \mapsto f} \cdot \widetilde{g}^{-r'}_{s' \mapsto f})$. We recall that r and r' are scalar deterministically computed from the transaction information *info*, and thus publicly computable.

The `Identify` algorithm thus has a linear cost in the number of registered users, but we can argue that this is not a major drawback because it will be run *offline* and double-spending should not occur too often: the possibility of tracing back defrauders is an incentive not to try to cheat. Note that this algorithm does not make use of any private information, it can thus be run in parallel by many users, as for the public traceability in broadcast encryption [CPP05].

5 Our Divisible E-Cash System

5.1 The protocol

In this section, we focus on the anonymous version of our solution. We then describe all the algorithms in more details, except the `Setup` one, already fully described above. Our `Spend` protocol will make use of non-interactive zero-knowledge (NIZK) proofs which can be provided either in the random oracle model (using the Fiat-Shamir heuristic [FS86]) or in the standard model (using the Groth-Sahai proof systems [GS08], since we are in a bilinear setting). Even if the frameworks are similar, some algorithms differ according to the model. We provide in Appendix A some instantiations of our protocol.

- `BKeygen`(): Upon receiving the public parameters, the bank will select two different signatures schemes:
  - $\Sigma_0 = (\mathtt{Keygen}, \mathtt{Sign}, \mathtt{Verify})$, whose message space is $\mathbb{G}_1^2$, to sign some elements of the public parameters. We can use the structure preserving construction from [AGHO11]. But we stress that we do not need the EUF-CMA security. A signature scheme secure against selective chosen-message attacks would be enough.
  - $\Sigma_1 = (\mathtt{Keygen}, \mathtt{Sign}, \mathtt{Verify})$, whose message space depends on the security model.
    - **ROM:** The message space is $\mathbb{Z}_p^2$. But we additionally require that Σ_1 is compatible with a protocol $\Sigma_1.\mathtt{SignCommit}$ which, given (u_1^x, u_2^y) for some $(x, y) \in \mathbb{Z}_p^2$ (and so a kind of commitment of (x, y)), outputs a valid signature σ on (x, y) (we can then use the scheme from [CL04] or a variant of [BB08]).
    - **Standard Model:** The message space is $\mathbb{G}_1^2$, and we can then use again the scheme from [AGHO11].

  The bank will then get $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \Sigma_1.\mathtt{Keygen}(p.p.)$ and $(\mathsf{sk}_0^{(i)}, \mathsf{pk}_0^{(i)}) \leftarrow \Sigma_0.\mathtt{Keygen}(p.p.)$ for each level of the tree, $0 \leq i \leq n$, and compute, for every $s \in \mathcal{S}_n$, $\tau_s \leftarrow \Sigma_0.\mathtt{Sign}(\mathsf{sk}_0^{(|s|)}, (g_s, h_s))$. Eventually, it will set bsk as sk_1 and bpk as $(\{\mathsf{pk}_0^{(i)}\}_i, \mathsf{pk}_1, \{\tau_s\}_{s \in \mathcal{S}_n})$.
- `Keygen`(): Each user (resp. merchant) selects a random $\mathsf{usk} \leftarrow \mathbb{Z}_p$ (resp. msk) and gets $\mathsf{upk} \leftarrow g^{\mathsf{usk}}$ (resp. $\mathsf{mpk} \leftarrow g^{\mathsf{msk}}$). In the following we assume that upk (resp. mpk) is public, meaning that anyone can get an authentic copy of it.
- `Withdraw`($\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk})$): To withdraw a divisible coin, the user first selects a random $x \in \mathbb{Z}_p$ and computes u_1^{usk} and u_2^x. He then sends $\mathsf{upk}, u_1^{\mathsf{usk}}, u_2^x$ and proves, using for example the Schnorr's interactive protocol [Sch89], knowledge of x and usk. If the proof is valid and if u_2^x was not previously used, the bank
  - **ROM:** runs the $\Sigma_1.\mathtt{SignCommit}$ on $(u_1^{\mathsf{usk}}, u_2^x)$ and sends the resulting signature σ to the user who sets $C \leftarrow (x, \sigma)$.
  - **Standard Model:** computes $\sigma \leftarrow \Sigma_1.\mathtt{Sign}(\mathsf{sk}_1, (u_1^{\mathsf{usk}}, u_2^x))$ and sends it to the user who sets $C \leftarrow (x, \sigma)$.
- `Spend`($\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, \mathsf{mpk}, 2^\ell), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, 2^\ell)$): To spend a value 2^ℓ, the user selects an unspent node s of level $n - \ell$ and computes $r \leftarrow H(info)$ and $(t_s, v_s) \leftarrow (g_s^x, \mathsf{upk}^r \cdot h_s^x)$. He must then prove that t_s and v_s are well-formed, *i.e.* that he used values certified during a withdrawal, hence a proof of knowledge of σ, and that he used a valid pair (g_s, h_s), hence a proof of existence of τ_s. The protocol in the ROM differs from the one in the standard model.

- **ROM:** The user provides a zero-knowledge proof of knowledge of usk, x, g_s, h_s, τ_s, and σ such that:

$$t_s = g_s^x \wedge v_s = (g^r)^{\mathsf{usk}} \cdot h_s^x \quad \wedge \quad \Sigma_1.\mathtt{Verify}(\mathsf{pk}_1, (\mathsf{usk}, x), \sigma) = 1$$
$$\wedge \quad \Sigma_0.\mathtt{Verify}(\mathsf{pk}_0^{(n-\ell)}, (g_s, h_s), \tau_s) = 1.$$

  Using appropriate signature schemes, as shown in Appendix A, such zero-knowledge proofs of knowledge 'à la Schnorr' can be done. The global proof is then converted into a signature of knowledge Π on the message r, using the Fiat-Shamir heuristic [FS86].
- **Standard Model:** The user first generates a new key pair $(\mathsf{sk}_{ots}, \mathsf{pk}_{ots}) \leftarrow \Sigma_{ots}.\mathtt{Keygen}(1^k)$ and computes $\mu \leftarrow w^{\frac{1}{\mathsf{usk}+H(\mathsf{pk}_{ots})}}$. He then computes Groth-Sahai commitments to usk, x, g_s, h_s, τ_s, σ, μ, $U_1 = u_1^{\mathsf{usk}}$, $U_2 = u_2^x$ and provides a NIZK proof π that the committed values satisfy:

$$t_s = g_s^x \wedge v_s = (g^r)^{\mathsf{usk}} \cdot h_s^x \wedge U_2 = u_2^x \wedge U_1 = u_1^{\mathsf{usk}} \wedge \mu^{(\mathsf{usk}+H(\mathsf{pk}_{ots}))} = w$$

  along with a NIWI proof π' that the committed values satisfy:

$$1 = \Sigma_0.\mathtt{Verify}(\mathsf{pk}_0^{(n-\ell)}, (g_s, h_s), \tau_s) \wedge 1 = \Sigma_1.\mathtt{Verify}(\mathsf{pk}_1, (U_1, U_2), \sigma)$$

  Again, using appropriate signature schemes, as shown in Appendix A, the Groth-Sahai methodology [GS08] can be used. Finally, the user computes $\eta \leftarrow \Sigma_{ots}.\mathtt{Sign}(\mathsf{sk}_{ots}, H(t_s||v_s||\pi||\pi'||r))$ and sends it to $\mathcal{M}$ along with $t_s, v_s, \mathsf{pk}_{ots}, \pi, \pi'$.

  In both cases, the merchant checks the validity of the proofs and of the signatures and accepts the transaction if everything is correct. In such a case, he stores $Z \leftarrow (t_s, v_s)$ and either the signature of knowledge Π in the ROM or $\Pi \leftarrow (\pi, \pi', \mathsf{pk}_{ots}, \eta)$ in the standard model.

– $\mathtt{Deposit}(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (2^\ell, Z, \Pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: Upon receiving a transcript, the bank will check that it was not previously deposited and verify its validity. Then, for each s' of level $n - \ell$ and $f \in \mathcal{F}_n(s')$ it will compute $z_i \leftarrow e(t_s, \widetilde{g}_{s' \mapsto f})$ and check whether $z_i \in L$. If $\forall i, z_i \notin L$ then the bank will add these elements to this list (see Remark 5) and store the transcript $(2^\ell, Z, \Pi)$. Else, there is an element $z' \in L$ such that $z_i = z'$. The bank will recover the corresponding transcript $(2^{\ell'}, Z', \Pi')$ and output $[(2^\ell, Z, \Pi), (2^{\ell'}, Z', \Pi')]$.

– $\mathtt{Identify}((2^{\ell_1}, Z_1, \Pi_1), (2^{\ell_2}, Z_2, \Pi_2), \mathsf{bpk})$: To recover the identity of a double-spender, the entity running this algorithm will first check the validity of both transcripts and return $\perp$ if one of them is not correct. He then computes, for $i \in \{1, 2\}$ and for every leaf f, the lists $S_{i,f} \leftarrow \{e(t_{s_i}, \widetilde{g}_{s \mapsto f}),$ where $s \in \mathcal{S}_n$ is the prefix of length $|s_i|$ of $f\}$, and returns $\perp$ if there is no collision between $S_{1,f}$ and $S_{2,f}$ for any leaf f. Else, we can assume (see Remark 4) that we have $e(t_{s_1}, \widetilde{g}_{s_1 \mapsto f}) = e(t_{s_2}, \widetilde{g}_{s_2 \mapsto f})$ with $t_{s_1} = g_{s_1}^x$ and $t_{s_2} = g_{s_2}^x$ for some $s_1, s_2 \in \mathcal{S}_n$. As explained in section 4.2, $e(v_{s_1}, \widetilde{g}_{s_1 \mapsto f}) \cdot e(v_{s_2}, \widetilde{g}_{s_2 \mapsto f})^{-1} = e(\mathsf{upk}, \widetilde{g}_{s \mapsto f}^r \cdot \widetilde{g}_{s' \mapsto f}^{-r'})$ so it remains to compute $e(\mathsf{upk}_i, \widetilde{g}_{s \mapsto f}^r \cdot \widetilde{g}_{s' \mapsto f}^{-r'})$ for each public key upk_i until we get a match, in which case one outputs upk_i.

Remark 5. Since the node used to spend a coin C is not known, the bank has to store 2^n elements z_i each time a transcript is deposited by a merchant, even if the amount deposited is 2^ℓ with $\ell \leq n$: one for each leaf f. In the worst case scenario (if the user only spends values of 1), a divisible coin of 2^n will require that the bank stores 2^{2n} elements. However, the bank does not need to store the elements $z_i \in \mathbb{G}_T$, it may only store $H'(z_i)$ (for some hash function H') and so compare the (smaller) hash values. If a collision is found, the bank will first recover and compare the elements z_i from the involved transcripts to ensure that this collision was not due to the function H' (which would anyway be quite unlikely). Hash tables or Bloom filters can be used to improve on the storage efficiency.

Even if the constructions in both models are quite similar, there are some necessary differences, especially in the `Spend` protocol. Our results concerning the security of our scheme will then also differ according to each model. The security proofs of the following theorems are provided in Section 6.

Theorem 6. *In the random oracle model, assuming that the hash function H is collision-resistant, our divisible e-cash system is anonymous under the* XDH *assumption, traceable if Σ_0 is secure against selective chosen-message attacks and Σ_1 is an EUF-CMA secure signature scheme, and achieves the exculpability property under the* DL *assumption.*

Theorem 7. *In the standard model, assuming that the hash function H is collision-resistant, our divisible e-cash system is anonymous under the* XDH *assumption, traceable if Σ_0 is secure against selective chosen-message attacks and Σ_1 is an EUF-CMA secure signature scheme, and achieves the exculpability property under the $q-$* SDH *assumption if Σ_{ots} is a strong one-time signature scheme.*

5.2 Efficiency

We compare in Figure 5 the efficiency of our construction with the one of [CG10], which is the most efficient protocol in the ROM, and the one of [IL12], which is the only construction in the standard model.

Schemes	Canard-Gouget [CG10]	Izabachène-Libert [IL12]	Our work
Standard Model	no	yes	yes
Public Parameters	$2^{n+3}\|q\| + 1\ pk$	$2\ \mathbb{G}_1 + 1\ \mathbb{G}_2 + 1\ pk$	$(n+2)\ pk + 1\ \mathbb{G}_2$ $+ (2^{n+2}+3)\ \mathbb{G}_1$ $+ (2^{n+1}-1)\ \|\texttt{Sign}\|$
`Withdraw` Computations	$(2^{n+3}+2^{n+2}-5)\texttt{exp}$ $+ (n+2)\ \texttt{Sign}$	$1\ \texttt{Sign}$	$1\ \texttt{Sign}$
Coin Size	$(2^{n+2}+n+1)\ \|q\|$ $+ (n+2)\ \|\texttt{Sign}\|$	$3\ \|p\| + \|\texttt{Sign}\|$	$2\ \|p\| + \|\texttt{Sign}\|$
`Spend` Computations	$NIZK\{\ 3\ \texttt{exp}^*$ $+ 2\ \texttt{Sign} + 2\ \texttt{Pair}\ \}$ $+ 1\ \texttt{exp}$	$NIZK\{\ (n-l)\ \texttt{exp}$ $+ (7(n-l)+6)\ \texttt{Pair}$ $+ 1\ \texttt{Sign}\ \}$ $+ (8(n-l)+4)\ \texttt{exp}$	$NIZK\{\ 2\ \texttt{exp}$ $+ 2\ \texttt{Sign}\ \} + 3\ \texttt{exp}$ $+ 1\ \texttt{Sign}$
Transfer size of `Spend`	$3\ \|q\| + \|NIZK\|$	$3(n-l)\ \mathbb{G}_2 +$ $3(n-l)\mathbb{G}_T + \|NIZK\|$	$3\ \mathbb{G}_1 + 1\ \|\texttt{Sign}\|$ $+\|NIZK\|$
`Deposit` Computations	$2^{l+1}\texttt{exp}$	unbounded	$2^n\ \texttt{Pair}$
`Deposit` size	$2^l\ \|q\| + \|\texttt{Spend}\|$	$\|\texttt{Spend}\|$	$2^n\ \mathbb{G}_T + \|\texttt{Spend}\|$

Fig. 5. Efficiency comparison between related works and our construction for coins of value 2^n and `Spend` and `Deposit` of value 2^l. The space and times complexities are given from the user's point of view. `exp` refers to an exponentiation, `pair` to a pairing computation, `Sign` to the cost of the signature issuing protocol, from the user point of view, whose public key is pk. $NIZK\{\texttt{exp}\}$ denotes the cost of a $NIZK$ proof of a multi-exponentiation equation, $NIZK\{\texttt{pair}\}$ the one of a pairing product equation and $NIZK\{\texttt{Sign}\}$ the one of a valid signature. $NIZK\{\texttt{exp}^*\}$ refers to the cost of a proof of equality of discrete logarithms in groups of different orders.

The different settings of these constructions make this comparison difficult but this table can still be useful to compare the asymptotic complexity of the different algorithms. We refer to Appendix A for instantiations of our construction. One may note some differences between our table and the one provided in [CG10]. They are due to the fact that the authors of [CG10] denote the computations of accumulators by `Acc`. Since these accumulators can store up to 2^{n+2} elements their computations actually involve up to 2^{n+2} exponentiations (during the creation of these accumulators inside the withdrawal protocol) while their definitions significantly increase the size of the public parameters (hence the 2^{n+3} elements).

The scheme from [CG10] uses subgroups of $\mathbb{Z}_{r_i}$ for some primes r_i and bilinear groups of similar orders for their accumulators. Assuming that the parameters of the accumulators (which are elliptic curve points) are provided in a compressed way (*i.e.* only one coordinate by point) and that q is an approximation of the orders of the different groups, we will consider that each element involved in their protocol has a $|q|$-bit representation. The scheme from [IL12] and the one described in this paper

use bilinear groups of prime order p. For a 128-bits security level, we have $|p| = 256$, $|\mathbb{G}_1| = 256$ and $|\mathbb{G}_2| = 512$ by using Barreto-Naehrig curves [BN05] whereas $|q|$ must be greater than 3072 for [CG10] (see [GPS08]).

Public Parameters. A downside of our protocol is the size of the public parameters. However, it is worthy to note that by using the curve parameters from [BN05] and the structure preserving signature scheme from [AGHO11], the storage space required by these parameters for $n = 10$ (enabling users to divide their coin in 1024 parts) is 330 KBytes which can easily be handled by any smartphone. Our parameters (see Appendix A) require then less storage space than the ones of [CG10] (since $2^{10+3} \cdot |q| = 3.1$ MBytes). For the bank, the additional cost of storing the elements $\{\tilde{g}_{s \mapsto f}\}$ is only 721 KBytes.

Withdrawal and Spending. The strong time constraints of electronic transactions require very efficient withdrawal and spending protocols. Compared to any paper in the literature with similar security level (especially [CG10] and [IL12]), our protocol is the only one to achieve constant time for both the `Withdraw` and the `Spend` protocols. Moreover, even if the `Spend` protocol from [CG10] can be performed in constant time, it involves zero-knowledge proofs in groups of different orders which are rather complex, even in the ROM.

Deposit. Unfortunately, our `Deposit` protocol involves up to 2^n pairings and so is less efficient than the one from [CG10]. For $n = 10$ it means that the bank must compute 1024 pairings. Even if they can be computed in parallel and even if each of them can be performed on a computer in less than 1 ms [BGM+10], the computational cost is significant. However, since this protocol is run offline (without the strong time constraints of the previous protocols) and since the computational power of a bank can be assumed to be far more important than the one of a computer, we argue that the cost for the bank remains reasonable. Regarding the storage size, the bank must store 2^n serial numbers by transaction. As explained in Remark 5, the bank does not need to store the elements z_i but only their hash values $H'(z_i)$ for some hash function H' whose output space can be rather small since, in the event of a collision, the bank will first recompute the elements z_i before running the `Identify` algorithm. For example, considering that the output space of H' has a 80 bits size, the space required to store the serial numbers of one million transactions is about 10 GBytes, which is still practical for the bank.

Finally, we stress that our `Deposit` protocol is the first one in the standard model with a bounded computational cost, *i.e.* which does not depend on the number of previous transactions, as in [IL12] (excepted for the lookup in tables for double-spending detection).

6 Security Proofs

The proofs of anonymity and traceability are similar in the ROM and in the standard model so we only describe one proof for both models. This is no longer the case for the exculpability property which requires two different proofs.

6.1 Proof of Anonymity

Let $\mathcal{A}$ be an adversary against the anonymity with advantage ϵ. We construct a reduction $\mathcal{R}$ using $\mathcal{A}$ against XDH challenges in $\mathbb{G}_1$. Let (g, g^x, g^y, g^z) be a XDH-challenge in $\mathbb{G}_1$, $\mathcal{R}$ randomly selects $f^* \in \mathcal{F}_n$ and generates the public parameters as follows.

- $(h, u_1, u_2) \leftarrow (g^c, g^{d_1}, g^{d_2})$ for some $c, d_1, d_2 \xleftarrow{\$} \mathbb{Z}_p$
- For each $f \in \mathcal{F}_n$, $l_f \xleftarrow{\$} \mathbb{Z}_p$
- For each $s \in \mathcal{S}_n$, $r_s \xleftarrow{\$} \mathbb{Z}_p$

– For each $s \in \mathcal{S}_n$:
  • If s is a prefix of f^* then $g_s \leftarrow (g^y)^{r_s}$
  • Else $g_s \leftarrow g^{r_s}$
  • $h_s \leftarrow g_s^c$
– For each $s \in \mathcal{S}_n$, for each $f \in \mathcal{F}_n(s)$, output $\widetilde{g}_{s \mapsto f} \leftarrow \widetilde{g}^{\frac{l_f}{r_s}}$.

In this way, only the prefixes of f^* will involve challenge elements. In the standard model, $\mathcal{R}$ also generates a simulated common reference string. Let q_w be a bound on the number of $\mathcal{O}\texttt{Withdraw}$ queries, $\mathcal{R}$ randomly selects i^* from $[0,q_w]$ and answers to the oracle queries as follows:

– $\mathcal{O}\texttt{Add}()$ queries: $\mathcal{R}$ runs the $\texttt{Keygen}$ algorithm and returns upk (or mpk).
– $\mathcal{O}\texttt{Withdraw}_{\mathcal{U}}(\mathsf{bsk}, \mathsf{upk})$ queries: When the adversary makes the i^{th} query to the $\mathcal{O}\texttt{Withdraw}_{\mathcal{U}}$ oracle, the reduction acts normally if $i \neq i^*$ and as if the secret value of the coin is x otherwise (by sending $(g^x)^{d_2}$ and simulating the proof of knowledge, since x is not known by $\mathcal{R}$). The chosen public key corresponding to this last case will be denoted upk^*.
– $\mathcal{O}\texttt{Corrupt}(\mathsf{upk}/\mathsf{mpk})$ queries: $\mathcal{R}$ acts normally if the query was not made on upk^*. Else, it aborts the experiment.
– $\mathcal{O}\texttt{AddCorrupt}(\mathsf{upk}/\mathsf{mpk})$: $\mathcal{R}$ stores the public key which is now considered as registered.
– $\mathcal{O}\texttt{Spend}(\mathsf{upk}, 2^\ell)$ queries: $\mathcal{R}$ is able to deal with any of these queries if $\mathsf{upk} \neq \mathsf{upk}^*$. Else, the reduction is able to answer as long as $c_{\mathsf{upk}} < m_{\mathsf{upk}} \cdot 2^n - 2^\ell$ (and aborts otherwise) since this condition means that there is at least one unspent node s which is not the prefix of f^*. The reduction can then compute a valid pair $(t_s, v_s) \leftarrow ((g^x)^{r_s}, \mathsf{upk}^r \cdot t_s^c)$ where $r \leftarrow H(info)$ and simulates the non-interactive proof (which is possible even in the standard model since we use a simulated CRS).

During the challenge phase, $\mathcal{A}$ outputs $\{\mathsf{upk}_0, \mathsf{upk}_1\}$ along with a value 2^ℓ. Of course, it is assumed that none of these users has spent more than $m_{\mathsf{upk}_b} \cdot 2^n - 2^\ell$. If $\mathsf{upk}^* \notin \{\mathsf{upk}_0, \mathsf{upk}_1\}$ then $\mathcal{R}$ aborts, else it selects the prefix s^* of length $n - \ell$ of f^*, which cannot have been spent, by the assumption made on the $\mathcal{O}\texttt{Spend}$ queries. $\mathcal{R}$ also provides a simulated proof and then answers the oracle queries as previously. Since $g_{s^*} = (g^y)^{r_{s^*}}$, the reduction returns $(t_{s^*}, v_{s^*}) \leftarrow ((g^z)^{r_{s^*}}, (\mathsf{upk}^*)^r \cdot t_{s^*}^c)$, which is valid for upk^* iff $z = x \cdot y$. $\mathcal{R}$ returns a random element from $\mathbb{G}_1^2$ if $z \neq x \cdot y$. Then, $\mathcal{R}$ uses the bit returned by $\mathcal{A}$ to solve the XDH challenge.

When $\mathcal{A}$ selects the users involved in the challenge phase, it actually selects the two subsets S_0 and S_1 of the withdrawn coins belonging to these users. The condition on the challenge phase implies that there is at least one coin in each subset which has not been totally spent. If the coin withdrawn during the i^{*th} query is one of them, $\mathcal{R}$ will not abort. Its probability of success in breaking the XDH-assumption is then greater than $2\epsilon/q_w$.

6.2 Proof of Traceability

Let $\mathcal{A}$ be an adversary against the traceability. We construct a reduction $\mathcal{R}$ using $\mathcal{A}$ against the unforgeabiliy of Σ_0 or Σ_1. $\mathcal{R}$ generates the public parameters as in the $\texttt{Setup}$ algorithm and selects $0 \leq i^* \leq n$. It then generates n keys pairs $(sk_0^{(i)}, pk_0^{(i)}) \leftarrow \Sigma_0.\texttt{Keygen}(1^k)$ for $1 \leq i \neq i^* \leq n$ and uses $sk_0^{(i)}$ to sign (g_s, h_s) such that $|s| = i$. Finally, it sends (g_s, h_s) for every $s \in \mathcal{S}_n$ such that $|s| = i^*$ to the $\Sigma_0.\texttt{Sign}$ oracle which returns the signatures τ_s along with the verification key $pk_0^{(i^*)}$. $\mathcal{R}$ also receives the public key pk_1 from the challenger of the experiment of the EUF-CMA security of Σ_1 and sets its public key as $(\{pk_0^{(j)}\}_j, pk_1, \{\tau_s\}_{s \in \mathcal{S}_n})$. The reduction will proceed as usual when it receives $\mathcal{O}\texttt{Add}$, $\mathcal{O}\texttt{Corrupt}$, $\mathcal{O}\texttt{AddCorrupt}$ and $\mathcal{O}\texttt{Spend}$ queries and uses its $\Sigma_1.\texttt{Sign}$ oracle to answer $\mathcal{O}\texttt{Withdraw}_{\mathcal{B}}$ queries.

Let q_w be the number of withdrawn queries. In order to succeed, $\mathcal{A}$ must output u valid transcripts $(2^{\ell_j}, Z_j, \Pi_j)$ such that $\sum 2^{\ell_j} > q_w \cdot 2^n$ and such that $\texttt{Identify}((2^{\ell_i}, Z_i, \Pi_i), (2^{\ell_j}, Z_j, \Pi_j)) = \perp$ for every $1 \leq i \neq j \leq n$. The perfect soundness of the proof implies that each transcript $(2^{\ell_j}, Z_j, \Pi_j)$ involves

a pair (g_j, h_j) and a signature τ_j such that $\Sigma_0.\texttt{Verify}((g_j, h_j), \tau_j, pk_0^{(n-l_j)}) = 1$. We may then assume that $(g_j, h_j) = (g_{s_j}, h_{s_j})$ for some $s_j \in \mathcal{S}_n$ such that $|s_j| = n - l_j$. Else, $((g_j, h_j), \tau_j)$ is a valid forgery which breaks the security of Σ_0 with probability $\frac{1}{n+1}$ (*i.e.* if $i^* = n - l_j$).

Let $x_1, ..., x_{q_w}$ be the q_w secret values (one for each withdrawn coin). Since an amount of $\sum 2^{\ell_j} > q_w \cdot 2^n$ has been deposited, the bank has computed $\sum 2^{\ell_j}$ elements $z_i \leftarrow e(t_{s_j}, \widetilde{g}_{s_j \mapsto f})$. If $\{z_i\}_i \subset \{e(g, \widetilde{g})^{l_f \cdot x_i}\}_{f \in \mathcal{F}_n, 1 \leq i \leq q_w}$ then there is at least one couple (i, j) such that $i \neq j$ and $z_i = z_j$, because the size of the last set is $q_w \cdot 2^n$. Such a collision implies (see remark 4) that the security tags v_{s_i} and v_{s_j} have been produced with the same secret x and so with the same public key upk which would have been returned by the $\texttt{Identify}$ algorithm. We can therefore assume that $\{z_i\}_i \nsubseteq \{e(g, \widetilde{g})^{l_f \cdot x_i}\}_{f \in \mathcal{F}_n, 1 \leq i \leq q_w}$, implying that at least one of the element t_{s_j} is equal to $g_{s_j}^x$ for some $x \notin \{x_1, ..., x_{q_w}\}$. We can then extract, from the corresponding spending, a valid forgery σ on (usk, x) in the ROM and on $(u_1^{\mathsf{usk}}, u_2^x)$ in the standard model and so breaks the security of Σ_1.

6.3 Proof of Exculpability

We distinguish the proof in the ROM from the one in the standard model.

ROM: Let $\mathcal{A}$ be an adversary against the exculpability property. We construct a reduction $\mathcal{R}$ using $\mathcal{A}$ against the DL challenges in $\mathbb{G}_1$. Let (g, g^α) be a DL challenge, $\mathcal{R}$ generates the public parameters as in the $\texttt{Setup}$ algorithm and selects $1 \leq i^* \leq q_a$ where q_a is a bound on the number of $\mathcal{O}\texttt{Add}$ queries. $\mathcal{R}$ will answer the oracle queries as follows.

- $\mathcal{O}\texttt{Add}()$ queries: When the adversary makes the i-th $\mathcal{O}\texttt{Add}$ query to register a user, $\mathcal{R}$ will run the $\texttt{Keygen}$ algorithm if $i \neq i^*$ and set $\mathsf{upk}^* \leftarrow g^\alpha$ otherwise.
- $\mathcal{O}\texttt{Corrupt}(\mathsf{upk}/\mathsf{mpk})$ queries: $\mathcal{R}$ returns the secret key if $\mathsf{upk} \neq \mathsf{upk}^*$ and aborts otherwise.
- $\mathcal{O}\texttt{AddCorrupt}(\mathsf{upk}/\mathsf{mpk})$ queries: $\mathcal{R}$ stores the public key which is now considered as registered.
- $\mathcal{O}\texttt{Withdraw}_{\mathcal{U}}(\mathsf{bsk}, \mathsf{upk})$ queries: $\mathcal{R}$ acts normally if $\mathsf{upk} \neq \mathsf{upk}^*$ and simulates the interactive proof of knowledge of α otherwise.
- $\mathcal{O}\texttt{Spend}(\mathsf{upk}, 2^\ell)$ queries: $\mathcal{R}$ acts normally if $\mathsf{upk} \neq \mathsf{upk}^*$ and simulates the non-interactive proof of knowledge of α otherwise.

The adversary then outputs two valid transcripts $(2^{\ell_1}, Z_1, \Pi_1)$ and $(2^{\ell_2}, Z_2, \Pi_2)$ which accuse upk of double-spending. If $\mathsf{upk} \neq \mathsf{upk}^*$ then $\mathcal{R}$ aborts. Else, at least one of this transcript was not produced by $\mathcal{R}$ (else it would have double-spent its own coins). The soundness of the signature of knowledge implies then that we can extract α from this forged transcript. $\mathcal{R}$ is then able to solve the discrete logarithm problem in $\mathbb{G}_1$ since it will not abort with probability $1/q_a$.

Standard Model: An adversary $\mathcal{A}$ against the exculpability property outputs two transcripts accusing an honest user upk of double-spending. As explained above, at least one of these transcripts was not produced by $\mathcal{R}$. Let pk'_{ots} be the one-time signature key used in this forged transcript, there are two kinds of attacks that can be mounted by $\mathcal{A}$:

- Type-1 Attack: pk'_{ots} is one of the key used by $\mathcal{R}$ to answer a $\mathcal{O}\texttt{Spend}$ query.
- Type-2 Attack: pk'_{ots} was not used by $\mathcal{R}$ to answer $\mathcal{O}\texttt{Spend}$ queries.

Clearly, an adversary succeeding in a Type-1 attack with non-negligible probability can be used to break the security of the one-time signature scheme Σ_{ots}. We therefore only consider Type-2 attacks in what follows.
Let $(g, g^\alpha, ..., g^{\alpha^{q_s}})$ be a SDH-challenge where q_s is a bound on the number of $\mathcal{O}\texttt{Spend}$ queries, $\mathcal{R}$ generates the public parameters as in the $\texttt{Setup}$ algorithm (except that it sets u_1 as g^t for some random $t \in \mathbb{Z}_p$) and selects $1 \leq i^* \leq q_a$ where q_a is a bound on the number of $\mathcal{O}\texttt{Add}$ queries. $\mathcal{R}$ computes q_s key pairs $(sk_{ots}^{(i)}, pk_{ots}^{(i)}) \leftarrow \Sigma_{ots}.\texttt{Keygen}(1^k)$ and sets w as $g^{\prod_{i=1}^{q_s}(\alpha + H(pk_{ots}^{(i)}))}$ (which is possible using the SDH challenge [BB08]). The reduction will answer the oracle queries as follows.

- $\mathcal{O}\texttt{Add}()$ queries: When the adversary makes the i-th $\mathcal{O}\texttt{Add}$ query to register a user, $\mathcal{R}$ will run the $\texttt{Keygen}$ algorithm if $i \neq i^*$ and set $\texttt{upk}^* \leftarrow g^\alpha$ otherwise.
- $\mathcal{O}\texttt{Corrupt}(\texttt{upk}/\texttt{mpk})$ queries: $\mathcal{R}$ returns the secret key if $\texttt{upk} \neq \texttt{upk}^*$ and aborts otherwise.
- $\mathcal{O}\texttt{AddCorrupt}(\texttt{upk}/\texttt{mpk})$ queries: $\mathcal{R}$ stores the public key which is now considered as registered.
- $\mathcal{O}\texttt{Withdraw}_{\mathcal{U}}(\texttt{bsk}, \texttt{upk})$ queries: $\mathcal{R}$ acts normally if $\texttt{upk} \neq \texttt{upk}^*$ and simulates the interactive proof of knowledge of α otherwise.
- $\mathcal{O}\texttt{Spend}(\texttt{upk}, 2^\ell)$ queries: $\mathcal{R}$ acts normally if $\texttt{upk} \neq \texttt{upk}^*$. Else, to answer the $j-$th query on $\texttt{upk}^*$, it will recover the pair $(sk_{ots}^{(j)}, pk_{ots}^{(j)})$ and computes $\mu \leftarrow g^{\prod_{i=1, i\neq j}^{q_s}(\alpha + H(pk_{ots}^{(i)}))}$ which verifies $\mu = w^{\frac{1}{\alpha + H(pk_{ots}^{(j)})}}$. It then uses $sk_{ots}^{(j)}$ as in the $\texttt{Spend}$ protocol.

The adversary then outputs two valid transcripts $(2^{\ell_1}, Z_1, \Pi_1)$ and $(2^{\ell_2}, Z_2, \Pi_2)$ which accuse $\texttt{upk}$ of double-spending. If $\texttt{upk} \neq \texttt{upk}^*$ then $\mathcal{R}$ aborts. The soundness of the proof implies that the forged transcript was signed using a key sk_{ots} and so that the proof involves an element $\mu = w^{\frac{1}{\alpha + H(\texttt{pk}_{ots})}}$. Since here we consider Type-2 attacks, $pk_{ots} \notin \{pk_{ots}^{(i)}\}_i$, so $\mathcal{R}$ extracts from the proof the element μ which can be used to break the q_s-SDH assumption in $\mathbb{G}_1$ (as in [BB08]).

$\mathcal{R}$ is then able to solve the SDH problem or to break the security of Σ_{ots} since it will not abort with probability $1/q_a$.

7 Conclusion

In this work, we have proposed the first practical construction of divisible E-cash which can be instantiated and proven secure in both the random oracle and standard models. Our $\texttt{Withdraw}$ and $\texttt{Spend}$ protocols are efficient and can be performed in constant times. Moreover, the bank can detect double-spendings by comparing the serial numbers of deposited coins to the ones of previously spent coins. Our protocol thus answers the problem left open by Izabachène and Libert. However, the computational cost and the storage space of our $\texttt{Deposit}$ protocol remains important but we argue that it is reasonable to assume that the bank has enough storage capacity and computational power. Finally, the way we build our tree is also compatible with divisible E-cash systems achieving weaker notions of anonymity (such as unlinkability) leading to very efficient protocols without these downsides (see Appendix A.2).

Acknowledgments

This work was supported in part by the French ANR Project ANR-12-INSE-0014 SIMPATIC and ANR-11-INS-0013 LYRICS, and in part by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 339563 – CryptoCloud).

References

[AGHO11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 649–666. Springer, August 2011.

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

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

[BFG+13] David Bernhard, Georg Fuchsbauer, Essam Ghadafi, Nigel P. Smart, and Bogdan Warinschi. Anonymous attestation with user-controlled linkability. *Int. J. Inf. Sec.*, 12(3):219–249, 2013.

[BGK95] Ernest F. Brickell, Peter Gemmell, and David W. Kravitz. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In Kenneth L. Clarkson, editor, *6th SODA*, pages 457–466. ACM-SIAM, January 1995.

[BGM+10] Jean-Luc Beuchat, Jorge E. González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal Ate pairing over Barreto-Naehrig curves. In Marc Joye, Atsuko Miyaji, and Akira Otsuka, editors, *PAIRING 2010*, volume 6487 of *LNCS*, pages 21–39. Springer, December 2010.

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

[BN05] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 319–331. Springer, August 2005.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

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

[CFT98] Agnes Hui Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 561–575. Springer, May / June 1998.

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

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

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

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

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

[Cha83] David Chaum. Blind signature system. In David Chaum, editor, *CRYPTO'83*, page 153. Plenum Press, New York, USA, 1983.

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

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

[CLM07] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *2007 IEEE Symposium on Security and Privacy*, pages 101–115. IEEE Computer Society Press, May 2007.

[CPP05] Hervé Chabanne, Duong Hieu Phan, and David Pointcheval. Public traceability in traitor tracing schemes. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 542–558. Springer, May 2005.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, August 1986.

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

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

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

[IL12] Malika Izabachène and Benoît Libert. Divisible E-cash in the standard model. In Michel Abdalla and Tanja Lange, editors, *PAIRING 2012*, volume 7708 of *LNCS*, pages 314–332. Springer, May 2012.

[NS00] Toru Nakanishi and Yuji Sugiyama. Unlinkable divisible electronic cash. In Josef Pieprzyk, Eiji Okamoto, and Jennifer Seberry, editors, *Information Security, Third International Workshop, ISW 2000, Wollongong, NSW, Australia, December 20-21, 2000, Proceedings*, volume 1975 of *Lecture Notes in Computer Science*, pages 121–134. Springer, 2000.

[Oka95] Tatsuaki Okamoto. An efficient divisible electronic cash scheme. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 438–451. Springer, August 1995.

[OO91] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 324–337. Springer, August 1991.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, August 1989.

[SPC95] Markus Stadler, Jean-Marc Piveteau, and Jan Camenisch. Fair blind signatures. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *EUROCRYPT'95*, volume 921 of *LNCS*, pages 209–219. Springer, May 1995.

A Implementations

We describe in this section the implementations of two divisible E-cash systems. The first one is an implementation, secure in the standard model, of the protocol described in section 5. The second one

is an implementation, secure in the ROM, of an unlinkable E-cash system derived from the high level description of section 4.2. Although this last implementation does not fulfil the anonymity requirement (as defined in our security model), it still fulfils the unlinkability property with a remarkable efficiency.

A.1 An Anonymous E-cash System in the Standard Model

For sake of clarity, we first recall the AGHO signature scheme from [AGHO11] that we will use to instantiate both Σ_0 and Σ_1. This signature scheme makes use of Type-3 groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ [GPS08] which offer the most efficient instantiations of pairings and of Groth-Sahai proofs since the SXDH assumption [GS08] holds in this setting.

For all the proofs of knowledge that users will have to produce in the various protocols, we will give the relations to be proven, and underline the variables that are private to the prover, and for which he wants to prove his knowledge.

Optimal Structure-Preserving Signatures for messages in $\mathbb{G}_1^2$ [AGHO11].

- $\texttt{Keygen}$: Set $\mathsf{sk} = (\alpha, \beta, \gamma_1, \gamma_2) \xleftarrow{\$} \mathbb{Z}_p^4$ and $\mathsf{pk} = (\widetilde{v}_1, \widetilde{v}_2, \widetilde{w}_1, \widetilde{w}_2) \leftarrow (\widetilde{g}^{\alpha}, \widetilde{g}^{\beta}, \widetilde{g}^{\gamma_1}, \widetilde{g}^{\gamma_2})$
- $\texttt{Sign}(\mathsf{sk}, (m_1, m_2))$: To sign a pair of messages $(m_1, m_2) \in \mathbb{G}_1^2$, select a random $r \xleftarrow{\$} \mathbb{Z}_p$ and return $\sigma = (z_1, z_2, \widetilde{z}) \leftarrow (g^r, g^{\beta - r\alpha} \cdot m_1^{-\gamma_1} \cdot m_2^{-\gamma_2}, \widetilde{g}^{\frac{1}{r}})$
- $\texttt{Verify}(\mathsf{pk}, (m_1, m_2), \sigma)$: Accept if $e(z_1, \widetilde{v}_1) \cdot e(z_2, \widetilde{g}) \cdot e(m_1, \widetilde{w}_1) \cdot e(m_2, \widetilde{w}_2) = e(g, \widetilde{v}_2)$ and $e(z_1, \widetilde{z}) = e(g, \widetilde{g})$.

Instantiation.

- $\texttt{Setup}$: The parameters are constructed as described in section 4.2. Since we are in the standard model a common reference string $CRS \in \mathbb{G}_1^4 \times \mathbb{G}_2^4$ for the Groth-Sahai proofs system [GS08] in the SXDH setting is also generated. The scheme Σ_{ots} will be instantiated by the signature scheme from [BB08], which is deterministic.
- $\texttt{BKeygen}()$: The bank runs $n+2$ times the $\texttt{Keygen}$ algorithm of the signature scheme described above to generate $(\mathsf{sk}_0^{(i)}, \mathsf{pk}_0^{(i)}) \leftarrow ((\alpha^{(i)}, \beta^{(i)}, \gamma_1^{(i)}, \gamma_2^{(i)}), (\widetilde{v}_1^{(i)}, \widetilde{v}_2^{(i)}, \widetilde{w}_1^{(i)}, \widetilde{w}_2^{(i)}))$ for each $0 \leq i \leq n$ and $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow ((\alpha, \beta, \gamma_1, \gamma_2), (\widetilde{v}_1, \widetilde{v}_2, \widetilde{w}_1, \widetilde{w}_2))$. It then computes, using the AGHO signature scheme, $\tau_s \leftarrow \texttt{Sign}(\mathsf{sk}_0^{(|s|)}, (g_s, h_s))$ for every $s \in \mathcal{S}_n$. Eventually, it sets bsk as sk_1 and bpk as $(\{\mathsf{pk}_0^{(i)}\}_i, \mathsf{pk}_1, \{\tau_s\}_{s \in \mathcal{S}_n})$.
- $\texttt{Keygen}()$: Each user (resp. merchant) selects a random $\mathsf{usk} \xleftarrow{\$} \mathbb{Z}_p$ (resp. msk) and gets $\mathsf{upk} \leftarrow g^{\mathsf{usk}}$ (resp. $\mathsf{mpk} \leftarrow g^{\mathsf{msk}}$).
- $\texttt{Withdraw}(\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk}))$: To withdraw a divisible coin, the user first selects a random $x \in \mathbb{Z}_p$ and computes $(U_1, U_1) \leftarrow (u_1^{\mathsf{usk}}, u_2^x)$. He then sends upk, U_1, U_2 and proves using the Schnorr's interactive protocol [Sch89], that

$$upk = g^{\underline{\mathsf{usk}}} \qquad \wedge \qquad U_1 = u_1^{\underline{\mathsf{usk}}} \qquad \wedge \qquad U_2 = u_2^{\underline{x}}$$

  If the proof is valid and if U_2 was not previously used, the bank computes the AGHO signature $\sigma \leftarrow \texttt{Sign}(\mathsf{sk}_1, (U_1, U_2))$ and sends it to the user who sets $C \leftarrow (x, \sigma)$.
- $\texttt{Spend}(\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, \mathsf{mpk}, 2^\ell), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, 2^\ell))$: To spend a value 2^ℓ, the user selects an unspent node s of level $n - \ell$ and computes $r \leftarrow H(info)$ and $(t_s, v_s) \leftarrow (g_s^x, \mathsf{upk}^r \cdot h_s^x)$. He then generates $\mathsf{sk}_{ots} \xleftarrow{\$} \mathbb{Z}_p$, sets pk_{ots} as $\widetilde{g}^{\mathsf{sk}_{ots}}$ and computes $\mu \leftarrow w^{\frac{1}{\mathsf{usk} + H(\mathsf{pk}_{ots})}}$.
  He parses τ_s as $(z_1^{(s)}, z_2^{(s)}, \widetilde{z}^{(s)})$ and σ as $(z_1, z_2, \widetilde{z})$ and computes Groth-Sahai commitments to usk and x (2 elements of $\mathbb{G}_2$ each), to $g_s, h_s, U_1, U_2, \mu, z_1^{(s)}, z_2^{(s)}, z_1$ and z_2 (2 elements of $\mathbb{G}_1$ each) and to $\widetilde{z}^{(s)}$ and $\widetilde{z}$ (2 elements of $\mathbb{G}_2$ each). Then, he computes a NIZK proof π that

$$t_s = \underline{g_s}^{\underline{x}} \wedge v_s = (g^r)^{\underline{\mathsf{usk}}} \cdot \underline{h_s}^{\underline{x}} \wedge \underline{U_1} = u_1^{\underline{\mathsf{usk}}} \wedge \underline{U_2} = u_2^{\underline{x}} \wedge \underline{\mu}^{(\underline{\mathsf{usk}} + H(\mathsf{pk}_{ots}))} = w$$

The proof consists of 2 elements of $\mathbb{G}_1$ and 4 elements of $\mathbb{G}_2$ for each equation. Now, the user must compute a NIWI proof π' that τ_s and σ are valid signatures, *i.e.*,

$$e(\underline{z_1}^{(s)}, \widetilde{v}_1^{(n-\ell)}) \cdot e(\underline{z_2}^{(s)}, \widetilde{g}) \cdot e(\underline{g_s}, \widetilde{w}_1^{(n-l)}) \cdot e(\underline{h_s}, \widetilde{w}_2^{(n-\ell)}) = e(g, \widetilde{v}_2^{(n-\ell)}) \quad \wedge \quad e(\underline{z_1}^{(s)}, \underline{\widetilde{z}}^{(s)}) = e(g, \widetilde{g})$$
$$e(\underline{z_1}, \widetilde{v}_1) \cdot e(\underline{z_2}, \widetilde{g}) \cdot e(\underline{g_s}, \widetilde{w}_1) \cdot e(\underline{h_s}, \widetilde{w}_2) = e(g, \widetilde{v}_2) \qquad \wedge \qquad e(\underline{z_1}, \underline{\widetilde{z}}) = e(g, \widetilde{g}).$$

The proof of the first and third equations consists of 2 elements of $\mathbb{G}_2$ each. The proof of the second and the fourth equations consists of 4 elements of $\mathbb{G}_1$ and 4 elements of $\mathbb{G}_2$ each.
Finally, the user computes the BB one-time signature $\eta \leftarrow g^{\frac{1}{\mathsf{sk}_{ots}+H(t_s||v_s||\pi||\pi'||r)}}$ and sends it to $\mathcal{M}$ along with $t_s, v_s, \mathsf{pk}_{ots}, \pi, \pi'$. The global size of the `Spend` transcript is then of 39 elements of $\mathbb{G}_1$ and 40 elements of $\mathbb{G}_2$.
The merchant checks the validity of the proofs and accepts the transaction if everything is correct. In such a case, he stores $Z \leftarrow (t_s, v_s)$ and $\Pi \leftarrow (\pi, \pi', \mathsf{pk}_{ots}, \eta)$.
The `Deposit` and `Identify` algorithms do not depend on the instantiation of the different signature schemes, their descriptions can thus be found in Section 5.

A.2 An Unlinkable E-cash System in the ROM

Our goal in this section is to show that, by relying on strong assumptions (the random oracle model and the blind-LRSW assumption [BFG+13]) and by weakening the anonymity requirements (namely achieving only the unlinkability property and allowing an entity to recover spenders' identities, as in fair E-cash), we can construct a very efficient divisible E-cash system whose `Spend` transcripts consist of only 5 elements of $\mathbb{G}_1$ and 2 elements of $\mathbb{Z}_p$. We first recall the signature scheme from [BFG+13], which is a variant of the Camenisch-Lysyanskaya signatures [CL04].

Blind-LRSW signatures [BFG+13]

- `Keygen` : Set $\mathsf{sk} = (\alpha, \beta) \xleftarrow{\$} \mathbb{Z}_p^2$ and $\mathsf{pk} = (\widetilde{v}_1, \widetilde{v}_2) \leftarrow (\widetilde{g}^\alpha, \widetilde{g}^\beta)$
- `Sign`(sk, g^m) : On input an element g^m for $m \in \mathbb{Z}_p$, select a random $r \xleftarrow{\$} \mathbb{Z}_p$ and return $\sigma = (z_1, z_2, z_3, z_4) \leftarrow (g^r, g^{r\beta}, g^{r(\alpha + m\cdot\alpha\beta)}, g^{r\beta m})$
- `Verify`(pk, m, σ) : Accept if $z_4 = z_2^m$, $e(z_1, \widetilde{v}_2) = e(z_2, \widetilde{g})$ and $e(z_3, \widetilde{g}) = e(z_1 \cdot z_4, \widetilde{v}_1)$.

An interesting feature of this scheme is that any signature $\sigma = (z_1, z_2, z_3, z_4)$ on a message m can be randomized by selecting a random $t \xleftarrow{\$} \mathbb{Z}_p$ and computing $\sigma' \leftarrow (z_1^t, z_2^t, z_3^t, z_4^t)$ which is still a valid signature on m. The point is that, given (σ, σ') (but not m), it is hard, under the XDH assumption, to decide whether these signatures were issued on the same message or not.

Instantiation.

Remark 8. As explained above, our purpose here is to achieve the best efficiency with our construction. We will therefore consider an entity, that we will call *opening* authority, able to recover the spender's identity from any transcript. Assuming such an authority releases users from the need to compute a security tag since the identification of double spenders will now be performed differently. Again, we stress we can use the security tag to avoid such an authority but this will increase the complexity of the resulting scheme.

- `Setup`: The parameters, whose construction is described in section 4.2, are now generated by the opening authority which also stores the scalars $\{r_s\}_{s \in \mathcal{S}_n}$. Since we are in the ROM the description of a hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ is added to the public parameters.
- `BKeygen`(): The bank runs the `Keygen` algorithm of the signature scheme described above to generate $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow ((\alpha, \beta), (\widetilde{v}_1, \widetilde{v}_2))$.
- `Keygen`(): Each user (resp. merchant) selects a random $\mathsf{usk} \xleftarrow{\$} \mathbb{Z}_p$ (resp. msk) and gets $\mathsf{upk} \leftarrow g^{\mathsf{usk}}$ (resp. $\mathsf{mpk} \leftarrow g^{\mathsf{msk}}$).

– `Withdraw`$(\mathcal{B}(\mathsf{bsk}, \mathsf{upk}), \mathcal{U}(\mathsf{usk}, \mathsf{bpk}))$: To withdraw a divisible coin, the user first selects a random $x \in \mathbb{Z}_p$ and computes $U \leftarrow g^x$. He then sends upk and U and proves using the Schnorr's interactive protocol [Sch89], that:

$$\mathsf{upk} = g^{\underline{\mathsf{usk}}} \qquad \wedge \qquad U = g^{\underline{x}}$$

If the proof is valid and if U was not previously used, the bank computes $\sigma \leftarrow \mathtt{Sign}(\mathsf{sk}_1, U)$ and sends it to the user who sets $C \leftarrow (x, \sigma)$.

– `Spend`$(\mathcal{U}(\mathsf{usk}, C, \mathsf{bpk}, \mathsf{mpk}, 2^\ell), \mathcal{M}(\mathsf{msk}, \mathsf{bpk}, 2^\ell))$: To spend a value 2^ℓ, the user selects an unspent node s of level $n - \ell$ and computes $t_s \leftarrow g_s^x$. He then parses σ as (z_1, z_2, z_3, z_4) and randomizes it by selecting a random $t \xleftarrow{\$} \mathbb{Z}_p$ and computing $\sigma' = (z_1', z_2', z_3', z_4') \leftarrow (z_1^t, z_2^t, z_3^t, z_4^t)$. Next, he proves that σ' is a valid signature on x and that $t_s = g_s^x$. To do so he selects a random $k \in \mathbb{Z}_p$ and computes $k_1 \leftarrow g_s^k$, $k_2 \leftarrow z_2^k$, $c \leftarrow \mathcal{H}(\sigma'||t_s||k_1||k_2||\mathit{info})$ and $z \leftarrow k + c \cdot x$. Finally, he sends (σ', t_s, c, z) to the merchant.
The transcript thus contains 5 elements of $\mathbb{G}_1$ and 2 elements of $\mathbb{Z}_p$ and requires only 7 exponentiations in $\mathbb{G}_1$. It is worthy to note that these exponentiations can be pre-computed since they only involve elements from the public parameters or from the coin C.
Upon receiving (σ', t_s, c, z), the merchant (who knows g_s since here we only require the unlinkability property) computes $l_1 \leftarrow g_s^z \cdot t_s^{-c}$, $l_2 \leftarrow (z_2')^z \cdot (z_4')^{-c}$ and $c' \leftarrow \mathcal{H}(\sigma'||t_s||l_1||l_2||\mathit{info})$. He accepts the transaction if:

$$c = c' \qquad \wedge \qquad e(z_1', \widetilde{v}_2) = e(z_2', \widetilde{g}) \qquad \wedge \qquad e(z_3', \widetilde{g}) = e(z_1' \cdot z_4', \widetilde{v}_1)$$

In such a case, he stores $Z \leftarrow (g_s, t_s)$ and $\Pi \leftarrow (\sigma', c, z)$.

– `Deposit`$(\mathcal{M}(\mathsf{msk}, \mathsf{bpk}, (2^\ell, Z, \Pi)), \mathcal{B}(\mathsf{bsk}, L, \mathsf{mpk}))$: Upon receiving a transcript, the bank will check that it was not previously deposited and verify its validity. Since the bank here knows the spent node s, it computes, for each $f \in \mathcal{F}_n(s)$ the elements $z_i \leftarrow e(t_s, \widetilde{g}_{s' \mapsto f})$ and check whether $z_i \in L$. If $\forall i, z_i \notin L$ then the bank will add these elements to this list and store the transcript $(2^\ell, Z, \Pi)$. Else, there is an element $z' \in L$ such that $z_i = z'$. The bank will recover the corresponding transcript $(2^{\ell'}, Z', \Pi')$ and output $[(2^\ell, Z, \Pi), (2^{\ell'}, Z', \Pi')]$. It worthy to note that the bank only has to compute 2^l pairings instead of the 2^n pairings required by the anonymous version described in section 5.
Since we have assumed an opening authority there is no `Identify` algorithm. To open a transcript $(2^\ell, Z, \Pi)$, this entity will parse Z as (g_s, t_s) and compute $U \leftarrow t_s^{r_s^{-1}}$ (recall that the scalars r_s have been stored during the `Setup`). Therefore, $U = g^x$ so the bank requesting the opening is able to recover the identity of the spender by seeking which user has used U during a `Withdraw` protocol.