Practical Divisible E-Cash

Patrick Märtens

Mathematisches Institut, Justus-Liebig-Universität Gießen
patrickmaertens@gmx.de

April 9, 2015

Abstract. Divisible e-cash systems allow a user to withdraw a wallet containing K coins and to spend $k \leq K$ coins in a single operation, respectively. Independent of the new work of Canard, Pointcheval, Sanders and Traoré (Proceedings of PKC '15) we present a practical and secure divisible e-cash system in which the bandwidth of each protocol is constant while the system fulfills the standard security requirements (especially which is *unforgeable* and *truly anonymous*) in the random oracle model. In other existing divisible e-cash systems that are truly anonymous, either the bandwidth of withdrawing depends on K or the bandwidth of spending depends on k. Moreover, using some techniques of the work of Canard, Pointcheval, Sanders and Traoré we are also able to prove the security in the standard model.

Furthermore, we show an efficient attack against the unforgeability of Canard and Gouget's divisible e-cash scheme (FC '10).

Finally, we extend our scheme to a divisible e-cash system that provides withdrawing and spending of an arbitrary value of coins (not necessarily a power of two) and give an extension to a fair e-cash scheme.

Keywords: E-Cash, divisible, constant-size, accumulator, pairings, standard model

1 Introduction

Electronic cash (e-cash) was introduced by Chaum [Cha83] as the digital analogue of regular money. Basically, an (offline) e-cash system consists of three parties (the bank $\mathcal{B}$, user $\mathcal{U}$, and merchant $\mathcal{M}$) and three protocols (Withdrawal, Spend, and Deposit). A user withdraws coins from the bank and applies the coins to pay a merchant (without involving the bank during spending). Then, the merchant deposits the coins to the bank.

Since electronic coins are just digital data, they can easily be duplicated, and hence spent several times, which is called double-spending. Thus, an (offline) e-cash system needs a mechanism to detect such a double-spending and to identify the defrauder afterwards.

From the bank's point of view, the most important security requirement is that no one is able to forge a valid coin or can double-spend a coin without being identified (*balance*). Furthermore, the bank should be able to identify a double-spender without a third party. From the users point of view, the most important security requirements are that honest users are anonymous (*anonymity*) and cannot be accused to have performed a double-spending (*exculpability*).

The main challenge in designing e-cash systems is to achieve strong security requirements and high efficiency.

1.1 Related Work

Compact E-Cash. To speed up the withdrawal phase, Camenisch, Hohenberger and Lysyanskaya [CHL05] designed the first compact e-cash scheme, which has been modified and extended in [Wei05,

CHL06a, ASM07, CLM07, CGT08, BCKL09, Mär15]. In such a scheme, a user can withdraw a wallet $\mathcal{W}$ containing K coins and can spend each coin unlinkably. However, each coin has to be spend one by one, which is a drawback of such systems.

Divisible E-Cash. In a divisible e-cash system, a user also withdraws a wallet $\mathcal{W}$ containing K coins but then can spend $k \leq K$ coins together efficiently. According to this, such schemes solve the drawback of compact e-cash. Another way of seeing divisible e-cash is that a user withdraws one coin with monetary value K and then divides this coin to spend a coin with monetary value $k \leq K$. It is common that K resp. k is a power of two.

There are a lot of divisible e-cash systems in the literature [OO91, Oka95, CFT98, NS00, CG07, ASM08, CG10, IL13, CPST15], but the first scheme that provides the security requirements shown above (especially which is truly anonymous) has been proposed in [CG07]. Like in other divisible e-cash systems, a binary tree is used to represent the wallet. Let $K = 2^L$ be the number of coins of the wallet, then the binary tree consists of $L + 2$ levels $0, \ldots, L + 1$. Each node at level $i, 0 \leq i \leq L$, has monetary value 2^{L-i} and is related to a serial number, denoted by $S_{i,j}$, which itself is related to a key $\kappa_{i,j}$ for $0 \leq j \leq 2^i - 1$. The nodes at level $L + 1$ have no monetary value. Each serial number can be computed uniquely from one of its ascendants. During the withdrawal protocol, the user obtains a signature on $\kappa_{0,0}$. To spend a coin with monetary value $2^\ell \leq 2^L$, the user sends an unspent serial number $S_{L-\ell,j}$ and proves in zero knowledge that $S_{L-\ell,j}$ is correctly computed from $\kappa_{0,0}$, using expensive proofs of double discrete logarithms and the "or" statement. This is a big drawback of the system. Additionally, due to the system setup, it is questionable whether this system is in fact implementable (see also [ASM08, CG10]).

A practical divisible e-cash system based on bounded accumulators has been proposed in [ASM08]. The binary tree consists of $L+1$ levels $0, \ldots, L$. During the withdrawal protocol, all keys $\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}$ in the same level $i, 0 \leq i \leq L$, are accumulated into an accumulator V_i. Then the user obtains $L + 1$ signatures, each on one accumulator. To spend a coin with monetary value $2^\ell \leq 2^L$, the user sends an unspent serial number $S_{L-\ell,j}$ and only proves in zero knowledge that the (secret) key $\kappa_{L-\ell,j}$, corresponding to $S_{L-\ell,j}$, is inside the accumulator $V_{L-\ell}$. Notice, that neither during the withdrawal protocol nor during the spending phase the user proves any relation about the accumulator values. Hence, a malicious user is able to withdraw a wallet containing $(L + 1)2^L$ coins[1], instead of 2^L. Consequently, this system does not fulfill the balance requirement, only a weaker *statistical balance* property (see Section 4.1 for details).

This problem has been solved in [CG10]. The system is also based on bounded accumulators, but with a new technique to prove that several revealed values are inside an accumulator. Like in [CG07], the binary tree consists of $L + 2$ levels $0, \ldots, L + 1$ and the nodes at level $L + 1$ have no monetary value. As in [ASM08], during the withdrawal protocol, all keys $\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}$ in the same level $i, 1 \leq i \leq L + 1$, are accumulated into an accumulator V_i. Furthermore, all keys except of $\kappa_{0,0}$ are accumulated into one additional accumulator V. Then the user obtains $L + 2$ signatures, each on one accumulator. To spend a coin with monetary value $2^\ell \leq 2^L$, the user sends an unspent serial number $S_{L-\ell,j} = \kappa_{L-\ell+1,2j} || \kappa_{L-\ell+1,2j+1}$, corresponding to the key $\kappa_{L-\ell,j}$. Then the user proves in zero knowledge that the two keys $\kappa_{L-\ell+1,2j}$ and $\kappa_{L-\ell+1,2j+1}$ are inside the accumulator $V_{L-\ell+1}$ and that all keys which can be computed from $S_{L-\ell,j}$ are inside the accumulator V. Hence, during spending the user has to prove that a serial number is derived from its father. Nevertheless, in Section 4.2.1 we show that every (dishonest) user is able to withdraw a wallet containing $1.5 \cdot 2^L$ coins instead of 2^L coins and thus breaks the unforgeability of the system.

While the bandwidth of the spending phase in [ASM08] and [CG10] is independent of the monetary value 2^ℓ, the bandwidth of the withdrawal protocol linearly depends on $L = \log(K)$.

Izabacène and Libert [IL13] provided the first divisible e-cash scheme which security is proven in the standard model (rather than the random oracle model). However, their construction is rather inefficient

[1] In [ASM08] is said, that a dishonest user is able to withdraw a wallet containing $L2^L$ coins. However, each of the $L+1$ accumulators leads to a wallet containing 2^L coins.

(see also [CPST15]).

Fair E-Cash. Regardless of whether e-cash systems are compact or divisible, most of them provide perfect anonymity (e.g. [CFN89, Bra94]) or computational anonymity. However, as pointed out by von Solms and Naccache [vSN92], anonymity "can potentially lead to perfect crime", such as perfect blackmailing and money laundering (see also [SPC95]). To prevent perfect crime, an e-cash scheme can be extended to a so-called fair e-cash scheme. Fair (offline) e-cash was independently introduced by [FTY96] and [CMS96]. As an extension of an e-cash system, a fair scheme has an additional party $\mathcal{T}$, which is referred to as trusted third party (TTP). Further, a fair scheme has two additional protocols, named OwnerTracing, which allows tracing of suspicious spendings, and CoinTracing, which allows finding the destination of suspicious withdrawals. Hence, $\mathcal{T}$ (and only $\mathcal{T}$) is able to revoke the anonymity under a suspicious activity. Frankel, Tsiounis and Yung [FTY98] (see also [GT03]) provided an elegant construction of fair e-cash, where $\mathcal{T}$ has only to publish its public key and is only involved in the two tracing protocols if required.

1.2 Our Contribution

We combine features of the systems in [ASM08] and [CG10] to design a divisible e-cash system that provides the standard security requirements and in which the bandwidth of each protocol is constant. Furthermore, using some techniques of [CPST15], we are able to prove the security of the system in the standard model, assuming that the used hash function is collision-resistant and outputs values that are indistinguishable from uniform. The binary tree and the serial numbers are computed as in [ASM08] (but with related hash functions), but our tree has one more level. We will refer to the K nodes in this $(L+1)$-th level as *serial keys*. During the withdrawal protocol, only this K serial keys are accumulated into one accumulator V. Hence, the user only obtains one signature on V. To spend a coin with monetary value $2^\ell \leq K$, the user sends an unspent serial number $S_{L-\ell,j}$ and only has to prove in zero knowledge (with the technique presented in [CG10]) that all 2^ℓ serial keys that derive from $S_{L-\ell,j}$ are inside the accumulator V.[2] Because of the suitable computation of the binary tree, this proof is sufficient to provide balance.

Besides, this new feature allows the construction of efficient protocols where users can withdraw and spend arbitrary monetary values smaller than or equal to K (and not necessarily a power of two).

Finally, we present an approach how to extend our scheme to a fair e-cash system that allows owner and coin tracing, where $\mathcal{T}$ has only to publish its public key and is only involved in the two tracing protocols if required.

1.3 Organization

We discuss technical preliminaries such as mathematical assumptions and cryptographic building blocks in the next section. In Section 3 we define the security model for divisible e-cash schemes. Section 4 gives an overview of the divisible e-cash schemes given in [ASM08] and [CG10]. We also present an attack against the unforgeability of the scheme in [CG10]. A detailed presentation of our new construction and the security analysis are given in Section 5. In Section 6 we compare the efficiency of our scheme with the underlying schemes in [ASM08] and [CG10] and with the new divisible e-cash system in [CPST15]. Three extensions of our construction are presented in Section 7. Finally we conclude in Section 8.

[2] We could also interpret each serial number S as a *master serial number* and each serial key as a serial number. Hence, each master serial number $S_{L-\ell,j}$ would lead to 2^ℓ serial numbers.

2 Preliminaries

2.1 Bilinear Maps

A bilinear map, also called pairing, maps two group elements to one group element. Let $\hat{e}$ be a bilinear map $\hat{e}\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ such that the following properties hold.

- $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic multiplicative groups of prime order p.
- Each element of $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ has a unique binary representation.
- g is a generator of $\mathbb{G}_1$ and h is a generator of $\mathbb{G}_2$.
- (Bilinear:) $\forall\, a \in \mathbb{G}_1, b \in \mathbb{G}_2$ and $x, y \in \mathbb{Z}_p$, $\hat{e}\left(a^x, b^y\right) = \hat{e}\left(a, b\right)^{xy}$.
- (Non-degenerate:) $\hat{e}\left(g, h\right) \neq 1$.
- The group action in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ and the bilinear map $\hat{e}$ are all efficiently computable.

We call $(\mathbb{G}_1, \mathbb{G}_2)$ a bilinear group pair. Let $\mathcal{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g, h)$ be the global parameters of a pairing.

Galbraith, Paterson and Smart [GPS06] separated different possible pairing instantiations into three basic types:

Type-1: $\mathbb{G}_1 = \mathbb{G}_2$.

Type-2: $\mathbb{G}_1 \neq \mathbb{G}_2$ but there is an efficiently computable homomorphism $\psi\colon \mathbb{G}_2 \to \mathbb{G}_1$.

Type-3: $\mathbb{G}_1 \neq \mathbb{G}_2$ and there are no efficiently computable homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$.

2.2 Mathematical Assumptions

The security of our construction depends on the following assumptions.

Definition 1 (**Decisional Diffie-Hellman**). *The Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is the following: On input a quadruple $\left(g, g^a, g^b, g^c\right) \in \mathbb{G}^4$, output 1 if $g^c = g^{ab}$ and 0 otherwise. We say that the DDH assumption holds in $\mathbb{G}$ if no PPT algorithm has non-negligible advantage over random guessing in solving the DDH problem in $\mathbb{G}$.*

Definition 2 (**Symmetric External Diffie-Hellman [ACHdM05, AWSM07]**). *The Symmetric External Diffie-Hellman (SXDH) assumption in $(\mathbb{G}_1, \mathbb{G}_2)$ states that the DDH assumption holds in $\mathbb{G}_1$ and $\mathbb{G}_2$. It implies that there are no efficiently computable isomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ (see [AWSM07]).*

Definition 3 (q-**Strong Diffie-Hellman [BB04, ASM07, Sch11]**). *The q-Strong Diffie-Hellman (q-SDH) problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is the following: On input a $(q+3)$-tuple $(g, g^x, g^{x^2}, \ldots, g^{x^q}, h, h^x) \in \mathbb{G}_1^{q+1} \times \mathbb{G}_2^2$, output a pair $(g^{1/(x+c)}, c) \in \mathbb{G}_1 \times \mathbb{Z}_p^*$. We say that the q-SDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2)$ if no PPT algorithm has non-negligible advantage in solving the q-SDH problem in $(\mathbb{G}_1, \mathbb{G}_2)$.*

2.3 Building Blocks

Zero-Knowledge Proofs of Knowledge. A zero-knowledge proof of knowledge (PoK) is an interactive protocol during which a prover proves to a verifier that he knows a secret that verifies a given relation. We need those proofs for proving statements related to knowledge of discrete logarithms constructed over cyclic groups of known prime order p. We follow the notation given by [CS97], for example, $PoK\left\{(a, b, c) : A = g_1^a g_2^b \ \wedge \ B = g_3^a g_4^c\right\}$ denotes a PoK such that the prover knows the secret $(a, b, c) \in \mathbb{Z}_p^3$

such that $A = g_1^a g_2^b$ and $B = g_3^a g_4^c$. Using the Fiat-Shamir heuristic [FS86], these proofs can be made non-interactive and are secure in the random oracle model [BR93]. This is referred to as a signature of knowledge (SoK) (see [CS97]). Back to the example shown above, the corresponding SoK is denoted as $SoK\left\{(a,b,c) : A = g_1^a g_2^b \ \wedge \ B = g_3^a g_4^c\right\}(m)$, where $m \in \{0,1\}^*$ is some message.

Groth-Sahai Non-interactive Proof Systems. Groth and Sahai [GS08] constructed efficient and practical non-interactive witness-indistinguishable (NIWI) proofs and non-interactive zero knowledge (NIZK) proofs for bilinear groups. These proof systems can be instantiated based on the SXDH assumption. To design our e-cash scheme we make use of NIZK proofs of the form

$$\prod_{i=1}^{m} \hat{e}\left(\underline{\mathcal{X}_i}, \mathcal{B}_i\right) = 1, \qquad \prod_{i=1}^{m} \hat{e}\left(\underline{\mathcal{X}_i}, \mathcal{B}'_i\right) = \hat{e}\left(g,h\right), \qquad \prod_{i=1}^{n'} \mathcal{A}_i^{\underline{y_i}} \prod_{i=1}^{m} \underline{\mathcal{X}_i}^{b_i} \prod_{i=1}^{m} \prod_{j=1}^{n'} \underline{\mathcal{X}_i}^{\gamma_{i,j}\underline{y_j}} = \mathcal{T}_1,$$

where $\underline{\mathcal{X}_1}, \ldots, \underline{\mathcal{X}_m} \in \mathbb{G}_1$ and $\underline{y_1}, \ldots, \underline{y_{n'}} \in \mathbb{Z}_p$ are variables that are private to the prover and $\mathcal{A}_i, \mathcal{T}_1 \in \mathbb{G}_1$, $\mathcal{B}_i, \mathcal{B}'_i \in \mathbb{G}_2$ and $b_i, \gamma_{i,j} \in \mathbb{Z}_p$ are known constants.[3]

For detailed information we refer to [GS08, GS10, EG14].

Bounded Accumulator. An accumulator scheme is a method for accumulating several values into one element, called the accumulator. The notion, bounded accumulator, was introduced in [AWSM07] as an accumulator with a limit q of values that can be accumulated. Au *et al.* [AWSM07] observed that the construction due to Nguyen [Ngu05], whose security relies on the q-SDH assumption, is already a bounded accumulator. A PoK to prove that several known values are accumulated into a secret accumulator has been proposed in [CG10]. We briefly describe the construction.

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$ and let u_0 be a generator of $\mathbb{G}_1$ and v_0 be a generator of $\mathbb{G}_2$. The generation algorithm randomly selects $\alpha \in_R \mathbb{Z}_p^*$ and computes the elements $u_i = u_0^{\alpha^i}$ and $v_i = v_0^{\alpha^i}$ for $1 \leq i \leq q$. The public parameters of the accumulator scheme are $(\mathcal{G}, u_0, \ldots, u_q, v_0, \ldots, v_q)$. The accumulator of a set of values $\{x_0, \ldots, x_{q-1}\} \subset \mathbb{Z}_p \setminus \{-\alpha\}$ is computed as $V = u_0^{\prod_{j=0}^{q-1}(\alpha + x_j)}$, what does not require knowledge of the secret α since the elements $u_0, \ldots, u_q$ are publicly known. In the following, we will denote the accumulator of the set $\{x_0, \ldots, x_{q-1}\}$ as $\mathsf{Acc}\left(\{x_0, \ldots, x_{q-1}\}\right)$. Let $I \subseteq \{0, \ldots, q-1\}$. A witness W_I, to prove that several values x_i for $i \in I$ are accumulated in V, is computed as $W_I = u_0^{\prod_{j=0, j \notin I}^{q-1}(\alpha + x_j)}$. Thus, the witness and values satisfy the equation $\hat{e}\left(V, v_0\right) = \hat{e}\left(W_I, v_0^{\prod_{j \in I}(\alpha + x_j)}\right)$. Notice, that the element $v_I := v_0^{\prod_{j \in I}(\alpha + x_j)}$ can be analogously computed as V with the elements $v_0, \ldots, v_q$.

To prove non-interactively that several known values $x_0, \ldots, x_{k-1}$ are accumulated in a secret accumulator V, a user computes the corresponding witness W_I and has to prove the relation

$$\hat{e}\left(\underline{V}, v_0\right) \hat{e}\left(\underline{W_I}, v_I^{-1}\right) = 1,$$

where $v_I^{-1} = v_0^{-\prod_{j=0}^{k-1}(\alpha + x_j)}$. This proof is feasible in the random oracle model and in the standard model using the Groth-Sahai proof systems.

Special Signature Schemes. Camenisch and Lysyanskaya [CL02] presented special signature schemes with two efficient protocols for (1) issuing a signature on a committed message and for (2) proving knowledge of a message-signature pair. ESS+ is such a signature scheme and has been proposed in [ASM08] (see also [Au09, Section 3.3]) as an extended version of ESS ([AWSM07]). It allows signing blocks of messages $m_1, \ldots, m_L \in \mathbb{Z}_p$ together with a group element $M \in \mathbb{G}_1$, and is secure (EUF-CMA) in the

[3] As shown in [EG14], NIWI proofs of the form $\prod_{i=1}^{m} \hat{e}\left(\underline{\mathcal{X}_i}, \mathcal{B}'_i\right) = \hat{e}\left(g,h\right)$ are in fact NIZK proofs.

generic group model under the SXDH assumption. We briefly describe the constructions of ESS and ESS+. We start with ESS+ because ESS can be seen as a special case of ESS+.

Again, let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$ and additional generators $g_A, g_B, g_0, \ldots, g_L \in \mathbb{G}_1$. The secret signing key is $\mathsf{sk} = (X, y) \in \mathbb{G}_1 \times \mathbb{Z}_p^*$ and the public key is $\mathsf{pk} = (Y, Z) = (h^y, \hat{e}(X, h)) \in \mathbb{G}_2 \times \mathbb{G}_T$. To sign a message $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$, the signer randomly selects $a, b, c \in_R \mathbb{Z}_p^*$ and computes $A = X(Mg_A^a)^c, B = \left(gg_B^a g_0^b g_1^{m_1} \cdots g_L^{m_L}\right)^{1/(y+c)}$ and $C = h^c$. The signature on the message is $\sigma = (A, B, C, a, b)$. Everyone can verify the equations $\hat{e}(A, h) \stackrel{?}{=} Z\hat{e}(Mg_A^a, C)$ and $\hat{e}(B, CY) \stackrel{?}{=} \hat{e}\left(gg_B^a g_0^b g_1^{m_1} \cdots g_L^{m_L}, h\right)$.

To obtain a signature on a committed message $m = (M, m_1, \ldots, m_L) \in \mathbb{G}_1 \times \mathbb{Z}_p^L$, a user and the signer perform the following issue protocol. The user randomly selects $a, b' \in_R \mathbb{Z}_p^*$ and computes the perfectly hiding commitment $(C_1, C_2) = \left(Mg_A^a, g_B^a g_0^{b'} g_1^{m_1} \cdots g_L^{m_L}\right)$. Then, the user sends C_1, C_2 together with a proof of knowledge of representation of C_2 to the signer. After successful verification of the proof, the signer randomly selects $b'', c \in_R \mathbb{Z}_p^*$ and computes $A = XC_1^c, B = \left(gg_0^{b''} C_2\right)^{1/(y+c)}$ and $C = h^c$. After that, the signer sends A, B, C, b'' back to the user. Finally, the user computes $b = b' + b'' \pmod p$ and sets the signature as (A, B, C, a, b).

The ESS scheme is the same as ESS+, but we always have $a = 0$ rather than $a \in_R \mathbb{Z}_p^*$. So, there is no need for the generators g_A, g_B and no need for a. Thus, ESS is slightly more efficient than ESS+, but the element $M = C_1$ is known to the signer.

ESS and Accumulators. The goal of ESS+ is constructing a signature scheme for signing a committed message. So, in contrast to ESS, the signer should not learn anything about M and $m_1, \ldots, m_L$. If the element M is an accumulator $V = \mathsf{Acc}(\{x_0, \ldots, x_{q-1}\}) = u_0^{\prod_{j=0}^{q-1}(\alpha + x_j)}$, and that is exactly the case in [AWSM07], [ASM08], [CG10] and our new construction, we point out that there is no need for ESS+ since ESS is sufficient. To construct a divisible e-cash system, we don't need that the bank (signer) is able to sign a committed message on V, we just need that the bank (signer) learns nothing about V during the withdrawal (issue) protocol.

We give a new approach. To obtain a signature, the user just computes $V^* = V^s = u_0^{s\prod_{j=0}^{q-1}(\alpha + x_j)}$ for a random value $s \in_R \mathbb{Z}_p^*$ and sends V^* instead of C_1 to the signer (together with the commitment C_2). Consequently, the accumulator V is perfectly hidden. Furthermore, the user is still able to compute a valid witness $W_I^* = W_I^s = u_0^{s\prod_{j=0, j\notin I}^{q-1}(\alpha + x_j)}$. Thus, we still have $\hat{e}(V^*, v_0) = \hat{e}\left(W_I^*, v_0^{\prod_{j\in I}(\alpha + x_j)}\right)$. Obviously, this accumulator scheme still is a bounded accumulator, i.e. it is infeasible to output a tuple $\left(V^*, W_0^*, x_0, \ldots, W_q^*, x_q\right)$ such that $\hat{e}(V^*, v_0) = \hat{e}(W_i^*, v_1 v_0^{x_i})$ for all $0 \le i \le q$ (as defined in [Au09, Definition 4.9]). Note, that an adversarial user can anyway compute an accumulator as $V = u_0^{s\prod_{j=0}^{q-1}(\alpha + x_j)}$ for a randomly chosen s in the e-cash schemes [AWSM07], [ASM08] and [CG10].

In comparison to ESS+, the user has to store the value s instead of a. Thus, the storage space is the same. However, the user doesn't need to prove any relation about s. Consequently, this new approach is slightly more efficient than using ESS+.

AGHO Signature. Abe *et al.* [AGHO11] designed a structure-preserving signature scheme for bilinear groups that is secure under the SXDH assumption in the generic group model. The messages and signatures only consists of group elements and the verification of signatures consists of evaluating pairing product equations. So, the AGHO signature is compatible with the Groth-Sahai proof systems. In our divisible e-cash scheme we only require signatures on elements of $\mathbb{G}_1$. To get an efficient scheme, we take the rerandomizable signature scheme of [AGHO11, Section 5.3], but we interchange the elements of $\mathbb{G}_1$

and $\mathbb{G}_2$ to obtain a signature scheme for messages in $\mathbb{G}_1^L$ rather than $\mathbb{G}_2^L$. Under the SXDH assumption, the signature scheme is still secure ([Gro15]).

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$. The secret key is $\mathsf{sk} = (x, y_1, \ldots, y_L) \in \mathbb{Z}_p^{L+1}$ and the public key is $\mathsf{pk} = (X, Y_1, \ldots, Y_L) = (g^x, h^{y_1}, \ldots, h^{y_L}) \in \mathbb{G}_1 \times \mathbb{G}_2^L$. To sign a message $m = (M_1, \ldots, M_L) \in \mathbb{G}_1^L$, the signer randomly selects $r \in_R \mathbb{Z}_p^*$ and computes $A = h^r, B = A^x$ and $C = \left(gM_1^{-y_1} \cdots M_L^{-y_L}\right)^{1/r}$. The signature on the message is $\sigma = (A, B, C)$. Everyone can verify the equations $\hat{e}(X, A) \stackrel{?}{=} \hat{e}(g, B)$ and $\hat{e}(C, A)\,\hat{e}(M_1, Y_1) \cdots \hat{e}(M_L, Y_L) \stackrel{?}{=} \hat{e}(g, h)$.

For rerandomization, choose $r' \in_R \mathbb{Z}_p^*$ and compute the rerandomized signature $(A', B', C') = \left(A^{r'}, B^{r'}, C^{1/r'}\right)$.

To design our e-cash scheme, we just require messages for small L and don't need to hide this messages. Thus, in the random oracle model, we can either use the ESS or the AGHO signature scheme. However, we require compability with Groth-Sahai proof systems in the standard model. Consequently, we can't use ESS and so we (have to) use the AGHO signature scheme in the standard model. Hence, we also use this signature scheme in the random oracle model.

Furthermore, in the random oracle model, we observed that there is a more efficient method than generating a SoK to prove the relation $\hat{e}(\underline{V}, v_0) = \hat{e}(\underline{W_I}, v_I)$ as proposed in [CG10]. A user has just to compute $V' = V^{1/r}$ and $W' = W_I^{1/r}$ for a randomly chosen number $r \in_R \mathbb{Z}_p^*$. Everyone can verify $\hat{e}(V', v_0) = \hat{e}(W', v_I)$. However, the user has to prove knowledge of r with the result that an extractor is able to extract V and W_I. Thus, if the user anyway has to prove possession of a signature on the message V, the user can just generate a SoK to prove possession of a signature on the message V'^r (which is fully compatible with the AGHO signature scheme).

ElGamal Encryption. ElGamal [ElG85] proposed an encryption scheme which is semantically secure under the DDH assumption (see [TY98]).

The key pair is $(\mathsf{pk}, \mathsf{sk}) = ((g, g_1), x) \in \mathbb{G}^2 \times \mathbb{Z}_p^*$ such that $g_1 = g^x$ for a group $\mathbb{G}$ with prime order p. To encrypt a message $m \in \mathbb{G}$ under pk, randomly choose $r \in_R \mathbb{Z}_p$ and compute the ciphertext $c = (c_1, c_2) = (g^r, mg_1^r)$. The decryption is $m = c_2/c_1^x$. We use the ElGamal encryption scheme to extend our e-cash system to a fair e-cash scheme.

3 Syntax

Let $K = 2^L$ be the size of a wallet and let λ be the security parameter. A divisible e-cash system can be defined by the following polynomial time algorithms and protocols between the bank $\mathcal{B}$, the user $\mathcal{U}$ and the merchant $\mathcal{M}$:

- $\mathsf{Setup}\left(1^\lambda, L\right)$ is a probabilistic algorithm that outputs the system parameters sp. In the following, 1^λ and sp are implicitly in the input of all algorithms and protocols.
- $\mathsf{BKeyGen}()$ is a probabilistic algorithm that outputs a key pair $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B})$ for $\mathcal{B}$. An empty database $\mathcal{D}$ is set up.
- $\mathsf{UKeyGen}()$ is a probabilistic algorithm that outputs a key pair $(\mathsf{pk}_\mathcal{U}, \mathsf{sk}_\mathcal{U})$ for $\mathcal{U}$. A merchant $\mathcal{M}$ executes the same algorithm to get $(\mathsf{pk}_\mathcal{M}, \mathsf{sk}_\mathcal{M})$.
- $\mathsf{Withdrawal}\left[\mathcal{U}(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{U}, K) \leftrightarrow \mathcal{B}(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{U}, K)\right]$ is a protocol where $\mathcal{U}$ withdraws a wallet $\mathcal{W}$ containing K coins.
- $\mathsf{Spend}\left[\mathcal{U}(\mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{M}, \mathsf{sk}_\mathcal{U}, \mathcal{W}, \mathsf{ts}, k) \leftrightarrow \mathcal{M}(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{M}, \mathsf{ts}, k)\right]$ is a protocol where $\mathcal{U}$ spends a coin coin from the wallet $\mathcal{W}$ with monetary value k. $\mathcal{U}$ outputs an updated wallet $\mathcal{W}'$ and $\mathcal{M}$ outputs the coin coin. ts denotes the time stamp of the transaction.

- $\mathsf{Deposit}\,[\mathcal{M}\,(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{M}}, \mathsf{coin}) \leftrightarrow \mathcal{B}\,(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{M}})]$ is a protocol where $\mathcal{M}$ deposits a coin coin. If it concerns a double-spending, $\mathcal{B}$ executes the $\mathsf{Identify}$ algorithm. Otherwise, $\mathcal{B}$ outputs 1 and stores the coin coin in the database $\mathcal{D}$.

- $\mathsf{Identify}(\mathsf{coin}, \mathsf{coin}')$ is a deterministic algorithm that outputs the public key $\mathsf{pk}_{\mathcal{U}}$ of a fraudulent user or merchant.

- $\mathsf{VerifyGuilt}(\mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{coin}, \mathsf{coin}')$ is a deterministic algorithm that outputs 1 if the two coins are valid and the output of $\mathsf{Identify}(\mathsf{coin}, \mathsf{coin}')$ is $\mathsf{pk}_{\mathcal{U}}$, and 0 otherwise.

For correctness, we require that whenever an honest user obtains a wallet $\mathcal{W}$ from $\mathcal{B}$, an honest merchant shall accept the coin coin from $\mathcal{W}$. Whenever an honest merchant obtains a coin coin, the deposit of coin will be accepted by the honest bank.

3.1 Security Definitions

We first informally describe the security requirements of a divisible e-cash system.

- *Balance* guarantees that no collusion of users and merchants can deposit more coins than they have withdrawn from the bank without being identified.

  Thus, like in [ASM07, AWSM07], balance implies *unforgeability* and *identification of double-spenders* as defined in [CG07, CG10].

- *Anonymity* guarantees that no collusion of users, merchants and the bank can link a spent coin (that has not been double-spent) to other coins from the same wallet or can link a spent coin to its corresponding withdrawal protocol. We want that this property even holds if the user has double-spent some coins.

- *Strong Exculpability* guarantees that no collusion of users, merchants and the bank can falsely accuse an honest user from having double-spent a coin that he has not double-spent, even if the user has double-spent some other coins.

  This requirement corresponds to the *strong exculpability* definition in [CHL05, CHL06b] and the *non-frameability* definition in [Tro06]. However, in Section 3.1.1 we will explain that the formal definition in [CHL06b] has a mistake and that the formal definition in [Tro06] is not usable for divisible e-cash systems.

For a formal definition of security, we use a game-based approach. To ensure strong security properties, the adversary has more capabilities than in existing e-cash schemes (especially regarding anonymity). These capabilities are modeled by arbitrary and adaptive queries to the following oracles.

- $\mathcal{O}^U$. This oracle allows the adversary to add an honest or corrupted user in the system. If $\mathcal{A}$ supplies a public key $\mathsf{pk}_{\mathcal{A}}$, the oracle stores $\mathsf{pk}_{\mathcal{A}}$ in the set of corrupted users $\mathcal{U}_{\mathcal{A}}$. Else, $\mathcal{O}^U$ runs $\mathsf{UKeyGen}$, stores the key pair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}})$ in the set of honest users $\mathcal{U}_H$ and sends $\mathsf{pk}_{\mathcal{U}}$ to the adversary.

- $\mathcal{O}^W_{\mathcal{B}}$. This oracle allows the adversary to withdraw money from the bank. The adversary supplies a public key $\mathsf{pk}_{\mathcal{A}} \in \mathcal{U}_{\mathcal{A}}$ and performs a $\mathsf{Withdrawal}$ protocol with the honest bank to obtain a wallet $\mathcal{W}$.

- $\mathcal{O}^W_{\mathcal{U}}$. This oracle allows the adversary to perform a $\mathsf{Withdrawal}$ protocol with an honest user $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$ of the adversary's choice. After successful execution, an entry $(i, \mathcal{W}_i, \mathsf{pk}_{\mathcal{U}}, K)$ is added in a set $\mathcal{U}_{\mathcal{W}}$ and the counter i is then incremented by 1.

- $\mathcal{O}^W_{\mathcal{U},\mathcal{B}}$. This oracle allows the adversary to instruct an execution of Withdrawal protocol between an honest user $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$ of the adversary's choice and the honest bank. An entry $(i, \mathcal{W}_i, \mathsf{pk}_{\mathcal{U}}, K)$ is added in the set $\mathcal{U}_{\mathcal{W}}$ and the counter i is incremented by 1.

- $\mathcal{O}^S_{\mathcal{U}}$. This oracle allows the adversary to perform a Spend protocol with an honest user $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$ to obtain a coin coin with a legal monetary value k from a wallet $\mathcal{W}_i$ of the adversary's choice (such that the wallet $\mathcal{W}_i$ can spend a coin with monetary value k without over-spending the wallet). That is, the adversary chooses an index i such that $(i, \mathcal{W}_i, \mathsf{pk}_{\mathcal{U}}, \$_i) \in \mathcal{U}_{\mathcal{W}}$ and a monetary value $k \leq \$_i$. The adversary also chooses the merchant $\mathsf{pk}_{\mathcal{M}} \in \mathcal{U}_H \cup \mathcal{U}_A$ and the time stamp ts. After successful execution, the entry $(i, \mathcal{W}_i, \mathsf{pk}_{\mathcal{U}}, \$_i)$ is updated to $(i, \mathcal{W}'_i, \mathsf{pk}_{\mathcal{U}}, \$_i - k)$, where $\mathcal{W}'_i$ is the updated wallet. Further, an entry $(\mathsf{pk}_{\mathcal{U}}, \mathsf{coin})$ is added in a set $\mathcal{U}_{\mathsf{coin}}$.

- $\mathcal{O}^{S^\star}_{\mathcal{U}}$. This oracle is the same as $\mathcal{O}^S_{\mathcal{U}}$, but the adversary is allowed to choose $k > \$_i$ and to force the user $\mathsf{pk}_{\mathcal{U}}$ to over-spend his wallet $\mathcal{W}_i$, i.e. the user has to double-spend some coins. If the adversary chooses $k > \$_i$, first the state information of the wallet $\mathcal{W}_i$ is reset such that it is the same as it was before any coins were spent (see [CHL06b, Definition of (strong) Exculpability]).

- $\mathcal{O}^H$. In the random oracle model, this oracle allows the adversary to ask for the values of the hash functions at any time it wants in the following games.

Definition 4. (Game Balance)

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and runs* Setup *and* BKeyGen *to generate the system parameters* sp *and the key pair* $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}})$. *$\mathcal{C}$ sends* sp *and* $\mathsf{pk}_{\mathcal{B}}$ *to $\mathcal{A}$.*

- (Probing Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles $\mathcal{O}^U, \mathcal{O}^W_{\mathcal{B}}, \mathcal{O}^W_{\mathcal{U},\mathcal{B}}$ and $\mathcal{O}^S_{\mathcal{U}}$. Let q_W be the number of $\mathcal{O}^W_{\mathcal{B}}$ queries and $k^\star$ be the monetary value $\mathcal{A}$ obtains from $\mathcal{O}^S_{\mathcal{U}}$.*

- (End Game Phase.) *The adversary $\mathcal{A}$ outputs q tuples $(\mathsf{coin}_j, \mathsf{pk}_j, \mathsf{ts}_j, k_j)$ for $1 \leq j \leq q$ such that each coin_j is a valid coin of monetary value k_j, spent to merchant $\mathsf{pk}_j \in \mathcal{U}_H \cup \mathcal{U}_{\mathcal{A}}$ at time ts_j and such that $(\mathsf{pk}_j, \mathsf{ts}_j) \neq (\mathsf{pk}_{j'}, \mathsf{ts}_{j'})$ for all $j \neq j'$. The adversary $\mathcal{A}$ wins the game if $\sum_{j=1}^{q} k_j > K q_W + k^\star$ and* Identify *on any input $\mathsf{coin}, \mathsf{coin}' \in \{\mathsf{coin}_1, \ldots, \mathsf{coin}_q\}$ does not output any $\mathsf{pk}_{\mathcal{A}} \in \mathcal{U}_{\mathcal{A}}$.*

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins.

Definition 5. (Game Anonymity)

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and runs* Setup *to generate the system parameters* sp. *$\mathcal{C}$ sends* sp *to $\mathcal{A}$. The adversary returns a public key* $\mathsf{pk}_{\mathcal{B}}$.

- (Pre-Challenge Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles $\mathcal{O}^U, \mathcal{O}^W_{\mathcal{U}}$ and $\mathcal{O}^{S^\star}_{\mathcal{U}}$.*

- (Challenge Phase.) *The adversary $\mathcal{A}$ chooses two different indices i_0, i_1 such that $(i_0, \mathcal{W}_0, \mathsf{pk}_0, \$_0)$, $(i_1, \mathcal{W}_1, \mathsf{pk}_1, \$_1) \in \mathcal{U}_{\mathcal{W}}$ (where $\mathsf{pk}_0 = \mathsf{pk}_1$ is allowed), a merchant $\mathsf{pk}_{\mathcal{M}} \in \mathcal{U}_H \cup \mathcal{U}_{\mathcal{A}}$, a time stamp* ts *and a legal monetary value $k \leq \min\{\$_0, \$_1\}$ as challenge (such that both wallets $\mathcal{W}_0, \mathcal{W}_1$ can spend a coin with monetary value k without over-spending the wallet). The challenger $\mathcal{C}$ randomly chooses $b \in_R \{0, 1\}$ and performs the* Spend *protocol as user pk_b with wallet $\mathcal{W}_b$ to spend a coin with monetary value k to merchant $\mathsf{pk}_{\mathcal{M}}$.*

  After that, both entries get updated to $(i_0, \mathcal{W}'_0, \mathsf{pk}_0, \$_0 - k)$ resp. $(i_1, \mathcal{W}'_1, \mathsf{pk}_1, \$_1 - k)$, as if each of both wallets currently has spent a coin of monetary value k.

- (Post-Challenge Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles $\mathcal{O}^U, \mathcal{O}^W_{\mathcal{U}}, \mathcal{O}^S_{\mathcal{U}}$ and $\mathcal{O}^{S^\star}_{\mathcal{U}}$, but the oracle $\mathcal{O}^{S^\star}_{\mathcal{U}}$ cannot be asked on indices i_0, i_1. (Otherwise $\mathcal{A}$ can force the user pk_b to over-spend the wallet $\mathcal{W}_b$ that was used for challenge and can easily deduced which index was used during the challenge phase.)*

- (End Game Phase.) *The adversary $\mathcal{A}$ outputs a bit b' and wins the game if $b' = b$.*

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins minus $1/2$.

Definition 6. (Game Strong Exculpability)

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and runs* Setup *to generate the system parameters* sp. *$\mathcal{C}$ sends* sp *to $\mathcal{A}$. The adversary returns a public key* $\mathsf{pk}_{\mathcal{B}}$.

- (Probing Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles $\mathcal{O}^U, \mathcal{O}^W_{\mathcal{U}}$ and $\mathcal{O}^{S^\star}_{\mathcal{U}}$.*

- (End Game Phase.) *The adversary $\mathcal{A}$ outputs $(\mathsf{pk}_{\mathcal{U}}, \mathsf{coin}, \mathsf{coin}')$ such that $(\mathsf{pk}_{\mathcal{U}}, \mathsf{coin}) \notin \mathcal{U}_{\mathsf{coin}}$ or $(\mathsf{pk}_{\mathcal{U}}, \mathsf{coin}') \notin \mathcal{U}_{\mathsf{coin}}$ for $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$. The adversary $\mathcal{A}$ wins the game if* $\mathsf{VerifyGuilt}$ *on input $(\mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{coin}, \mathsf{coin}')$ outputs* 1.

The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins.

Definition 7. *A divisible e-cash system is* secure *if no PPT adversary $\mathcal{A}$ can win in Game Balance, Game Anonymity and Game Strong Exculpability with non-negligible advantage.*

3.1.1 Comparison of different Security Definitions

First, we explain why we require a new formal definition for strong exculpability.

- The formal definition in [CHL06b] has the mistake, that the following fact is not ruled out by this definition: A user applies one serial number for two different spendings (hence, the user double-spends this serial number), but the adversary successfully accuses the user of using this serial number for more than two spendings.

- The formal definition in [Tro06] is not usable for divisible e-cash systems because the following fact is not ruled out by this definition: A user double-spends a coin with monetary value k, but the adversary successfully accuses the user of double-spending a coin with monetary value greater than k.

Thus, the fact that "strong exculpability means that a guilty user would only be responsible for the coins that he indeed double-spent, i.e., his guilt can be quantified and he can be punished according to guilt" ([CHL06b]) is not achieved due to the definitions in [CHL06b] and [Tro06].

In [Tro06], Trolin pointed out that many security definitions do not rule out a corrupt bank cheating a user. Trolin argued that the withdrawal protocol should include a mechanism to solve the scenario that the bank claims that a user had withdrawn a wallet, but the user denies this. So, we require a security property that guarantees that no collusion of users, merchants and the bank can falsely accuse an honest user from having withdrawn a wallet that he has not withdrawn.

This property is called *exculpability* in [Tro06]. In [CG07], Canard and Gouget argued that this property is implied by the (weak) exculpability property of [CHL06b, CG07]. But this statement is false. As an example we take the divisible e-cash system in [ASM08]. This system fulfills the (weak) exculpability property (see [ASM08]). But during the withdrawal protocol, the user does not have to prove knowledge of his secret key. Thus, a corrupt bank can easily perform the users role in the withdrawal protocol and falsely accuse an honest user from having withdrawn a wallet. Moreover, a

corrupt bank can easily perform the users role in the withdrawal protocol and falsely accuse an honest user from trying to cheat (see InspectionRoutine in [ASM08] for more details). So, an honest user is found dishonest and has to pay a fine to the corrupt bank.

Nevertheless, it seems that each e-cash scheme that fulfills the following two conditions guarantees that no collusion of users, merchants and the bank can falsely accuse an honest user from having withdrawn a wallet:

1. During the Withdrawal protocol, a user generates a signature with respect to his public key $\mathsf{pk}_{\mathcal{U}}$. So, in the random oracle model, the user can easily generate a signature of knowledge instead of an interactive proof of knowledge (see Section 2.3).

2. The bank can't compute the secret key $\mathsf{sk}_{\mathcal{U}}$. Consequently, the Identify algorithm has to compute the public key $\mathsf{pk}_{\mathcal{U}}$ rather than the secret key $\mathsf{sk}_{\mathcal{U}}$ (such as in [AWSM07]).

If these two conditions are fulfilled, only the user with knowledge of the secret key $\mathsf{sk}_{\mathcal{U}}$ can perform a withdrawal protocol with respect to the public key $\mathsf{pk}_{\mathcal{U}}$.

Now, we have a solution for the scenario that the bank claims that a user had withdrawn a wallet, but the user denies this. If this scenario occurs, the bank has to publish its view of the withdrawal protocol. Everybody can check that the user indeed had withdrawn a wallet. Furthermore, it is a solution for the scenario that a corrupt bank doesn't send a valid signature to the user (see [Tro06, Section 3]). Because each view of the withdrawal protocol has to include a valid signature, the user obtains this signature.

Besides, since the bank has to store its views of withdrawal protocols, the e-cash system slightly becomes *auditable* in the sense of [STS99a, STS99b].

4 Divisible E-Cash Schemes

4.1 ASM Scheme

In [ASM08], Au *et al.* suggest a divisible e-cash scheme (see also [Au09, Section 5.5]). The main building blocks are the bounded accumulator and the ESS+ signature scheme presented in Section 2.3. Each user is in possession of a key pair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) \in \mathbb{G}_1 \times \mathbb{Z}_p^*$. To withdraw a wallet containing $K = 2^L$ coins, the user randomly selects a key root $\kappa_{0,0} \in_R \mathbb{Z}_p^*$ and generates a binary tree with $L+1$ levels as shown in Figure 1, where g is a generator of the group $\mathbb{G}_1$ and $H_0, H_1 \colon \{0,1\}^* \to \mathbb{Z}_p^*$ are two secure cryptographic hash functions.

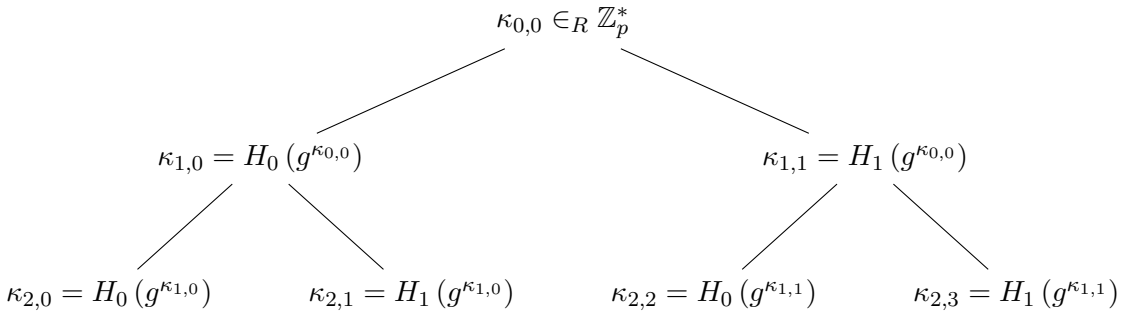

Figure 1: Binary tree in [ASM08] for $L = 2$

The Withdrawal protocol corresponds to $L+1$ executions of the issue protocol of ESS+ on $L+1$ messages $m_i = (V_i, \mathsf{sk}_{\mathcal{U}})$, where $V_i = \mathsf{Acc}(\{\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}\})$ is the accumulator of the 2^i keys $\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}$ in level i. The user sends the commitments of the messages to the bank and with probability $1/2$, the bank asks $\mathcal{U}$ to open the commitments and to reveal the whole binary tree. The bank checks whether all

values are honestly generated. If $\mathcal{U}$ is found dishonest, a fine is deducted from his account. Otherwise, the withdrawal protocol is repeated from the beginning. If $\mathcal{B}$ does not ask $\mathcal{U}$ to reveal the binary tree, the bank generates the corresponding $L+1$ ESS+ signatures $\sigma_1, \ldots, \sigma_{L+1}$. Thus, this e-cash system only fulfills a weak statistical balance property.

To spend a coin of value 2^ℓ, the user chooses an unused key $\kappa_{L-\ell,j}$ at level $L-\ell$. Then, $\mathcal{U}$ computes a serial number $S = g_S^{\kappa_{L-\ell,j}}$ and a security tag $T = \mathsf{pk}_{\mathcal{U}} g_T^{R\kappa_{L-\ell,j}}$, where $R \in \mathbb{Z}_p^*$ is a hash value and g_S, g_T are random generators of $\mathbb{G}_1$. Now, the user generates a signature of knowledge to prove that

1. S and T are computed correctly,
2. the key $\kappa_{L-\ell,j}$ is inside the accumulator $V_{L-\ell}$, and
3. the user is in possession of a signature $\sigma_{L-\ell}$ on message $(V_{L-\ell}, \mathsf{sk}_{\mathcal{U}})$.

The main drawback of this scheme is that it only fulfills a weak statistical balance property.

4.2 CG Scheme

In [CG10], Canard and Gouget present a divisible e-cash scheme that solves the problem of [ASM08]. The system is also based on ESS+ and bounded accumulators, but with a new technique to prove that several revealed values are inside an accumulator. To withdraw a wallet containing $K = 2^L$ coins, the user randomly selects a key root $\kappa_{0,0} \in_R \mathbb{Z}_p^*$ and generates a binary tree with $L+2$ levels as $\kappa_{i+1,2j} = \mathfrak{g}_0^{\kappa_{i,j} \pmod q}$ and $\kappa_{i+1,2j+1} = \mathfrak{g}_1^{\kappa_{i,j} \pmod q}$ for $0 \leq i \leq L$ and $0 \leq j \leq 2^i - 1$, where $\mathfrak{g}_0, \mathfrak{g}_1$ are generators of the subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p^*$ of order q, where q divides $p-1$.

The $\mathsf{Withdrawal}$ protocol corresponds to $L+2$ executions of the issue protocol of ESS+ on $L+2$ messages $m_0 = (V, \mathsf{sk}_{\mathcal{U}}, s)$ and $m_i = (V_i, s, i)$ for $1 \leq i \leq L+1$, where $V = \mathsf{Acc}(\{\kappa_{1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}\})$ is the accumulator of the whole binary tree except of $\kappa_{0,0}$ and each $V_i = \mathsf{Acc}(\{\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}\})$ is the accumulator of the 2^i keys $\kappa_{i,0}, \ldots, \kappa_{i,2^i-1}$ in level i. Next, the bank and the user interact using the ESS+ protocol in order to get the signatures $\sigma_0, \ldots, \sigma_{L+1}$. But here is a slight flaw in [CG10]. According to [CG10], the user sends the accumulators to the bank and then $\mathcal{B}$ produces the commitments. But, since the computation of the accumulators is deterministic, the bank can compute all accumulators after a user has spent a coin of value 2^L and thus $\mathcal{B}$ easily breaks the anonymity of the system. Hence, the user has to produce the commitments and has to send the commitments instead of the accumulators to the bank (as described in the ESS+ signature generation protocol in [ASM08, Appendix A] or in Section 2.3). Thus, each accumulator is perfectly hidden.

To spend a coin of value 2^ℓ, the user chooses an unused key $\kappa_{L-\ell,j}$ at level $L-\ell$. Then, $\mathcal{U}$ computes a serial number $S = \kappa_{L-\ell+1,2j} || \kappa_{L-\ell+1,2j+1} = \mathfrak{g}_0^{\kappa_L - \ell, j} || \mathfrak{g}_1^{\kappa_L - \ell, j}$ and a security tag $T = \mathsf{pk}_{\mathcal{U}} g_T^{R\kappa_{L-\ell,j}}$, where $R \in \mathbb{Z}_p^*$ is a hash value and g_T is random generator of $\mathbb{G}_1$. Now, the user sends S, T to the merchant and generates a signature of knowledge to prove that

1. S and T are computed correctly (this is a proof of equality of discrete logarithms in groups of distinct known prime order),
2. the keys $\kappa_{L-\ell+1,2j}, \kappa_{L-\ell+1,2j+1}$ and all their descendants are inside the accumulator V (hence, the merchant has to compute all descendants of S),
3. the keys $\kappa_{L-\ell+1,2j}$ and $\kappa_{L-\ell+1,2j+1}$ are inside the accumulator $V_{L-\ell+1}$,
4. the user is in possession of a signature σ_0 on message $(V, \mathsf{sk}_{\mathcal{U}}, s)$, and
5. the user is in possession of a signature $\sigma_{L-\ell+1}$ on message $(V_{L-\ell+1}, s, L-\ell+1)$.

4.2.1 Forgeability of [CG10]

In this section we present an attack against the balance property, resp. against the unforgeability property defined in [CG10]. The following Figure 2 shows the generation of the binary tree and the accumulators for $L = 2$.

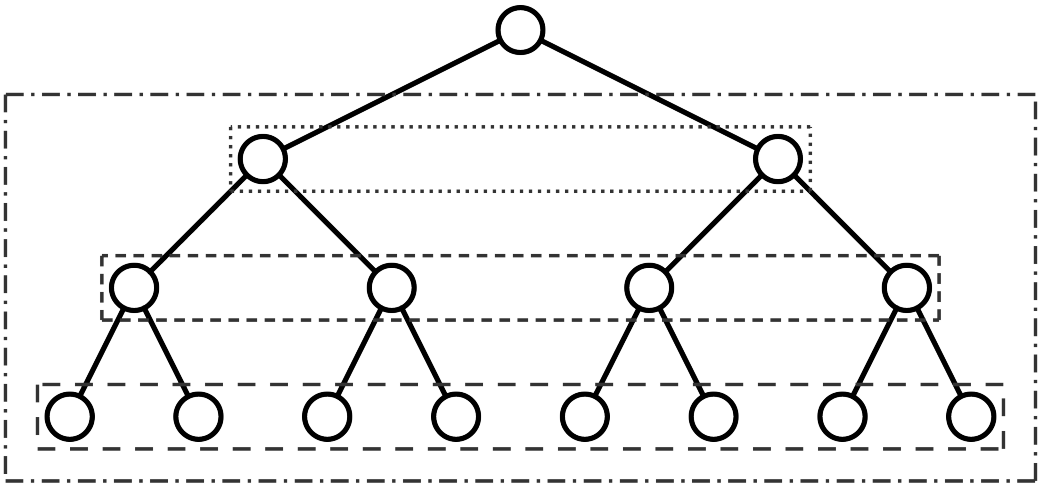

Figure 2: Binary tree and accumulators in [CG10] for $L = 2$

The keys in level $1, 2$ and 3 are inside the accumulators V_1, V_2 and V_3, respectively. Further, all keys in level $1, 2$ and 3 are inside the accumulator V. After computing the accumulators, the user produces the perfectly hiding commitments and sends the commitments to the bank. As a consequence, there is no chance for the bank to check whether or not the binary tree and the accumulators are generated honestly.

Now we present the attack to withdraw a wallet containing $1.5 \cdot 2^L$ coins for every $L \geq 2$. First, generate two independent binary trees, each with $L + 2$ levels. We denote the keys of the first tree as $\kappa_{i,j}$ and of the second tree as $\kappa'_{i,j}$ for $0 \leq i \leq L + 1$ and $0 \leq j \leq 2^i - 1$. Then, compute the accumulator $V_{L+1} = \mathsf{Acc}\left(\left\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}\right\}\right)$ with the keys of the first tree and the accumulators $V_i = \mathsf{Acc}\left(\left\{\kappa'_{i,0}, \ldots, \kappa'_{i,2^i-1}\right\}\right)$ with the keys of the second tree for $1 \leq i \leq L$. Especially, the two keys $\kappa'_{2,0}$ and $\kappa'_{2,1}$ are inside the accumulator V_2. The accumulators $V_1, V_3, \ldots, V_L$ will not be used for generating coins and they could also be computed in another way. Finally, compute the accumulator

$$V = \mathsf{Acc}\left(\left\{\kappa_{L+1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}\right\} \cup \bigcup_{i=2}^{L+1} \left\{\kappa'_{i,0}, \ldots, \kappa'_{i,2^{i-1}-i}\right\}\right).$$

Thus, V is the accumulator of the 2^{L+1} keys $\kappa_{L+1,0}, \ldots, \kappa_{L+1,2^{L+1}-1}$ in level $L + 1$ of the first tree and all $2^{L+1} - 2$ descendants of $\kappa'_{1,0}$ of the second tree. Since each accumulator is perfectly hidden, no one can detect this attack. The following Figure 3 shows the attack for $L = 2$.

The blackened nodes corresponds to the keys that can be used to generate valid coins. Hence, we can use the 2^L keys $\kappa_{L,0}, \ldots, \kappa_{L,2^L-1}$ to generate 2^L valid coins with monetary value 1 and can use the key $\kappa'_{1,0}$ to generate a valid coin with monetary value $2^{L-1} = 0.5 \cdot 2^L$. Consequently, we can produce valid coins with monetary value $1.5 \cdot 2^L$ instead of 2^L and thus we break the unforgeability of the scheme. Notice, that this is not the best attack but an example that works for all $L \geq 2$.

To prevent this attack, the bank can ask the user to reveal the whole binary tree and to open the commitments during the withdrawal protocol just as in [ASM08]. Consequently, the system will only fulfill statistical balance. Another approach is that during spending the user has to prove that each

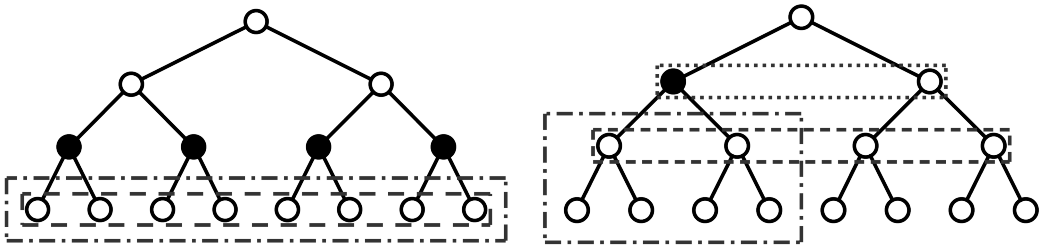

Figure 3: Attack against the unforgeability of [CG10] for $L = 2$

descendant from the serial number $S = \kappa_{L-\ell+1,2j}||\kappa_{L-\ell+1,2j+1}$ is inside the corresponding accumulator and that the user is in possession of an appropriate signature. As a consequence, the computational costs and the bandwidth of the spending phase will linearly depend on ℓ.

5 Our Construction

5.1 High Level Description

As said in Section 1.2, we combine features of the systems in [ASM08] and [CG10] to design the new divisible e-cash scheme. The binary tree and the serial numbers are computed as in [ASM08], but we use related hash functions and our binary tree has one more level. Let g be a generator of a cyclic group $\mathbb{G}_1$ and let H_0 and H_1 two hash functions with $H_0(m) := H(m||0)$ and $H_1(m) := H(m||1)$ for $m \in \{0,1\}^*$, where H is a secure cryptographic hash function $H\colon \{0,1\}^* \to \mathbb{Z}_p^*$. To generate the binary tree, the user randomly selects a key root $\kappa_{0,0} \in_R \mathbb{Z}_p^*$ and computes the keys of the tree as $\kappa_{i+1,2j} = H_0(g^{\kappa_{i,j}})$ and $\kappa_{i+1,2j+1} = H_1(g^{\kappa_{i,j}})$ for $0 \leq i \leq L-1, 0 \leq j \leq 2^i - 1$. Then, $\mathcal{U}$ computes the K so called serial keys $\kappa_j := \kappa_{L+1,j} = H(g^{\kappa_{L,j}})$ for $0 \leq j \leq K-1$. Figure 4 illustrates the construction of a binary tree with $L = 2$.

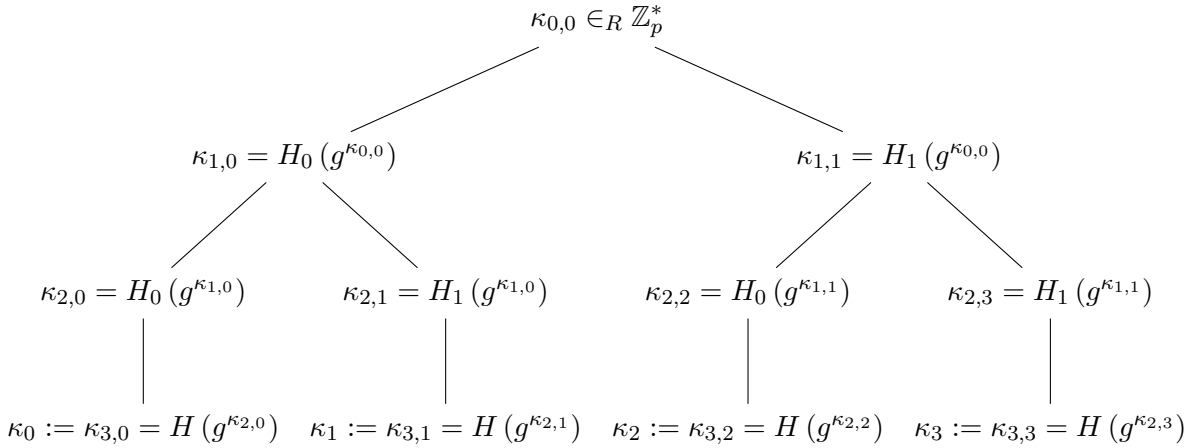

Figure 4: Construction of a binary tree ($L = 2$)

When a node key $\kappa_{i,j}$ is used, none of the descendant and ancestor nodes of $\kappa_{i,j}$ can be used, and no node can be used more than once.

Unlike the schemes in [ASM08] and [CG10], during the withdrawal protocol, only the K serial keys $\kappa_0, \ldots, \kappa_{K-1}$ are accumulated into one accumulator V. Thus, if the user just randomly selects the K serial keys $\kappa_0, \ldots, \kappa_{K-1} \in_R \mathbb{Z}_p^*$ instead of generating the binary tree, we will get a compact e-cash scheme

related to the one in [AWSM07]. To get a compact e-cash scheme that is secure in the standard model under the q-SDH and SXDH assumption, the user randomly selects K keys $\kappa_{L,0}, \ldots, \kappa_{L,K-1} \in_R \mathbb{Z}_p^*$ of level L and then computes each serial key as before as $\kappa_i := \kappa_{L+1,i} = H(g^{\kappa_{L,i}})$ for $0 \leq i \leq K-1$. However, with the use of the binary tree we can design an efficient divisible e-cash system. At the end of the withdrawal protocol, the user obtains one AGHO signature σ on message $(V, \mathsf{pk}_{\mathcal{U}})$.

To spend a coin with monetary value $2^\ell \leq 2^L$, the user chooses the unused key $\kappa_{L-\ell,j_0}$ at level $L-\ell$ with smallest index j_0. Then, $\mathcal{U}$ computes a serial number $S = g^{\kappa_{L-\ell,j_0}}$ and a security tag $T = \mathsf{pk}_{\mathcal{U}} g_1^{R\kappa_{L-\ell,j_0}}$, where $R \in \mathbb{Z}_p^*$ is a hash value (see also Section 7.4) and g_1 is a random generator of $\mathbb{G}_1$. Now, the user sends S, T to the merchant and generates a signature of knowledge to prove that

1. S and T are computed correctly,
2. the k (known) serial keys $\kappa_{L+1,kj_0}, \ldots, \kappa_{L+1,kj_0+k-1}$ are inside a secret accumulator V (with the technique presented in [CG10]), and
3. the user is in possession of a signature σ on message $(V, \mathsf{pk}_{\mathcal{U}})$.

5.2 A new Divisible E-Cash Scheme

Setup. Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$ for some λ bit prime p. Let g_1, u_0 be random generators of $\mathbb{G}_1$ and v_0 be a random generator of $\mathbb{G}_2$. The algorithm generates the elements $u_1, \ldots, u_K \in \mathbb{G}_1$ and $v_1, \ldots, v_K \in \mathbb{G}_2$ with $u_i = u_0^{\alpha^i}$ and $v_i = v_0^{\alpha^i}$ of the accumulator scheme. Let $H\colon \{0,1\}^* \to \mathbb{Z}_p^*$ be a secure cryptographic hash function. Define hash functions H_0 and H_1 as $H_0(m) := H(m||0)$ and $H_1(m) := H(m||1)$ for $m \in \{0,1\}^*$. If we are in the standard model, a common reference string $\mathsf{crs} = (g, g_2, g_3, g_4, h, h_1, h_2, h_3) \in \mathbb{G}_1^4 \times \mathbb{G}_2^4$ for the Groth-Sahai proofs system in the SXDH setting is also generated.[4] The system parameters are $\mathsf{sp} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \mathsf{crs}, g, g_1, u_0, \ldots, u_K, h, v_0, \ldots, v_K, H)$. This parameters could also include the element $\hat{e}(g,h) \in \mathbb{G}_T$. Since the random number $\alpha \in \mathbb{Z}_p^*$ is no longer needed, it should be deleted (see also [AWSM07, ASM08]).

BKeyGen. The bank generates an AGHO signature key pair $(\mathsf{pk}_{\mathcal{B}}, \mathsf{sk}_{\mathcal{B}}) = ((X, Y_1, Y_2), (x, y_1, y_2))$ and publishes $\mathsf{pk}_{\mathcal{B}}$. Further, $\mathcal{B}$ sets up a database $\mathcal{D}$. The bank can also publish the element $\hat{e}(g, Y_2) \in \mathbb{G}_T$. In that case, the element $Y_2 \in \mathbb{G}_2$ is not required in the random oracle model.

UKeyGen. To join the system, a user randomly selects a secret $u \in_R \mathbb{Z}_p^*$ and computes the public key as $\mathsf{pk}_{\mathcal{U}} := U = g^u$. The bank stores $\mathsf{pk}_{\mathcal{U}}$ as the unique identity of $\mathcal{U}$ in its database and the user stores the key pair $(\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{U}}) = (U, u)$.

Withdrawal. To withdraw a wallet containing $K = 2^L$ coins, the user and the bank perform the interactive Withdrawal protocol (see Figure 5), after they have authenticated each other.

Step 1: The user randomly selects a key root $\kappa_{0,0} \in_R \mathbb{Z}_p^*$ and computes the keys of the tree and the K serial keys as described in Section 5.1. The keys $\kappa_{1,0}, \ldots, \kappa_{L,K-1}$ (all keys except of $\kappa_{0,0}$) and the serial keys $\kappa_0, \ldots, \kappa_{K-1}$ are stored in two lists, $\mathcal{K}_0$ and $\mathcal{K}_1$, resp. Next, $\mathcal{U}$ accumulates all serial keys by computing $V = u_0^{s\prod_{j=0}^{K-1}(\alpha+\kappa_j)}$ for a random number $s \in_R \mathbb{Z}_p^*$ and sends V to the bank.

Step 2: The bank randomly chooses $r \in_R \mathbb{Z}_p^*$ and computes an AGHO signature $A = h^r, B = A^x, C = (g \cdot V^{-y_1} \cdot U^{-y_2})^{1/r}$. Then, $\mathcal{B}$ sends A, B, C to $\mathcal{U}$. (In the negligible case $C = 1$ the bank generates a new signature.)

[4] g and h are elements of crs. So, we need only six additional generators.

Step 3: The user verifies the signature and stores the wallet $\mathcal{W} = (\mathcal{K}_0, \mathcal{K}_1, \kappa_{0,0}, s, V, A, B, C)$.

Remark 1. It is possible to achieve a wallet complexity of $\mathcal{O}(\lambda + K)$ instead of $\mathcal{O}(\lambda \cdot K)$ by not storing the lists $\mathcal{K}_0$ and $\mathcal{K}_1$, but storing a K-bit string, that represents the used serial keys, and recomputing the binary tree from the key root $\kappa_{0,0}$ during each spending.

Remark 2. To achieve the security property mentioned in [Tro06] (see also Section 3.1.1), the user also sends a weak BB signature $\sigma' = u_0^{1/(u+H(U||V))}$ to the bank, where the element $h^u \in \mathbb{G}_2$ is also a part of the user public key $\mathsf{pk}_{\mathcal{U}}$. The bank then stores its view of the protocol $\mathsf{view} = (\mathsf{pk}_{\mathcal{U}}, V, \sigma', A, B, C)$. It is also possible that the bank only stores $(\mathsf{pk}_{\mathcal{U}}, V, \sigma')$ and computes another valid AGHO signature (A', B', C') in the case that the bank has to prove that the user indeed had withdrawn a wallet (as discussed in Section 3.1.1).

Remark 3. As already said, we get a compact e-cash scheme, if the user just randomly chooses the L-th level of the binary tree and then computes the serial keys as described.

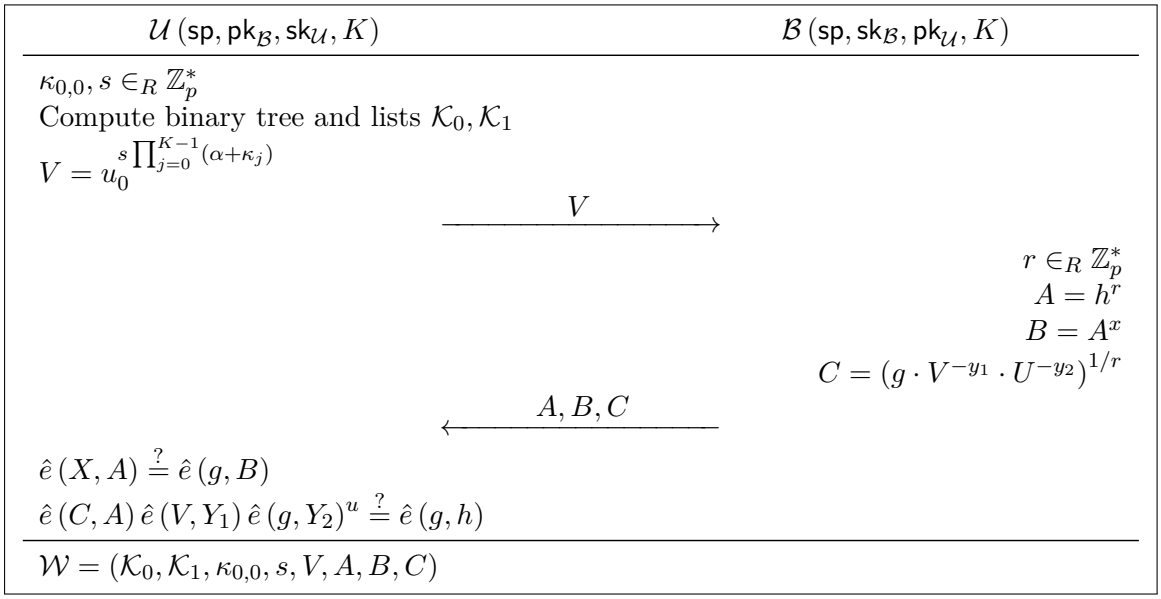

Figure 5: Withdrawal protocol

Spend. To spend a coin of monetary value $k = 2^\ell \leq 2^L$, the user and the merchant perform the Spend protocol (see Figure 6). First, the user and the merchant agree on transaction information which includes the monetary value $k = 2^\ell$, the time ts and the public key of the merchant $\mathsf{pk}_{\mathcal{M}}$. Then, both parties compute the hash value $R = H(\mathsf{pk}_{\mathcal{M}}||\mathsf{ts}||k)$. Each merchant is allowed to perform only one spend protocol at that time ts.[5]

Step 1: The user chooses the unused key $\kappa_{L-\ell,j_0} \in \mathcal{K}_0$ in level $L - \ell$ with smallest j_0. Let $\kappa := \kappa_{L-\ell,j_0}$ and $I := \{kj_0, \ldots, kj_0 + k - 1\}$. Then, $\mathcal{U}$ computes the serial number as $S = g^\kappa$ and the security tag as $T = U g_1^{R\kappa}$. Next, $\mathcal{U}$ computes the witness of the serial keys $\kappa_{kj_0}, \ldots, \kappa_{kj_0+k-1}$ as $W_I = u_0^{s \prod_{j=0, j \notin I}^{K-1}(\alpha+\kappa_j)}$. (If $k = 2^\ell = 2^L = K$, the witness W_I is computed as $W_I = u_0^s$.) Now, $\mathcal{U}$ has to prove that S and T are computed correctly, that all k serial keys $\kappa_{kj_0}, \ldots, \kappa_{kj_0+k-1}$ are accumulated in some V, and that he knows an AGHO signature on the message (V, U). So, the user generates a proof Φ as follows:

[5] That is in the interest of the merchant. Otherwise, a user could double-spent a coin to a merchant at the same time. Thus, the merchant possibly obtains two identical coins. As a consequence, the bank only will accept one of these coins because it seems that the merchant tries to deposit the same coin twice.

- **Random Oracle Model:** The user randomly chooses $r', r_1, r_2 \in_R \mathbb{Z}_p^*$, computes the elements $A' = A^{r'}, B' = B^{r'}, C'' = C^{1/(r'r_1)}, V' = V^{1/r_2}, W' = W_I^{1/r_2}$ and generates the following signature of knowledge:

$$\Pi = SoK\Big\{(u, \kappa, r_1, r_2) : S = g^{\kappa} \ \wedge \ T = g^u g_1^{R\kappa} \ \wedge \\ \hat{e}\left(g, h\right) = \hat{e}\left(C'', A'\right)^{r_1} \hat{e}\left(V', Y_1\right)^{r_2} \hat{e}\left(g, Y_2\right)^{u}\Big\}\left(S, T, A', C'', V', R\right).$$

  The proof is $\Phi = (A', B', C'', V', W', \Pi)$.

- **Standard Model:** As in [CPST15], the user randomly chooses $\mathsf{sk}_{ots} \in_R \mathbb{Z}_p^*$, sets $\mathsf{pk}_{ots} = h^{\mathsf{sk}_{ots}} \in \mathbb{G}_2$ and computes $\mu = u_0^{1/(u+H(\mathsf{pk}_{ots}))} \in \mathbb{G}_1$. Then, the user generates a rerandomized signature $(A', B', C') = \left(A^{r'}, B^{r'}, C^{1/r'}\right)$ for a random number $r' \in_R \mathbb{Z}_p^*$. Next, the user generates Groth-Sahai commitments to $u, \kappa \in \mathbb{Z}_p$ (2 elements of $\mathbb{G}_2$ each) and to $U, C', V, W_I, \mu \in \mathbb{G}_1$ (2 elements of $\mathbb{G}_1$ each). Let Com include all these commitments. Then, $\mathcal{U}$ generates the following NIZK proof π that the committed values satisfy:

$$g^{\underline{\kappa}} = S \ \wedge \ g^{\underline{u}}\left(g_1^R\right)^{\underline{\kappa}} = T \ \wedge \ g^{\underline{u}}\underline{U}^{-1} = 1 \ \wedge \ \underline{\mu}^{\underline{u}}\underline{\mu}^{H(\mathsf{pk}_{ots})} = u_0 \ \wedge \\ \hat{e}\left(\underline{V}, v_0\right)\hat{e}\left(\underline{W_I}, v_I^{-1}\right) = 1 \ \wedge \ \hat{e}\left(\underline{C'}, A'\right)\hat{e}\left(\underline{V}, Y_1\right)\hat{e}\left(\underline{U}, Y_2\right) = \hat{e}\left(g, h\right).$$

  The proof of the first two equations consists of 1 element of $\mathbb{G}_1$ each. The proof of the third and fourth equations consists of 2 elements of $\mathbb{G}_1$ and 4 elements $\mathbb{G}_2$ each. The proof of the fifth and sixth equations consists of 2 elements of $\mathbb{G}_2$ each.

  Next, the user computes the BB one-time signature $\eta = g^{\frac{1}{\mathsf{sk}_{ots}+H(S||T||A'||B'||\mathsf{Com}||\pi||R)}} \in \mathbb{G}_1$ (as in [CPST15]). The proof is $\Phi = (\mathsf{pk}_{ots}, A', B', \mathsf{Com}, \pi, \eta)$.

The user sends S, T along with the proof Φ to the merchant.

Finally, the user deletes the key $\kappa_{L-\ell,j_0}$ and all its ancestors and descendants in $\mathcal{K}_0$ (or updates the stored K-bit string). The updated wallet is $\mathcal{W}' = (\mathcal{K}_0', \mathcal{K}_1, \kappa_{0,0}, s, V, A, B, C)$, where $\mathcal{K}_0'$ is the updated list.

Step 2: The merchant computes the $k = 2^{\ell}$ serial keys κ_j for $j \in I := \left\{2^{\ell}j_0, \ldots, 2^{\ell}j_0 + 2^{\ell} - 1\right\}$ from the serial number S. Then, the merchant verifies the proof Φ as follows:

- **Random Oracle Model:** The merchant checks if $\hat{e}\left(X, A'\right) \stackrel{?}{=} \hat{e}\left(g, B'\right)$, if $\hat{e}\left(V', v_0\right) \stackrel{?}{=} \hat{e}\left(W', v_I\right)$, where $v_I = v_0^{\prod_{j\in I}(\alpha+\kappa_j)}$, and if the signature of knowledge Π is valid.
- **Standard Model:** The merchant checks if $\hat{e}\left(\eta, \mathsf{pk}_{ots}h^{H(S||T||A'||B'||\mathsf{Com}||\pi||R)}\right) \stackrel{?}{=} \hat{e}\left(g, h\right)$, if $\hat{e}\left(X, A'\right) \stackrel{?}{=} \hat{e}\left(g, B'\right)$ and if π is valid.

If the proof Φ is valid, the merchant accepts the payment and stores the coin $\mathsf{coin} = (k, S, T, R, \Phi, \mathsf{ts})$.

Deposit. The merchant sends a coin $\mathsf{coin} = (k, S, T, R, \Phi, \mathsf{ts})$ to the bank. The bank checks $R \stackrel{?}{=} H\left(\mathsf{pk}_{\mathcal{M}}||\mathsf{ts}||k\right)$ to verify that $\mathsf{pk}_{\mathcal{M}}$ is the real merchant. Then, the bank computes the $k = 2^{\ell}$ serial keys $\kappa_{L+1,2^{\ell}j_0}, \ldots, \kappa_{L+1,2^{\ell}j_0+2^{\ell}-1}$ from the serial number S and verifies Φ (as the merchant during spend protocol). After successful verification, the bank checks if at least one of the serial keys is already in its database $\mathcal{D}$. If so, the bank runs the identification algorithm $\mathsf{Identify}$ to identify the double-spender. Else, $\mathcal{B}$ adds coin and the k serial keys in $\mathcal{D}$.

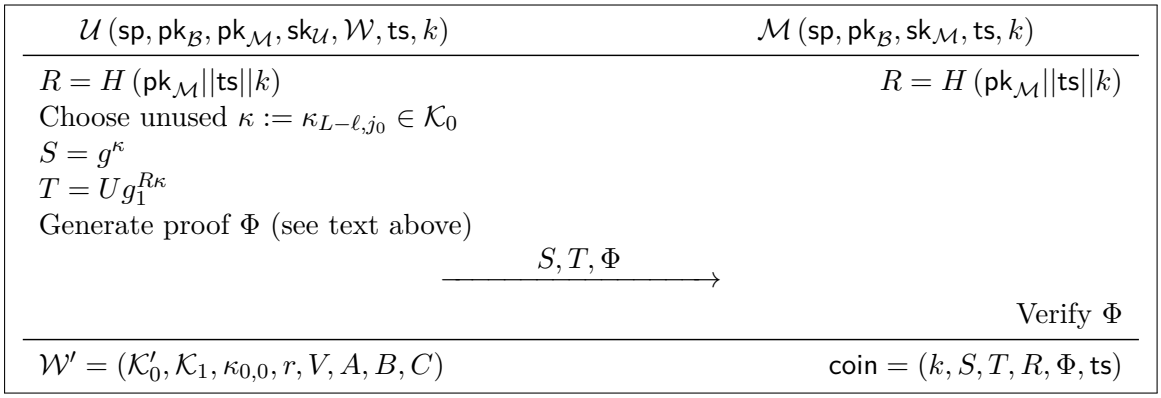

Figure 6: Spend protocol

Identify. Let $\mathsf{coin} = (k, S, T, R, \Phi, \mathsf{ts})$ and $\mathsf{coin}' = (k', S', T', R', \Phi', \mathsf{ts}')$ be the coins where a double spending occurred and $\mathsf{pk}_{\mathcal{M}}$ the merchant who deposits a coin. The bank checks if $R \stackrel{?}{=} R'$. If so, the merchant $\mathsf{pk}_{\mathcal{M}}$ tries to deposit the same coin twice since the hash function H is collision-resistant. Else, $\mathcal{B}$ computes the public key of the double-spender as follows:

- If $k = 2^{\ell} = 2^{\ell'} = k'$, then we have $S = S'$ and hence the same key κ has been used in T and T'. Thus, $\mathcal{B}$ computes the public key of the double-spender as

$$\left(\frac{T^{R'}}{T'^{R}}\right)^{1/(R'-R)} = \left(\frac{U^{R'} g_1^{RR'\kappa}}{U^{R} g_1^{RR'\kappa}}\right)^{1/(R'-R)} = \left(U^{R'-R}\right)^{1/(R'-R)} = U.$$

- If $k = 2^{\ell} \neq 2^{\ell'} = k'$, without loss of generality, assume $\ell > \ell'$. Consequently, S' is a descendant from S. Hence, $\mathcal{B}$ computes the key κ' with $S' = g^{\kappa'}$ and $T' = Ug_1^{R'\kappa'}$ from the serial number S. Thus, $\mathcal{B}$ computes the public key of the double-spender as

$$\frac{T'}{g_1^{R'\kappa'}} = \frac{Ug_1^{R'\kappa'}}{g_1^{R'\kappa'}} = U.$$

VerifyGuilt. The bank publishes the two double-spent coins and the identity U of the double-spender. Everyone can verify the two coins and execute $\mathsf{Identify}$ to verify that U is the double-spender.

5.3 Security Analysis

We have the following theorems for our new compact and divisible e-cash systems.

Theorem 1. *In the random oracle model, our compact and our divisible e-cash system are secure under the q-SDH assumption and the SXDH assumption.*

Theorem 2. *In the standard model, assuming that the hash function H is collision-resistant and the output of H is indistinguishable from the uniform distribution, our divisible e-cash system is secure under the q-SDH assumption and the SXDH assumption.*

Theorem 3. *In the standard model, assuming that the hash function H is collision-resistant, our compact e-cash system is secure under the q-SDH assumption and the SXDH assumption.*

Proof. We prove Strong Exculpability, Balance and Anonymity.

Strong Exculpability. Let $\mathcal{A}$ be a PPT adversary. We show that the advantage of $\mathcal{A}$ in winning Game Strong Exculpability is negligible under the q-SDH assumption, by constructing a reduction $\mathcal{R}$ acting as challenger $\mathcal{C}$. As in [CPST15], we distinguish the proof in the random oracle model from the one in the standard model.

- **Random Oracle Model:** The adversary $\mathcal{A}$ has to output an honest user $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$ and two valid coins $\mathsf{coin} = (k, S, T, R, \Phi, \mathsf{ts})$ and $\mathsf{coin}' = (k', S', T', R', \Phi', \mathsf{ts}')$ such that $(\mathsf{pk}_{\mathcal{U}}, \mathsf{coin}) \notin \mathcal{U}_{\mathsf{coin}}$ or $(\mathsf{pk}_{\mathcal{U}}, \mathsf{coin}') \notin \mathcal{U}_{\mathsf{coin}}$. Hence, at least one of this coins was not produced by $\mathcal{R}$. Both coins contain a signature of knowledge, which involves proving knowledge of the user secret $\mathsf{sk}_{\mathcal{U}}$ of an honest user $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$. Thus, $\mathcal{A}$ only wins the game, if it fakes the knowledge of T resp. T' which involves knowledge of $u = \log_g U$, what is negligible under the discrete logarithm assumption and consequently negligible under the q-SDH assumption.

- **Standard Model:** As in [CPST15], there are two kinds of attacks:

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

  As in [CPST15], a Type-1 attack is negligible under the security of the one-time signature scheme and thus negligible under the q-SDH assumption. So, we only have to consider a Type-2 attack.

  The reduction $\mathcal{R}$ gets the global parameters $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g, h)$ and a q-SDH challenge $\left(g, g^x, g^{x^2}, \ldots, g^{x^{q_S}}, h, h^x\right)$ as input, where $q = q_S$ is the bound on the number of $\mathcal{O}^S_{\mathcal{U}}$ queries.

  The reduction $\mathcal{R}$ randomly chooses generators $g_1 \in_R \mathbb{G}_1, v_0 \in_R \mathbb{G}_2$ and generates a common reference string crs for the soundness setting such that $\mathcal{R}$ knows the related discrete logarithms of the elements in $\mathbb{G}_2$ (i.e. let $(h, h_1, h_2, h_3) \in \mathbb{G}_2^4$ be the part of crs, then $\mathcal{R}$ randomly chooses $a, b \in_R \mathbb{Z}_p^*$ and computes $h_1 = h^a, h_2 = h^b, h_3 = h^{ab}$). Further, $\mathcal{R}$ randomly selects $1 \leq i^* \leq q_U$, where q_U is the bound on the number of $\mathcal{O}^U$ queries. The reduction honestly generates q_S key pairs $\left(\mathsf{pk}^{(1)}_{ots}, \mathsf{sk}^{(1)}_{ots}\right), \ldots, \left(\mathsf{pk}^{(q_S)}_{ots}, \mathsf{sk}^{(q_S)}_{ots}\right)$ and sets $u_0 := g^{r' \prod_{i=1}^{q_S} \left(x + H\left(\mathsf{pk}^{(i)}_{ots}\right)\right)}$ for a randomly chosen $r' \in_R \mathbb{Z}_p^*$.

  The reduction $\mathcal{R}$ sends $\mathsf{sp}' := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \mathsf{crs}, g, g_1, u_0, h, v_0)$ to the adversary. To show that the knowledge of the secret number α doesn't help $\mathcal{A}$ to win the game, it is allowed to generate $u_1, \ldots, u_K \in \mathbb{G}_1$ and $v_1, \ldots, v_K \in \mathbb{G}_2$. Further, the adversary generates $\mathsf{pk}_{\mathcal{B}} = (X, Y_1, Y_2)$.

  Each $\mathcal{O}^U$ query will be simulated honestly by $\mathcal{R}$ to add honest user and merchants for $i \neq i^*$. Otherwise, $\mathcal{R}$ will set $\mathsf{pk}^*_{\mathcal{U}} := g^x$. Further, each $\mathcal{O}^W_{\mathcal{U}}$ query and each $\mathcal{O}^{S^*}_{\mathcal{U}}$ query on $\mathsf{pk}_{\mathcal{U}} \neq \mathsf{pk}^*_{\mathcal{U}}$ will be simulated honestly by $\mathcal{R}$. For the j-th $\mathcal{O}^W_{\mathcal{U}}$ query on $\mathsf{pk}^*_{\mathcal{U}}$, the reduction computes $\mu = g^{r' \prod_{i=1, i \neq j}^{q_S} \left(x + H\left(\mathsf{pk}^{(i)}_{ots}\right)\right)}$ which verifies $\mu = u_0^{1/\left(x + H\left(\mathsf{pk}^{(j)}_{ots}\right)\right)}$. Then, $\mathcal{R}$ computes S, T, A', B' honestly and generates all commitments and the NIZK proof π honestly, which is possible since $\mathcal{R}$ has only to know the elements $h^x, h_2^x = (h^x)^b$ and $h_3^x = (h^x)^{ab}$ (see [GS10]). Finally, $\mathcal{R}$ computes $\eta = g^{\frac{1}{\mathsf{sk}_{ots} + H\left(S||T||A'||B'||\mathsf{Com}||\pi||R\right)}}$.

  The adversary $\mathcal{A}$ outputs an honest user $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$ and two valid coins $\mathsf{coin} = (k, S, T, R, \Phi, \mathsf{ts})$ and $\mathsf{coin}' = (k', S', T', R', \Phi', \mathsf{ts}')$. As explained above, at least one coin was not produced by $\mathcal{R}$. If $\mathsf{pk}_{\mathcal{U}} \neq \mathsf{pk}^*_{\mathcal{U}}$ the reduction aborts. Else, $\mathcal{R}$ extracts from the proof π of the forged coin an element $\mu = u_0^{1/(x + H(\mathsf{pk}_{ots}))}$, where $H\left(\mathsf{pk}_{ots}\right) \notin \left\{H\left(\mathsf{pk}^{(1)}_{ots}\right), \ldots, H\left(\mathsf{pk}^{(q_S)}_{ots}\right)\right\}$, since we consider a Type-2 attack. Thus, $\mathcal{R}$ can break the q-SDH assumption for $q = q_S$ (as in [BB08]) since $\mathcal{R}$ will not abort with probability $1/q_U$.

Consequently, the adversary can only create a valid coin corresponding to an honest user $\mathsf{pk}_{\mathcal{U}} \in \mathcal{U}_H$ with negligible probability under the q-SDH assumption. (This statement will be used to prove balance.)

Balance. Let $\mathcal{A}$ be a PPT adversary, let q_W be the number of $\mathcal{O}_{\mathcal{B}}^{W}$ queries, q'_W be the number of $\mathcal{O}_{\mathcal{U},\mathcal{B}}^{W}$ queries and $k^\star$ be the monetary value $\mathcal{A}$ obtains from $\mathcal{O}_{\mathcal{U}}^{S}$. We show that the advantage of $\mathcal{A}$ in winning Game Balance is negligible under the q-SDH assumption and the SXDH assumption, by constructing a reduction $\mathcal{R}$ acting as challenger $\mathcal{C}$.

The reduction $\mathcal{R}$ gets the global parameters $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, g, h)$, the public parameters of the accumulator scheme $(u_0, \ldots, u_K, v_0, \ldots, v_K)$, the generator $g_1 \in \mathbb{G}_1$ and the public key of AGHO signature scheme (X, Y_1, Y_2) as input. In the standard model, the reduction $\mathcal{R}$ generates a common reference string crs for the soundness setting.

Then, $\mathcal{R}$ sets $\mathsf{sp} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \mathsf{crs}, g, g_1, u_0, \ldots, u_K, h, v_0, \ldots, v_K)$ and $\mathsf{pk}_{\mathcal{B}} := (X, Y_1, Y_2)$ and sends the simulated parameters $\mathsf{sp}, \mathsf{pk}_{\mathcal{B}}$ to the adversary.

Each $\mathcal{O}^{U}$ query will be simulated honestly by $\mathcal{R}$ to add honest user and merchants. For each $\mathcal{O}_{\mathcal{B}}^{W}$ query, $\mathcal{R}$ invokes the AGHO signing oracle to obtain an AGHO signature (A, B, C) on the message (V, U) and sends (A, B, C) to $\mathcal{A}$. For each $\mathcal{O}_{\mathcal{U},\mathcal{B}}^{W}$ query, $\mathcal{R}$ acts like the honest user, invokes the AGHO signing oracle to obtain an AGHO signature (A, B, C) on the message (V, U), stores an entry $(i, \mathcal{W}_i = (\mathcal{K}_0, \mathcal{K}_1, \kappa_{0,0}, s, V, A, B, C), \mathsf{pk}_{\mathcal{U}}, K)$ in the set $\mathcal{U}_{\mathcal{W}}$ and increases the counter i by 1. For each $\mathcal{O}_{\mathcal{U}}^{S}$ query, $\mathcal{R}$ honestly computes the coin coin and updates the corresponding entry to $(i, \mathcal{W}'_i = (\mathcal{K}'_0, \mathcal{K}_1, \kappa_{0,0}, s, V, A, B, C), \mathsf{pk}_{\mathcal{U}}, \$_i - k)$. Let Coins be the set of all coins spent by honest users.

Finally, $\mathcal{A}$ outputs q valid coins $\mathsf{coin}_1, \ldots, \mathsf{coin}_q$ with monetary value at least $Kq_W + k^\star + 1$. The reduction $\mathcal{R}$ generates all corresponding serial keys. Let $\mathsf{Coins}^* := \{\mathsf{coin}_1, \ldots, \mathsf{coin}_q\} \setminus \mathsf{Coins}$ be the set of all coins that are generated by the adversary $\mathcal{A}$. For each such coin $\mathsf{coin}_i \in \mathsf{Coins}^*$, the reduction $\mathcal{R}$ extracts the elements $U_i, C'_i, V_i, W_{I,i}$ as follows:

- **Random Oracle Model:** The reduction extracts the values $u_i, \kappa_i, r_{1,i}, r_{2,i} \in \mathbb{Z}_p$ (this is possible due to the proof of knowledge property of Π in the random oracle model). Then, $\mathcal{R}$ computes $U_i := g^{u_i}, C'_i := (C''_i)^{r_{1,i}}, V_i := (V'_i)^{r_{2,i}}$ and $W_{I,i} := (W'_i)^{r_{2,i}}$.
- **Standard Model:** The reduction extracts the elements $U_i, C'_i, V_i, W_{I,i}, \mu_i$ from the Groth-Sahai commitments (this is possible due to the perfect soundness of the Groth-Sahai proof systems).

Next, $\mathcal{R}$ stores an entry $((V_i, U_i), (A'_i, B'_i, C'_i))$ in a list $\mathbb{L}_{\mathsf{Sign}}$ and an entry $(V_i, W_{I,i}, \kappa_{i,0}, \ldots, \kappa_{i,k_i-1})$ in a list $\mathbb{L}_{\mathsf{Acc}}$, where $\kappa_{i,0}, \ldots, \kappa_{i,k_i-1}$ are the k_i serial keys of the coin coin_i. Under the q-SDH assumption, we have $U_i \in \mathcal{U}_H$ for any i only with negligible probability (see Strong Exculpability). Consequently, none of the messages $(V_i, U_i) \in \mathbb{L}_{\mathsf{Sign}}$ was signed during a $\mathcal{O}_{\mathcal{U},\mathcal{B}}^{W}$ query.

$\mathcal{A}$ wins the game either if (1) all serial keys are unique or if (2) some of the serial keys are duplicated but the $\mathsf{Identify}$ algorithm does not output any $\mathsf{pk}_{\mathcal{A}} \in \mathcal{U}_A$. We will discuss both cases separately.

(1). Since $\mathcal{A}$ only obtains $k^\star$ serial keys from $\mathcal{O}_{\mathcal{U}}^{S}$, $\mathcal{A}$ has to produce another $Kq_W + 1$ serial keys. Each of the q_W $\mathcal{O}_{\mathcal{B}}^{W}$ queries only gives $\mathcal{A}$ one signature on one accumulator. As a consequence, $\mathcal{A}$ only obtained q_W signatures on at most q_W accumulators $V_1, \ldots, V_{q_W}$, where each accumulator gathers at most K serial keys. Hence, $\mathcal{A}$ obtained at most Kq_W valid serial keys from $\mathcal{O}_{\mathcal{B}}^{W}$. Thus, to produce more than Kq_W unique serial keys, $\mathcal{A}$ either (1) has to forge an AGHO signature on a new message $(V^\star, U^\star)$ such that $V^\star \notin \{V_1, \ldots, V_{q_W}\}$ or (2) at least one of the accumulators $V' \in \{V_1, \ldots, V_{q_W}\}$ is an accumulator of more than K serial keys. Consequently, $\mathcal{R}$ can output a forged message signature pair $((V^\star, U^\star), (A^\star, B^\star, C^\star)) \in \mathbb{L}_{\mathsf{Sign}}$ and hence breaks the security of the AGHO signature scheme, or can output an accumulator V' together with more than K serial keys and witnesses (all values are stored in $\mathbb{L}_{\mathsf{Acc}}$) which prove that all serial keys are inside the accumulator and hence breaks the security of the bounded accumulator scheme (like in [ASM08] and [CG10]), which only happens with negligible probability under the SXDH assumption and the q-SDH assumption, resp.

(2). Let $\mathsf{coin} = (k, S, T, R, \Phi, \mathsf{ts})$ and $\mathsf{coin}' = (k', S', T', R', \Phi', \mathsf{ts}')$ be two double-spent coins and $\mathsf{pk}_{\mathcal{M}}$ resp. $\mathsf{pk}'_{\mathcal{M}}$ be the corresponding merchant. Since we demand $(\mathsf{pk}_{\mathcal{M}}, \mathsf{ts}) \neq (\mathsf{pk}'_{\mathcal{M}}, \mathsf{ts}')$ and the values R and R' are the output of a collision-resistant hash function, they shall be different. Similarly, $\mathcal{A}$ shall not know different serial numbers that lead to the same serial key (otherwise $\mathcal{A}$ has found a collision of

the hash function H). So, if the security tags are correctly computed as $T = Ug_1^{R\kappa}$ and $T' = Ug_1^{R'\kappa'}$ regarding to the serial numbers $S = g^{\kappa}$ and $S' = g^{\kappa'}$, the correct identity $U \in \mathcal{U}_A$ will be computed by $\mathsf{Identify}$. The two security tags are the only valid tags to accompany serial numbers. To deviate from these valid tags, $\mathcal{A}$ has to forge an AGHO signature on a new message $(V^\star, U^\star)$ such that $U^\star \notin \mathcal{U}_\mathcal{A}$. Consequently, $\mathcal{R}$ can output a forged message signature pair $((V^\star, U^\star), (A^\star, B^\star, C^\star)) \in \mathbb{L}_{\mathsf{Sign}}$ and hence breaks the security of the AGHO signature scheme, which only happens with negligible probability under the SXDH assumption.

Thus, $\mathcal{A}$ only wins the game with negligible probability under the SXDH assumption and the q-SDH assumption.

Anonymity. Let $\mathcal{A}$ be a PPT adversary. We show that the advantage of $\mathcal{A}$ in winning Game Anonymity is negligible under the DDH assumption in $\mathbb{G}_1$ (what is replaced by the SXDH assumption), by constructing a reduction $\mathcal{R}$ acting as challenger $\mathcal{C}$.

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with parameters $\mathcal{G}$. The reduction gets $\mathcal{G}$ and a DDH instance $(g^a, g^b, g^c) \in \mathbb{G}_1^3$ as input. The task of $\mathcal{R}$ is to decide whether $g^c = g^{ab}$ or not.

The reduction $\mathcal{R}$ randomly chooses generators $u_0 \in_R \mathbb{G}_1, v_0 \in_R \mathbb{G}_2$ and defines $g_1 := g^a$. In the standard model, $\mathcal{R}$ also generates a common reference string crs for the witness-indistinguishability setting. The reduction $\mathcal{R}$ sends $\mathsf{sp}' := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, \hat{e}, \mathsf{crs}, g, g_1, u_0, h, v_0)$ to the adversary. To show that the knowledge of the secret number α doesn't help $\mathcal{A}$ to win the game, it is allowed to generate $u_1, \ldots, u_K \in \mathbb{G}_1$ and $v_1, \ldots, v_K \in \mathbb{G}_2$. Further, the adversary generates $\mathsf{pk}_\mathcal{B} = (X, Y_1, Y_2)$.

For each oracle query, the reduction just behaves like the honest party.

Assume, the adversary chooses two indices i_0, i_1 for challenge such that $\mathsf{pk}_0 \neq \mathsf{pk}_1$. For the challenge spending, the reduction randomly chooses $U \in_R \{\mathsf{pk}_0, \mathsf{pk}_1\}$ and defines $S := g^b$ and $T := U\,(g^c)^R$. Next, $\mathcal{R}$ randomly chooses $r' \in_R \mathbb{Z}_p^*$ and computes the perfectly simulated elements $A' = (A^*)^{r'}$ and $B' = (B^*)^{r'}$, where (A^*, B^*, C^*) is any valid signature which $\mathcal{R}$ obtained from $\mathcal{O}_\mathcal{U}^W$ (and $\mathcal{R}$ must have obtained at least one valid pair (A^*, B^*)). Finally, $\mathcal{R}$ perfectly simulates the proof Φ as follows:

- **Random Oracle Model:** The reduction $\mathcal{R}$ randomly chooses $r_1, r_2 \in_R \mathbb{Z}_p^*$ and computes the perfectly simulated elements $C'' = g^{r_1}, W' = u_0^{r_2}, V' = u_0^{r_2 \prod_{j=0}^{k-1}(\alpha+\kappa_j)}$, where $\kappa_0, \ldots, \kappa_{k-1}$ are the k serial keys computed from S. Further, $\mathcal{R}$ perfectly simulates the signature of knowledge Π. In the random oracle model, the serial number S cannot be linked to other serial numbers (except its ancestors and descendants). Hence, the simulation of S is perfect in the random oracle model for a random number $b =: \kappa$ (since no double-spending occurs).

- **Standard Model:** The reduction $\mathcal{R}$ perfectly simulates π and computes pk_{ots}, η honestly. In the standard model, the simulation of S is indistinguishable from a real serial number, if the output of H is indistinguishable from the uniform distribution.

- **Compact E-Cash:** The reduction $\mathcal{R}$ perfectly simulates Φ as in the random oracle model or in the standard model. For the compact e-cash scheme, the simulation of S is perfect in both models since it is possible that the user has chosen b during the withdrawal protocol.

If $g^c = g^{ab}$, the security tag $T = Ug^{abR} = Ug_1^{R\kappa}$ is perfectly simulated. If g^c is just a random element in $\mathbb{G}_1$, we can interpret it as $g^c = g^{ab+\delta}$ for an unknown, random number $\delta \in_R \mathbb{Z}_p^*$. Hence, the security tag looks like $T = Ug^{abR+\delta R} = U'g_1^{R\kappa}$ for a random user $U' := Ug^{R\delta}$. Thus, using the adversary as a black-box, $\mathcal{R}$ can solve the DDH problem.

It follows, that if the adversary chooses $\mathsf{pk}_0 = \mathsf{pk}_1$ for challenge, the advantage of $\mathcal{A}$ will be negligible (it will be equal to zero in the random oracle model). □

6 Efficiency Analysis

Table 1 compares the general efficiency of the e-cash schemes in [ASM08], [CG10], [CPST15] and our new construction (each implemented in the random oracle model) for withdrawing a monetary value of $K = 2^L$ and spending and depositing a monetary value of $k = 2^\ell$. This table can be seen as an extension of [CG10, Table 1] and [CPST15, Figure 5], but only with schemes in the random oracle model. First, we note that in [CPST15] the $(K(L+1))$ elements $\widetilde{g}_{s \mapsto f} \in \mathbb{G}_2$ (see [CPST15] for details) must be contained in the public system parameters sp, because otherwise the guilt of double-spenders can't be verified.

	[ASM08]	[CG10]	[CPST15]	Our Construction
sp	$\mathcal{G}$ + $\mathsf{pk}_{\mathsf{Com}}$ + $\mathsf{pk}^{(1)}_{\mathsf{Acc}_{2^0}}$ + $\cdots$ + $\mathsf{pk}^{(1)}_{\mathsf{Acc}_{2^L}}$	$\mathcal{G}$ + $\mathsf{pk}_{\mathsf{Com}}$ + $\mathsf{pk}^{(2)}_{\mathsf{Acc}_{2^1}}$ + $\cdots$ + $\mathsf{pk}^{(2)}_{\mathsf{Acc}_{2^{L+1}}}$ + $\mathsf{pk}^{(4K-2)}_{\mathsf{Acc}_{2^{L+2}-2}}$	$\mathcal{G}$ + $\mathsf{pk}_{\mathsf{Com}}$ + 2 $\mathsf{pk}_{\mathsf{Tree}_L}$ + $(K(L+1))$ $\mathbb{G}_2$	$\mathcal{G}$ + $\mathsf{pk}_{\mathsf{Com}}$ + $\mathsf{pk}^{(K)}_{\mathsf{Acc}_{2^L}}$
$\mathsf{pk}_{\mathcal{B}}$	$\mathsf{pk}_{\mathsf{Sign}}$	$\mathsf{pk}_{\mathsf{Sign}}$	$(L+2)$ $\mathsf{pk}_{\mathsf{Sign}}$ + $(2K-1)$ $\lvert\mathsf{Sign}\rvert$	$\mathsf{pk}_{\mathsf{Sign}}$
Withdrawal Computations $\mathcal{U}$ser	$(2K-2)$ E + 2 $(\mathsf{Acc}_{2^0} + \cdots + \mathsf{Acc}_{2^L})$ + $(2L+2)$ Com + SoK $\{(L+1)$ Com$\}$ + $(L+1)$ VerifySign	$(4K-2)$ E + 1 Acc_{2^1} + $\cdots$ + 1 $\mathsf{Acc}_{2^{L+1}}$ + 1 $\mathsf{Acc}_{2^{2+L}-2}$ + $(L+2)$ Com + SoK $\{(L+2)$ Com$\}$ + $(L+2)$ VerifySign	1 Com + SoK $\{1$ Com$\}$ + 1 VerifySign	$(2K-1)$ E + 1 Acc_{2^L} + 1 VerifySign
Withdrawal Computations $\mathcal{B}$ank	$(K-1)$ E + 1 Acc_{2^0} + $\cdots$ + 1 Acc_{2^L} + VerifySoK + $(L+1)$ Sign	VerifySoK + $(L+2)$ Sign	VerifySoK + 1 Sign	1 Sign
Withdrawal Transfer size	$(2L+2)$ $\lvert\mathsf{Com}\rvert$ + $(L+1)$ $\lvert\mathsf{OpenCom}\rvert$ + $\lvert\mathsf{SoK}\rvert$ + $(L+1)$ $\lvert\mathsf{Sign}\rvert$	$(L+2)$ $\lvert\mathsf{Com}\rvert$ + $\lvert\mathsf{SoK}\rvert$ + $(L+2)$ $\lvert\mathsf{Sign}\rvert$	1 $\lvert\mathsf{Com}\rvert$ + $\lvert\mathsf{SoK}\rvert$ + 1 $\lvert\mathsf{Sign}\rvert$	1 $\lvert\mathsf{Acc}\rvert$ + 1 $\lvert\mathsf{Sign}\rvert$
Binary Tree	$(2K-2)\,\lvert p\rvert$	$(4K-2)\,\lvert p\rvert$	K-bit string	$(3K-2)\,\lvert p\rvert$
$\mathcal{W}$allet	1 $\lvert p\rvert$ + $(L+1)$ $\lvert\mathsf{Acc}\rvert$ + $(L+1)$ $\lvert\mathsf{Sign}\rvert$	2 $\lvert p\rvert$ + $(L+2)$ $\lvert\mathsf{Acc}\rvert$ + $(L+2)$ $\lvert\mathsf{Sign}\rvert$	1 $\lvert p\rvert$ + 1 $\lvert\mathsf{Sign}\rvert$	2 $\lvert p\rvert$ + 1 $\lvert\mathsf{Acc}\rvert$ + 1 $\lvert\mathsf{Sign}\rvert$
Spend Computations $\mathcal{U}$ser	2 E + 1 $\mathsf{Wit}_{K/k-1}$ + SoK$\{$2 E + 1 $\mathsf{Acc}^{\star(1)}$ + 1 Sign$\}$	1 E + 1 $\mathsf{Wit}_{2K/k-2}$ + 1 $\mathsf{Wit}_{4(K-k)}$ + SoK$\{$3 E* + 1 $\mathsf{Acc}^{(2)}$ + 1 $\mathsf{Acc}^{(4k-2)}$ + + 2 Sign$\}$	1 E + 1 ME + SoK{2 E + 2 Sign}	2 E + 1 Wit_{K-k} + SoK$\{$2 E + 1 $\mathsf{Acc}^{(k)}$ + 1 Sign$\}$
Spend Computations $\mathcal{M}$erchant	VerifySoK	$(4k-4)$ E + 1 Acc^*_2 + 1 Acc^*_{4k-2} + VerifySoK	VerifySoK	$(2k-2)$ E + 1 Acc^*_k + VerifySoK
Spend Transfer size	2 $\mathbb{G}_1$ + $\lvert\mathsf{SoK}\rvert$	2 $\mathbb{G}_q$ + 1 $\mathbb{G}_1$ + $\lvert\mathsf{SoK}\rvert$	2 $\mathbb{G}_1$ + $\lvert\mathsf{SoK}\rvert$	2 $\mathbb{G}_1$ + $\lvert\mathsf{SoK}\rvert$
Deposit Computations $\mathcal{B}$ank	VerifySoK + $(2k-2)$ E	$(4k-4)$ E + 1 Acc^*_2 + 1 Acc^*_{4k-2} + VerifySoK	VerifySoK + K Pair	$(2k-2)$ E + 1 Acc^*_k + VerifySoK
coin $\in \mathcal{D}$	k $\mathbb{G}_1$ + $\lvert\mathsf{Spend}\rvert$	k $\mathbb{G}_q$ + $\lvert\mathsf{Spend}\rvert$	K $\mathbb{G}_T$ + $\lvert\mathsf{Spend}\rvert$	k $\lvert p\rvert$ + $\lvert\mathsf{Spend}\rvert$
Identify $\ell = \ell'$	1 ME	1 ME	1 ME + $(\#+2)$ P	1 ME
Identify $\ell > \ell'$	$(\ell - \ell')$ E	$(\ell - \ell')$ E	1 ME + $(\#+2)$ P	$(\ell - \ell')$ E

Table 1: General efficiency of [ASM08], [CG10], [CPST15] and our new construction

Whereas in our new system (and also in [ASM08] and [CPST15]) p can be e.g. a 256 bit prime, due to the generation method of the binary tree in [CG10], we have to take q as 256 and p as $3,072$ bit prime for an equivalent level of security in [CG10].[6] As a consequence, the bit length of the group elements in

[6] See `http://www.nsa.gov/business/programs/elliptic_curve.shtml`

[CG10] is much larger.

We used the following abbreviations: $\mathsf{pk}_{\mathsf{Com}}$ refers to a public key of a commitment scheme, $\mathsf{pk}_{\mathsf{Sign}}$ to a public key of a signature scheme, $\mathsf{pk}_{\mathsf{Tree}_L}$ to a public key of a public binary tree with L levels, and $\mathsf{pk}_{\mathsf{Acc}_i}^{(j)}$ to a public key of a bounded accumulator scheme, where i is the bound of values that can be accumulated and j is the limit of values, for which a user wants to prove that they are inside an accumulator. Using the Nguyen accumulator scheme (see Section 2.3), each $\mathsf{pk}_{\mathsf{Acc}_i}^{(j)}$ contains i elements in $\mathbb{G}_1$ and j elements in $\mathbb{G}_2$. E denotes an exponentiation, ME an multi-based exponentiation, P a pairing computation, Com a commitment computation, Sign a signature computation, Acc_i an accumulator computation with i values inside the accumulator, Wit_i a witness computation with i values inside the witness, and Acc_i^* an accumulator computation with i values inside the accumulator, where Acc_i^* has to be computed by the merchant and the bank (possibly in another group as an accumulator Acc_i) to verify the signature of knowledge. $\mathsf{VerifySign}$ refers to the computation cost to verify a signature and $\mathsf{VerifySoK}$ to the computation cost to verify a signature of knowledge (SoK). $\mathsf{SoK}\{\mathsf{Com}\}$ denotes the cost of a SoK to prove the correct computation of a commitment, $\mathsf{SoK}\{\mathsf{E}\}$ the cost of a SoK to prove equality of discrete logarithms, $\mathsf{SoK}\{\mathsf{E}^*\}$ the cost of a SoK to prove equality of discrete logarithms in groups of different order, $\mathsf{SoK}\left\{\mathsf{Acc}^{\star(1)}\right\}$ the cost of a SoK to prove that one secret value is inside a secret accumulator, $\mathsf{SoK}\left\{\mathsf{Acc}^{(i)}\right\}$ the cost of a SoK to prove that i known values are inside a secret accumulator, and $\mathsf{SoK}\{\mathsf{Sign}\}$ the cost of a SoK to prove possession of a valid signature. We denote the number of all users by #.

	Random Oracle Model		Standard Model	
	[CPST15]	Our work	[CPST15]	Our work
sp	$\mathbb{G}_1^{4K+3} \times \mathbb{G}_2^{K(L+1)+1}$	$\mathbb{G}_1^{K+3} \times \mathbb{G}_2^{K+2} \times \mathbb{G}_T$	$\mathbb{G}_1^{4K+7} \times \mathbb{G}_2^{K(L+1)+4}$	$\mathbb{G}_1^{K+6} \times \mathbb{G}_2^{K+5} \times \mathbb{G}_T$
Bits	6,832,388	791,301	6,834,955	793,611
$\mathsf{pk}_{\mathcal{B}}$	$\mathbb{G}_1^{4K-2} \times \mathbb{G}_2^{2K+4L+4}$	$\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_T$	$\mathbb{G}_1^{4K-2} \times \mathbb{G}_2^{2K+4L+7}$	$\mathbb{G}_1 \times \mathbb{G}_2^2 \times \mathbb{G}_T$
Bits	2,125,354	1,794	2,126,893	2,307
$\mathsf{sk}_{\mathcal{B}}$	$\mathbb{Z}_p$	$\mathbb{Z}_p^3$	$\mathbb{Z}_p^4$	$\mathbb{Z}_p^3$
Bits	256	768	1,024	768
Withdrawal	$\mathbb{G}_1^7 \times \mathbb{Z}_p^6$	$\mathbb{G}_1^2 \times \mathbb{G}_2^2$	$\mathbb{G}_1^8 \times \mathbb{G}_2 \times \mathbb{Z}_p^4$	$\mathbb{G}_1^2 \times \mathbb{G}_2^2$
Bits	3,335	1,540	3,593	1,540
$\mathcal{U}$ser	8 ME + 2 P	2,049 ME + 4 P	6 ME + 7 P	2,049 ME + 4 P
$\mathcal{B}$ank	5 ME	3 ME	7 ME	3 ME
Binary Tree	K-bit string	$\mathbb{Z}_p^{3K-2}$	K-bit string	$\mathbb{Z}_p^{3K-2}$
Bits	1,024	785,920	1,024	785,920
$\mathcal{W}$allet	$\mathbb{G}_1 \times \mathbb{Z}_p^3$	$\mathbb{G}_1^2 \times \mathbb{G}_2^2 \times \mathbb{Z}_p^2$	$\mathbb{G}_1^2 \times \mathbb{G}_2 \times \mathbb{Z}_p$	$\mathbb{G}_1^2 \times \mathbb{G}_2^2 \times \mathbb{Z}_p^2$
Bits	1,025	2,052	1,283	2,052
Spend	$\mathbb{G}_1^{10} \times \mathbb{G}_2 \times \mathbb{Z}_p^{20}$	$\mathbb{G}_1^5 \times \mathbb{G}_2^2 \times \mathbb{Z}_p^5$	$\mathbb{G}_1^{39} \times \mathbb{G}_2^{41}$	$\mathbb{G}_1^{19} \times \mathbb{G}_2^{19}$
Bits	8,203	3,591	31,056	14,630
$\mathcal{U}$ser	24 ME + 8 P	10 ME + 2 P	81 ME	40 ME
$\mathcal{M}$erchant	16 ME + 9 P	66 ME + 6 P	2 ME + 164 P	65 ME + 75 P
Deposit	$\mathbb{G}_1^{10} \times \mathbb{G}_2 \times \mathbb{Z}_p^{21}$	$\mathbb{G}_1^5 \times \mathbb{G}_2^2 \times \mathbb{Z}_p^6$	$\mathbb{G}_1^{39} \times \mathbb{G}_2^{41} \times \mathbb{Z}_p$	$\mathbb{G}_1^{19} \times \mathbb{G}_2^{19} \times \mathbb{Z}_p$
Bits	8,459	3,847	31,312	14,886
$\mathcal{B}$ank	16 ME + 1,033 P	66 ME + 6 P	2 ME + 1,188 P	65 ME + 75 P
$\mathsf{coin} \in \mathcal{D}$	$\mathbb{G}_1^{10} \times \mathbb{G}_2 \times \mathbb{G}_T^K \times \mathbb{Z}_p^{21}$	$\mathbb{G}_1^5 \times \mathbb{G}_2^2 \times \mathbb{Z}_p^{k+6}$	$\mathbb{G}_1^{39} \times \mathbb{G}_2^{41} \times \mathbb{G}_T^K \times \mathbb{Z}_p$	$\mathbb{G}_1^{19} \times \mathbb{G}_2^{19} \times \mathbb{Z}_p^{k+1}$
Bits	1,057,035	12,039	1,079,888	23,078

Table 2: Efficiency comparison between [CPST15] and our new construction

Table 2 shows the efficiency of specific implementations of [CPST15] and our new construction in

the random oracle model and the standard model.[7]

The bit lengths of the group elements ($|\mathbb{G}_1| = 257,\ |\mathbb{G}_2| = 513,\ |\mathbb{G}_T| = 1,024$) for a Type-3 pairing are taken from [CHKM09, CHM10]. As an example, we take $K = 2^L = 2^{10}$ and $k = 2^\ell = 2^5$.

The naive computation of an accumulator Acc_i requires i (single-based) exponentiations and i multiplications. However, this is seen as one multi-based exponentiation (like in [ASM08, Au09]) because a multi-base exponentiation takes a similar time as a single-base exponentiation (see [QWDFZ11, BGR98]).

If the bank only stores a 256-bit hash value of each serial number in [CPST15], the storage space of each coin $\mathsf{coin} \in \mathcal{D}$ will be reduced to 270,603 bits (see [CPST15, Remark 5]). Furthermore, we remark that the system in [CPST15] can also make use of the rerandomizable AGHO signature scheme. In addition, it seems that during the withdrawal protocol no proof of knowledge is required as long as the user authenticates himself to the bank and the element U_1 (and its correct computation) is known to the bank. (Each user could send $\mathsf{pk}_\mathcal{U}, U_1$ to the bank and prove the correct computation via a PoK to open an account.)

7 Extensions

7.1 Arbitrary Wallet Sizes

We can modify the withdrawal protocol such that the user can withdraw an arbitrary monetary value $K^\star \leq K = 2^L$, without increasing the complexity of the system. The main idea is the following: Each user is able to accumulate up to K values into an accumulator V. This leads to a wallet containing up to K serial keys. To withdraw a wallet with monetary value $K^\star \leq K$, the bank has to make sure, that only $K^\star$ serial keys are accumulated in V. Thus, $\mathcal{U}$ accumulates $K^\star \leq K$ serial keys $\kappa_0, \ldots, \kappa_{K^\star-1}$ as before but then has to set $\kappa_{K^\star} = \cdots = \kappa_{K-1} := 0$ as invalid keys. Thus, we have $V = u_0^{s\prod_{j=0}^{K^\star-1}(\alpha+\kappa_j)\prod_{j=K^\star}^{K-1}(\alpha+0)} = u_0^{s\prod_{j=0}^{K^\star-1}(\alpha+\kappa_j)\alpha^{K-K^\star}}$. To convince $\mathcal{B}$ that $\mathcal{U}$ accumulates only $K^\star \leq K$ serial keys, $\mathcal{U}$ computes the witness $W_I = u_0^{s\prod_{j=0}^{K^\star-1}(\alpha+\kappa_j)}$ (that is the usual witness for $I = \{K^\star, \ldots, K-1\}$) and sends W_I to the bank, which checks that $\hat{e}(V, v_0) = \hat{e}(W_I, v_{K-K^\star})$, where $0 \leq K - K^\star \leq K$ and hence $v_{K-K^\star}$ is public. Since the accumulator scheme is bounded, there can only be $K^\star$ valid serial keys $\kappa_0, \ldots, \kappa_{K^\star-1} \neq 0$ accumulated into V. Note, that since the hash function H maps to $\mathbb{Z}_p^*$, the case $\kappa_j = 0$ for a serial key can't occur.

There are two options to generate the binary tree. Note, that each value $K^\star \leq 2^L$ can be uniquely split into $K^\star = 2^{L_1} + \cdots + 2^{L_n}$ with $L \geq L_1 > \cdots > L_n \geq 0$, i.e. $n = \mathcal{O}(\log(K^\star))$ is just the number of 1's in the binary representation of $K^\star$.

1. As before, the user randomly chooses $\kappa_{0,0}$ and generates the whole binary tree. After that, $\mathcal{U}$ deletes the serial keys $\kappa_{K^\star}, \ldots, \kappa_{K-1}$ and all of theirs ancestors. Thus, the user gets n binary trees of levels $L_1, \ldots, L_n$.

2. The user directly generates n binary trees of levels $L_1, \ldots, L_n$.

The list $\mathcal{K}_0$ contains $\sum_{i=1}^n \left(2^{L_i+1} - 2\right) = 2\sum_{i=1}^n 2^{L_i} - 2n \leq 2K - 2$ keys and the list $\mathcal{K}_1$ contains $K^\star \leq K$ serial keys. Hence, if the user stores all keys, the storage space of both options is the same. However, the second method only requires $\sum_{i=1}^n \left(2^{L_i+1} - 1\right) = 2\sum_{i=1}^n 2^{L_i} - n \leq 2K - 1$ exponentiations.

Table 3 shows the costs of the modified $\mathsf{Withdrawal}$ protocol in comparison to the normal protocol.

[7] The [CPST15] scheme is implemented in the random oracle model with AGHO signature [AGHO11] to instantiate signature scheme Σ_0 and BBS+ signature [ASM06] to instantiate signature scheme Σ_1; not the non-anonymous scheme in [CPST15, Appendix A.2].

	costs of modified Withdrawal protocol
Withdrawal Transfer size	$+\,\mathbb{G}_1$
Bits	$+\,257$
$\mathcal{U}$ser	$+\,1$ ME
$\mathcal{B}$ank	$+\,2$ P

Table 3: Costs of the modified withdrawal protocol

7.1.1 Security Analysis

Obviously, this modification doesn't have any effect on the security properties. By sending the witness W_I, the user proves that there are at most $K^\star \leq K$ values inside the accumulator V. Thus, the balance property is still fulfilled. Further, anonymity is still fulfilled since the bank can compute the witness W_I by itself as $W_I = V^{1/\alpha^{K-K^\star}}$, if the bank knows α, which is is allowed in Game Anonymity. Hence, the element W_I can't help the adversary $\mathcal{A}$ to win the game and consequently there is no need for hiding W_I.

7.2 Coins with Arbitrary Monetary Value

We can modify the spend protocol such that the user can spend an arbitrary monetary value k, which is much more efficient than performing the (*normal*) spend protocol several times. As above, each value $k \leq K$ can be uniquely split into $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$ with $L \geq \ell_1 > \cdots > \ell_n \geq 0$, where we have an upper bound $n \leq L$ for $k \leq 2^L$. To spend a coin of arbitrary monetary value k, the user chooses n unused keys $\kappa_{L-\ell_1,j_1}, \ldots, \kappa_{L-\ell_n,j_n} \in \mathcal{K}_0$, each with smallest index j_i for $1 \leq i \leq n$. Let $\kappa_i := \kappa_{L-\ell_i,j_i}$ for all $1 \leq i \leq n$. Then, the user computes n serial numbers $S_i = g^{\kappa_i}$ and n appropriate security tags $T_i = U g_1^{R\kappa_i}$ for $1 \leq i \leq n$. But, $\mathcal{U}$ has only to compute one witness W_I and to generate one proof Φ:

- **Random Oracle Model:** The user randomly chooses $r', r_1, r_2 \in_R \mathbb{Z}_p^*$, computes the elements $A' = A^{r'}, B' = B^{r'}, C'' = C^{1/(r'r_1)}, V' = V^{1/r_2}, W' = W_I^{1/r_2}$ and generates the following signature of knowledge:

$$\begin{aligned}\Pi = SoK\Big\{(u, \kappa_1, \ldots, \kappa_n, r_1, r_2) : S_1 = g^{\kappa_1} \ \wedge \ \cdots \ \wedge \ S_n = g^{\kappa_n} \ \wedge \\ T_1 = g^u g_1^{R\kappa_1} \ \wedge \ \cdots \ \wedge \ T_n = g^u g_1^{R\kappa_n} \ \wedge \\ \hat{e}\,(g,h) = \hat{e}\,(C'', A')^{r_1}\, \hat{e}\,(V', Y_1)^{r_2}\, \hat{e}\,(g, Y_2)^u\Big\}\,(S_1, \ldots, S_n, T_1, \ldots, T_n, A', C'', V', R)\,.\end{aligned}$$

- **Standard Model:** The user generates $\mathsf{pk}_{ots}, A', B', C', \mu$ like in Section 5. Next, the user generates Groth-Sahai commitments to $u, \kappa_1, \ldots, \kappa_n \in \mathbb{Z}_p$ (2 elements of $\mathbb{G}_2$ each) and to $U, C', V, W_I, \mu \in \mathbb{G}_1$ (2 elements of $\mathbb{G}_1$ each). Finally, $\mathcal{U}$ generates the following NIZK proof π that the committed values satisfy:

$$\begin{aligned}&\underline{g^{\kappa_1}} = \underline{S_1} \ \wedge \ \cdots \ \wedge \ \underline{g^{\kappa_n}} = \underline{S_n} \ \wedge \ \underline{g^u} \left(\underline{g_1^R}\right)^{\underline{\kappa_1}} = \underline{T_1} \ \wedge \ \cdots \ \wedge \ \underline{g^u} \left(\underline{g_1^R}\right)^{\underline{\kappa_n}} = \underline{T_n} \ \wedge \ \underline{g^u}\,\underline{U}^{-1} = 1 \ \wedge \\ &\underline{\mu}^{\underline{u}}\,\underline{\mu}^{H(\mathsf{pk}_{ots})} = \underline{u_0} \ \wedge \ \hat{e}\,(\underline{V}, v_0)\, \hat{e}\left(\underline{W_I}, v_I^{-1}\right) = 1 \ \wedge \ \hat{e}\,(\underline{C'}, A')\, \hat{e}\,(\underline{V}, Y_1)\, \hat{e}\,(\underline{U}, Y_2) = \hat{e}\,(g,h)\,.\end{aligned}$$

  The proof of the first $2n$ equations consists of 1 element of $\mathbb{G}_1$ each. The rest of the proof is the same as in Section 5.

The corresponding k serial keys $\kappa_{L+1,j}$ can be computed from the n serial numbers $S_1, \ldots, S_n$. If the proof is valid, the merchant accepts the payment and stores the coin $\mathsf{coin} = (k, S_1, \ldots, S_n, T_1, \ldots, T_n, R, \Phi, \mathsf{ts})$. In terms of space complexities, this is up to about 4.6, resp. 7.1 times more efficient than performing

the normal spend protocol n times (for $n \to \infty$) in the random oracle model, resp. the standard model. Furthermore, the number of pairings in this new protocol is constant and thus independent of n in the random oracle model (see also Table 4).

Random Oracle Model	normal Spend protocol $k_1 = 2^{\ell_1}, \ldots, k_n = 2^{\ell_n}$	modified Spend protocol $k = 2^{\ell_1} + \cdots + 2^{\ell_n}$
Spend	$\mathbb{G}_1^{5n} \times \mathbb{G}_2^{2n} \times \mathbb{Z}_p^{5n}$	$\mathbb{G}_1^{3+2n} \times \mathbb{G}_2 \times \mathbb{Z}_p^{4+n}$
Bits	$3,591n$	$2,821 + 770n$
$n = 2$	$7,182$	$4,361$
$n = L = 10$	$35,910$	$10,521$
$\mathcal{U}$ser	$10n$ ME + $2n$ P	$(6 + 4n)$ ME + 2 P
$\mathcal{M}$erchant	$\left(2n + 2\sum_{i=1}^{n} 2^{\ell_i}\right)$ ME + $6n$ P	$\left(2 + 2\sum_{i=1}^{n} 2^{\ell_i}\right)$ ME + 6 P
Standard Model	**normal Spend protocol**	**modified Spend protocol**
Spend	$\mathbb{G}_1^{19n} \times \mathbb{G}_2^{19n}$	$\mathbb{G}_1^{15+4n} \times \mathbb{G}_2^{17+2n}$
Bits	$14,630n$	$12,576 + 2,054n$
$n = 2$	$29,260$	$16,684$
$n = L = 10$	$146,300$	$33,113$
$\mathcal{U}$ser	$40n$ ME	$(34 + 6n)$ ME
$\mathcal{M}$erchant	$\left(n + 2\sum_{i=1}^{n} 2^{\ell_i}\right)$ ME + $75n$ P	$\left(3 - 2n + 2\sum_{i=1}^{n} 2^{\ell_i}\right)$ ME + $(61 + 14n)$ P

Table 4: Efficiency of the modified Spend protocol

7.2.1 Security Analysis

The security properties balance and strong exculpability follows directly from the scheme and the proofs in Section 5. So, we only have to analyze anonymity.

The reduction $\mathcal{R}$ behaves like in Section 5, except for the challenge spending. Here, $\mathcal{R}$ randomly chooses $r_1, \ldots, r_n \in_R \mathbb{Z}_p^*$, defines $S_i := g^b g^{r_i}$ and $T_i := U\,(g^c)^R\,(g^a)^{Rr_i}$ for $1 \leq i \leq n$ and perfectly simulates the proof Φ.

7.3 Fair Divisible E-Cash

Independent of the above extensions, we can modify the system to get a fair e-cash system which provides very efficient owner tracing while all protocols remain unchanged in the random oracle model. To achieve owner tracing, the trusted third party $\mathcal{T}$ only has to know the discrete logarithm $\log_g g_1 =: z$ as its secret key ($\mathcal{T}$ can perform the Setup algorithm such that $\mathcal{T}$ knows z). So, the key pair $(\mathsf{pk}, \mathsf{sk}) = ((g, g_1), z)$ is an ElGamal key pair. Further, the pair $\left(S^R, T\right) = \left(g^{R\kappa}, U g_1^{R\kappa}\right)$ forms exactly an ElGamal encryption of the user public key U. Thus, to trace the owner of a coin $\mathsf{coin} = (k, S, T, R, \Phi, \mathsf{ts})$, $\mathcal{T}$ verifies the proof Φ and decrypts the ciphertext $\left(S^R, T\right)$ to obtain the plaintext $U = T/S^{Rz}$.

To ensure owner tracing in the standard model, we could use simulation-sound extractable NIZK proofs ([Gro06]).

To achieve coin tracing, one of the following variations of the methods proposed in [CHL05] or [Au09, Section 5.6] can be used.

All variations have in common, that $t \in \mathbb{Z}_p^*$ is a number which is generated randomly by the user and the bank, sings by the bank and only known to the user. Thus, during the withdrawal protocol, the user randomly chooses $t' \in \mathbb{Z}_p^*$, computes $C_1 = g^{t'} \in \mathbb{G}_1$ and sends C_1 to the bank. The bank randomly chooses $r, t'' \in \mathbb{Z}_p^*$, computes $A = h^r, B = A^x, C = \left(g \cdot V^{-y_1} \cdot U^{-y_2} \cdot \left(C_1 g^{t''}\right)^{-y_3}\right)^{1/r}$ and sends A, B, C, t'' to the user who sets $t := t' + t'' \pmod p$.

Further, $\mathcal{T}$ needs an additional key pair $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ for a semantically secure encryption scheme.

1. The variation of [CHL05] is the following. Instead of verifiably encrypting the value t under his own public key, $\mathcal{U}$ verifiably encrypts t under $\mathsf{pk}_{\mathcal{T}}$. During Spend protocol, $\mathcal{U}$ computes the tracing tag analogously to the serial number in [CHL05] as $T_t = \mathsf{Vrf}(t, J)$, where $\mathsf{Vrf}(t, J)$ denotes a verifiable random function on input a wallet counter J with respect to seed t (see [CHL05, CHL06b] for details).

2. Au [Au09] proposed another approach based on the idea of traceable signatures. As above, $\mathcal{U}$ verifiably encrypts t under $\mathsf{pk}_{\mathcal{T}}$ during $\mathsf{Withdrawal}$ protocol. But in the spending phase, $\mathcal{U}$ generates a random one time tag $Q \in \mathbb{G}_1$ and computes the tracing tag as $T_t = Q^t$. After decrypting t, everyone can verify $T_t \stackrel{?}{=} Q^t$.

3. We'll give a third solution. Instead of using a verifiable encryption scheme we can use the efficient ElGamal encryption with a proof of knowledge. During $\mathsf{Withdrawal}$ protocol, $\mathcal{U}$ generates an ElGamal encryption of the message $h^t \in \mathbb{G}_2$ under $\mathsf{pk}_{\mathcal{T}}$, which is secure under the SXDH assumption, and proves its correctness (using a standard proof of knowledge about discrete logarithms). The tracing tag is computed as in [Au09] as $T_t = Q^t$, but to trace a coin, now everyone can test the equation $\hat{e}(T_t, h) \stackrel{?}{=} \hat{e}(Q, h^t)$ after decrypting h^t.

7.3.1 Security Analysis

We just informally sketch the security and won't give a formal security definition for fair e-cash schemes. Compared to Section 5, $\mathcal{A}$ can ask for user and coin tracing in Game Strong Exculpability and Game Anonymity with the restriction that $\mathcal{A}$ is not allowed to ask for owner tracing regarding the challenge coin and not allowed to ask for coin tracing regarding the challenge indices i_0, i_1.

Thus, compared to Section 5, $\mathcal{R}$ now has to simulate $\mathcal{T}$'s role of the owner tracing protocol in Game Strong Exculpability and Game Anonymity (without knowledge of $\log_g g_1$ in Game Anonymity):

- If coin was generated by $\mathcal{R}$, it simply outputs U.
- Else, $\mathcal{R}$ extracts u in the random oracle model and outputs $U = g^u$ (or extracts U if we use simulation-sound extractable NIZK proofs).

Hence, the owner tracing mechanism doesn't have any effect on the security properties.

To simulate $\mathcal{T}$'s role of the coin tracing protocol, $\mathcal{R}$ generates $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ and hence is able to answer each query honestly. Thus, the coin tracing mechanism doesn't have any effect on the strong exculpability property. So again, we only have to analyze anonymity.

The first variation is secure under the security of the used verifiable encryption scheme and the verifiable random function (see [CHL06b]). The second variation is secure under the security of the verifiable encryption scheme and the DDH assumption (see [Au09, Section 5.6]).

Our third solution is secure, if the ElGamal encryption scheme with message space $\mathbb{G}_2$ is semantically secure and if the tracing tag $T_t \in \mathbb{G}_1$ is indistinguishable from a random element in $\mathbb{G}_1$. Both requirements hold under the SXDH assumption.

7.4 Remarks on Hash Value R

As seen in the security proofs, there is no need for defining $R = H(\mathsf{pk}_{\mathcal{M}} || \mathsf{ts} || k)$ as the output of a hash function $H\colon \{0,1\}^* \to \mathbb{Z}_p^*$. We only require the following two properties:

1. $0 < R < p$,
2. R is unique for each coin, except with negligible probability.

Clearly, each time stamp ts and each monetary value $k \leq K = 2^L$ can be represented as a bit string $\mathsf{ts} \in \{0,1\}^n$ and $k \in \{0,1\}^{L+1}$ for (small) numbers $n, L \in \mathbb{N}$.

Assuming that each merchant $\mathsf{pk}_{\mathcal{M}}$ has an additional, unique, public identity $\mathsf{ID} \in \{0,1\}^{n'}$ for $n' \in \mathbb{N}$ such that $\mathsf{ID}||1^{n+L+1} < p$, where 1^{n+L+1} denotes the bit string $111\cdots111$ of length $(n+L+1)$. So, if p is a λ bit prime, ID can be a bit string of length $n' = \lambda - (n+L+2)$.

Thus, we can define R as the concatenation $R = \mathsf{ID}||\mathsf{ts}||k$. Hence, we have $0 < R < p$ and $R = \mathsf{ID}||\mathsf{ts}||k \neq \mathsf{ID}'||\mathsf{ts}'||k' = R'$ if $(\mathsf{ID}, \mathsf{ts}) \neq (\mathsf{ID}', \mathsf{ts}')$ (and thus $(\mathsf{pk}_{\mathcal{M}}, \mathsf{ts}) \neq (\mathsf{pk}'_{\mathcal{M}}, \mathsf{ts}')$) as required.

As a consequence, there is no need for storing the public key $\mathsf{pk}_{\mathcal{M}}$, the time stamp ts and the monetary value k, since these information are contained in R.

8 Conclusion

We presented a divisible e-cash system in which the bandwidth of each protocol is constant while the system fulfills the standard security requirements in the random oracle model and in the standard model. Furthermore, we proposed a modified withdrawal protocol that allows to withdraw an arbitrary monetary value $K^\star \leq K$ and a modified spend protocol that allows the spending of an arbitrary monetary value $k \leq K^\star$. Finally, we showed an efficient approach to provide owner tracing by a trusted third party without changing any protocol in the random oracle model.

Independent of our new divisible e-cash system, we presented an attack against the unforgeability of the divisible e-cash scheme in [CG10] and suggested two possible preventions. However, these preventions either lead to a less secure or to a less efficient scheme.

Besides, we gave a new formal definition of strong exculpability, because the given definitions in [CHL06b] and [Tro06] don't fulfill the mentioned intention in [CHL05, CHL06b]. Furthermore, we showed that, contrary to the assertion in [CG07], the exculpability requirement defined in [Tro06] is not implied by the security definition in [CHL06b, CG07] and we mentioned the conditions to guarantee this requirement.

References

[ACHdM05] Giuseppe Ateniese, Jan Camenisch, Susan Hohenberger, and Breno de Medeiros. Practical group signatures without random oracles. Cryptology ePrint Archive, Report 2005/385, October 2005.

[AGHO11] Masayuki Abe, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Optimal structure-preserving signatures in asymmetric bilinear groups. In *Advances in Cryptology - CRYPTO '11*, volume 6841 of *Lecture Notes in Computer Science*, pages 649–666. Springer, 2011.

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In *Security and Cryptography for Networks – SCN '06*, volume 4116 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2006.

[ASM07] Man Ho Au, Willy Susilo, and Yi Mu. Practical compact e-cash. In *Information Security and Privacy – ACISP '07*, volume 4586 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 2007.

[ASM08] Man Ho Au, Willy Susilo, and Yi Mu. Practical anonymous divisible e-cash from bounded accumulators. In *Financial Cryptography and Data Security – FC '08*, volume 5143 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2008.

[Au09] Man Ho Au. *Contribution to Privacy-Preserving Cryptographic Techniques*. PhD thesis, School of Computer Science and Software Engineering, University of Wollongong, May 2009.

[AWSM07] Man Ho Au, Qianhong Wu, Willy Susilo, and Yi Mu. Compact e-cash from bounded accumulator. In *Topics in Cryptology – CT-RSA '07*, volume 4377 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2007.

[BB04] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In *Advances in Cryptology – EUROCRYPT '04*, volume 3027 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2004.

[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, 2008.

[BCKL09] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact e-cash and simulatable vrfs revisited. In *Pairing-Based Cryptography – Pairing '09*, volume 5671 of *Lecture Notes in Computer Science*, pages 114–131. Springer, 2009.

[BGR98] Mihir Bellare, Juan A. Garay, and Tal Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 1998.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security – CCS '93*, CCS, pages 62–73. ACM, 1993.

[Bra94] Stefan Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology – CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 302–318. Springer, 1994.

[CFN89] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Advances in Cryptology – CRYPTO '88*, volume 403 of *Lecture Notes in Computer Science*, pages 319–327. Springer, 1989.

[CFT98] Agnes Chan, Yair Frankel, and Yiannis Tsiounis. Easy come - easy go divisible cash. In *Advances in Cryptology – EUROCRYPT '98*, volume 1403 of *Lecture Notes in Computer Science*, pages 561–575. Springer, 1998.

[CG07] Sébastien Canard and Aline Gouget. Divisible e-cash systems can be truly anonymous. In *Advances in Cryptology – EUROCRYPT '07*, volume 4514 of *Lecture Notes in Computer Science*, pages 482–497. Springer, 2007.

[CG10] Sébastien Canard and Aline Gouget. Multiple denominations in e-cash with compact transaction data. In *Financial Cryptography and Data Security – FC '10*, volume 6052 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2010.

[CGT08] Sébastien Canard, Aline Gouget, and Jacques Traoré. Improvement of efficiency in (unconditional) anonymous transferable e-cash. In *Financial Cryptography and Data Security – FC '08*, volume 5143 of *Lecture Notes in Computer Science*, pages 202–214. Springer, 2008.

[Cha83] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology – CRYPTO '82*, pages 199–203. Plenum Press, 1983.

[CHKM09] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. Cryptology ePrint Archive, Report 2009/060, September 2009.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. In *Advances in Cryptology – EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 302–321. Springer, 2005.

[CHL06a] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks – SCN '06*, volume 4116 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 2006.

[CHL06b] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. Cryptology ePrint Archive, Report 2005/060, March 2006.

[CHM10] Sanjit Chatterjee, Darrel Hankerson, and Alfred Menezes. On the efficiency and security of pairing-based protocols in the type 1 and type 4 settings. In *Arithmetic of Finite Fields – WAIFI '10*, volume 6087 of *Lecture Notes in Computer Science*, pages 114–134. Springer, 2010.

[CL02] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks – SCN '02*, volume 2576 of *Lecture Notes in Computer Science*, pages 268–289. Springer, 2002.

[CLM07] Jan Camenisch, Anna Lysyanskaya, and Mira Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security and Privacy – SP '07*, pages 101–115. IEEE Computer Society, 2007.

[CMS96] Jan Camenisch, Ueli Maurer, and Markus Stadler. Digital payment systems with passive anonymity-revoking trustees. In *Computer Security – ESORICS '96*, volume 1146 of *Lecture Notes in Computer Science*, pages 33–43. Springer, 1996.

[CPST15] Sébastien Canard, David Pointcheval, Olivier Sanders, and Jacques Traoré. Divisible e-cash made practical. In *Public-Key Cryptography – PKC '15*, volume 9020 of *Lecture Notes in Computer Science*, pages 77–100. Springer, 2015.

[CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.

[EG14] Alex Escala and Jens Groth. Fine-tuning groth-sahai proofs. In *Public Key Cryptography – PKC '14*, volume 8383 of *Lecture Notes in Computer Science*, pages 630–649. Springer, 2014.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1985.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions of identification and signature problems. In *Advances in Cryptology – CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[FTY96] Yair Frankel, Yiannis Tsiounis, and Moti Yung. Indirect discourse proofs: Achieving efficient fair off-line e-cash. In *Advances in Cryptology – ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 286–300. Springer, 1996.

[FTY98] Yair Frankel, Yiannis Tsiounis, and Moti Yung. Fair off-line e-cash made easy. In *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 257–270. Springer, 1998.

[GPS06] Steven Galbraith, Kenneth Paterson, and Nigel Smart. Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165, May 2006.

[Gro06] Jens Groth. Simulation-sound nizk proofs for a practical language and constant size group signatures. In *Advances in Cryptology – ASIACRYPT '06*, volume 4284 of *Lecture Notes in Computer Science*, pages 444–459. Springer, 2006.

[Gro15] Jens Groth. Personal communication. April 2015.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *Advances in Cryptology – EUROCRYPT '08*, volume 4965 of *Lecture Notes in Computer Science*, pages 415–432. Springer, 2008.

[GS10] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. Cryptology ePrint Archive, Report 2007/155, February 2010.

[GT03] Matthieu Gaud and Jacques Traoré. On the anonymity of fair offline e-cash systems. In *Financial Cryptography and Data Security – FC '03*, volume 2742 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2003.

[IL13] Malika Izabacène and Benoît Libert. Divisible e-cash in the standard model. In *Advances in Cryptology – Pairing '12*, volume 7708 of *Lecture Notes in Computer Science*, pages 314–332. Springer, 2013.

[Mär15] Patrick Märtens. Practical compact e-cash with arbitrary wallet size. Cryptology ePrint Archive, Report 2015/086, February 2015.

[Ngu05] Lan Nguyen. Accumulators from bilinear pairings and applications to id-based ring signatures and group membership revocation. In *Topics in Cryptology – CT-RSA '05*, volume 3376 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2005.

[NS00] Toru Nakanishi and Yuji Sugiyama. Unlinkable divisible electronic cash. In *Information Security – ISW '00*, volume 1975 of *Lecture Notes in Computer Science*, pages 121–134. Springer, 2000.

[Oka95] Tatsuaki Okamoto. An efficient divisible electronic cash system. In *Advances in Cryptology – CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 1995.

[OO91] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In *Advances in Cryptology – CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 1991.

[QWDFZ11] Bo Qin, Qianhong Wu, Josep Domingo-Ferrer, and Lei Zhang. Preserving security and privacy in large-scale vanets. In *ICICS '11*, volume 7043 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2011.

[Sch11] Sven Schäge. Tight proofs for signature schemes without random oracles. In *Advances in Cryptology – EUROCRYPT '11*, volume 6632 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2011.

[SPC95] Markus Stadler, Jean-Marc Pivetau, and Jan Camenisch. Fair blind signatures. In *Advances in Cryptology – EUROCRYPT '95*, volume 921 of *Lecture Notes in Computer Science*, pages 209–219. Springer, 1995.

[STS99a] Tomas Sander and Amnon Ta-Shma. Auditable, anonymous electronic cash. In *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 555–572. Springer, 1999.

[STS99b] Tomas Sander and Amnon Ta-Shma. On anonymous electronic cash and crime. In *Information Security – ISW '99*, volume 1729 of *Lecture Notes in Computer Science*, pages 202–206. Springer, 1999.

[Tro06] Mårten Trolin. A stronger definition for anonymous electronic cash. Cryptology ePrint Archive, Report 2006/241, August 2006.

[TY98] Yiannis Tsiounis and Moti Yung. On the security of elgamal based encryption. In *Public Key Cryptography – PKC '98*, volume 1431 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 1998.

[vSN92] Sebastiaan von Solms and David Naccache. On blind signatures and perfect crimes. *Computers and Security*, 11(6):581–583, 1992.

[Wei05] Victor Wei. More compact e-cash with efficient coin tracing. Cryptology ePrint Archive, Report 2005/411, May 2005.