# Practical Anonymous Divisible E-Cash From Bounded Accumulators*

Man Ho Au, Willy Susilo, and Yi Mu

Centre for Computer and Information Security Research
School of Computer Science and Software Engineering
University of Wollongong, Australia
{mhaa456,wsusilo,ymu}@uow.edu.au

**Abstract.** We present an efficient off-line divisible e-cash scheme which is *truly anonymous* without a trusted third party. This is the second scheme in the literature which achieves full unlinkability and anonymity, after the seminal work proposed by Canard and Gouget. The main trick of our scheme is the use of a bounded accumulator in combination with the classical binary tree approach.
The aims of this paper are twofold. Firstly, we analyze Canard and Gouget's seminal work on the efficient off-line divisible e-cash. We point out some subtleties on the parameters generation of their scheme. Moreover, spending a coin of small value requires computation of several hundreds of multi-based exponentiations, which is very costly. In short, although this seminal work provides a new approach of achieving a truly anonymous divisible e-cash, unfortunately it is rather impractical. Secondly, we present our scheme that uses a novel approach of incorporating a bounded accumulator. In terms of time and space complexities, our scheme is 50 to 100 times more efficient than Canard and Gouget's work in the spend protocol at the cost of an 10 to 500 (the large range is due to whether pre-processing is taken into account and the probabilistic nature of our withdrawal protocol) times less efficient withdrawal protocol. We believe this trade-off between the withdrawal protocol and the spend protocol is reasonable as the former protocol is to be executed much less frequent than the latter. Nonetheless, while their scheme provides an affirmative answer to whether divisible e-cash can be *truly anonymous*, our result puts it a step further and we show that truly anonymous divisible e-cash can be *practical*.

## 1 Introduction

Electronic cash (e-cash) was introduced by Chaum [15] in 1982. In its simplest form, an *e-cash* system consists of three parties (the bank $\mathcal{B}$, the user $\mathcal{U}$ and the merchant $\mathcal{M}$) and four main procedures, namely, account establishment, withdrawal, spending and deposit. The user $\mathcal{U}$ first performs an account establishment protocol with the bank $\mathcal{B}$. The currency circulating around is quantized as

* This paper is the extended version of the paper to appear in FC 2008 under the same title[3].

coins. $\mathcal{U}$ obtains a coin by performing a withdrawal protocol with $\mathcal{B}$ and spends the coin by participating in a spend protocol with $\mathcal{M}$. To deposit a coin, $\mathcal{S}$ performs a deposit protocol with $\mathcal{B}$.

A practical electronic cash system should be *secure*, *offline* and *anonymous*. An e-cash system is *offline* when the spend protocol does *not* require $\mathcal{B}$'s participation. In a *secure* e-cash system, only $\mathcal{B}$ can produce a valid electronic coin and users who double-spent the same coin should be identified. The problem of double-spending occurs in the electronic world due to the digital coins ease of duplication. Additionally, honest spenders cannot be slandered to have double-spent (*exculpability*), and when $\mathcal{M}$ deposits the money from the payee, $\mathcal{B}$ should not be able to trace who the actual spender is (*anonymity)*. In a *truly anonymous* e-cash, $\mathcal{B}$, even with the help of $\mathcal{M}$, cannot obtain any information about the identity of the payee. In particular, spending of the same payee cannot be linked together (sometimes refer to as *unlinkability*).

High *efficiency* is also of key importance for practical *e-cash* systems. For efficiency, we look at: (1) the time and bandwidth needed for the withdrawal, spend and deposit protocols; (2) the size of an electronic coin; and (3) the size of the bank's database. In particular, it is desirable if several coins can be withdrawn or spent more efficiently than repeating several times a single withdrawal or spending protocol.

### 1.1 Related Results

In a compact e-cash system [9, 4], users can withdraw efficiently a wallet $\mathcal{W}$ containing $2^L$ coins. However, these coins must be spent one by one. Users in a divisible e-cash system can efficiently withdraw a wallet $\mathcal{W}$ containing $2^L$ coins (*à* la compact e-cash). However, these $2^L$ coins can be spent together efficiently. In particular, spending $2^\ell, \ell \leq L$, coins together can be done more efficiently than repeating the spend protocol for $2^\ell$ times.

A lot of divisible e-cash exists in the literature [24, 25, 16, 17, 23, 14, 21, 12]. Nonetheless, with the exception of [12], none of the above divisible e-cash system is *truly anonymous*. For instance, everyone can tell whether the spending in [23, 14] is from the same wallet (i.e., linkable). In [21], there exists a trusted party who can revoke the identity of every spender (also known as fair e-cash [13]). Moreover, which part of the wallet that is being used is known. That is, if the payee of transaction one and the payee of transaction two are using the same part of a wallet, everyone can conclude that these two transactions are indeed performed with different wallets. We shall investigate the practicality of the only *truly anonymous* divisible e-cash scheme [12] in the next subsection. On the other hand, in contrast to the divisible e-cash schemes, existing compact e-cash schemes [9, 4, 2] are all *truly anonymous*.

### 1.2 On the Practicality of the Truly Anonymous Divisible E-Cash in [12]

We analyze the Canard and Gouget's scheme from [12]. To allow efficeint withdrawal of $2^L$ coins, the construction in [12] requires a series of $L+2$ cyclic groups $(\mathbb{G} = \langle g \rangle, \mathbb{G}_1 = \langle g_1 \rangle, \ldots, \mathbb{G}_{L+1} = \langle g_{L+1} \rangle)$ such that $\mathbb{G}_i \subset \mathbb{Z}^*_{|\mathbb{G}_{i+1}|}$ for $i = 1$ to $L+1$ and $\mathbb{G} \subset \mathbb{Z}^*_{|\mathbb{G}_1|}$ [1] and the decisional discrete logarithm assumption (DDH) holds in all $\mathbb{G}_i$. However, whether such series of groups exists, for moderate $L$ (say, $L = 10$), is unknown. The authors suggest using the same setting of groups in [21] which proposes to set $|\mathbb{G}_i|$, for $i = 1$ to $L+1$, to be of prime order and assume $|\mathbb{G}_{i+1}| = 2|\mathbb{G}_i| + 1$ for $i = 1$ to $L+1$. This implies finding a series of primes $p_1, \ldots, p_{L+1}$ such that $p_{i+1} = 2p_i + 1$. Again, whether such series of primes exist, for moderate $L$, is unknown and it is also unknown how these series of primes can be efficiently generated. The authors in [21] propose using a brute-force approach. That is, randomly generate an odd number $n$ (equals to order of group $\mathbb{G}$) and test if $p_1 := 2n + 1$ is a prime. If yes, compute and test if $p_2 := 2p_1 + 1$ is prime. Continue until $p_{L+1} := 2p_L + 1$ is also a prime. A well-known result, the prime number theory, states that the number of primes not exceeding $m$ is approximately $\frac{m}{\ln(m)}$. Thus, probability that a $k$-bit odd number is a prime is about $\frac{2}{k \ln 2}$. For a randomly generated $k$-bit odd number $n$, probability that $(p_1, \ldots, p_{L+1})$ are primes such that $p_{i+1} := 2p_i + 1$ and $p_1 := 2n + 1$ is approximately $\frac{k! 2^L}{(k+L+1)!(\ln(2)^L)}$. Taking $k = 170$ and $L = 10$, probability of obtaining such series of prime numbers on a given $k$-bit odd number $n$ is about $2^{-66}$. In fact, in [21], $n$ is taken to be an RSA-modulus (which is normally of 1024-bit), and the corresponding probability is $2^{-94}$. Therefore, it is questionable whether the systems in [21] or [12] are in fact implementable.

The spend protocol in [12] is also quite inefficient. As mentioned in the same paper, the authors regard spending a single coin as quite an expensive operation. It is due to the need of $L$ "1-out-of-2 zero-knowledge proof-of-knowledge of of double discrete logarithm". For a cheating probability of $2^{-t}$, a single zero-knowledge proof-of-knowledge of double discrete logarithms requires $t$ exponentiations. For a cheating probability of $2^{-40}$ and a moderate $L$ (say 10), spending a single coin requires $2 * 40 * 10 = 800$ exponentiations. Details analysis of the cost of each protocol can be found in Section 5. Nonetheless, while [12] provides an affirmative answer to whether divisible e-cash can be truly anonymous, it is fair to say constructing a *practical* divisible e-cash which is truly anonymous is not as easy.

### 1.3 Our Approach

The construction of our divisible e-cash is derived from the classical binary tree approach [23, 21, 14, 12], in combination with the use of a bounded accumulator

[1] In [12], it was written as $\mathbb{G}_1 \subset \mathbb{Z}^*_{|\mathbb{G}|}$. However, according to their construction(as it involves computation of $g_1^{g^s}$ for some $s$ in $\mathbb{Z}^*_{|\mathbb{G}|}$), $\mathbb{G} \subset \mathbb{Z}^*_{|\mathbb{G}_1|}$ should be the case.

[4]. We make use of the bounded accumulator to make a trade-off between computational cost during the withdrawal protocol and the spend protocol. The cost (computational and bandwidth) of our withdrawal protocol and spend protocol is $O(L)$ and $O(1)$, respectively, while the corresponding figures for [12] is $O(1)$ and $O(Lt)$. Since the spending protocol is executed much more frequently than the withdrawal protocol, our system is much more desirable in practice.

The trade-off is achieved with the use of accumulators [7, 5]. During the withdrawal protocol, the user computes the accumulation of the binary tree into $L+1$ accumulator values $(V_1, \ldots, V_{L+1})$ and obtains $L+1$ signatures. In the spending protocol, if a node of level $\ell$ is to be used, the user only needs to compute a zero-knowledge proof-of-knowledge such that the node he is about to use is inside the accumulator $V_\ell$. In this way, our spend protocol achieves a complexity of $O(1)$.

An obvious way to ensure the user honestly accumulates node values that form a binary tree, while maintaining anonymity, is to require the user to produce zero-knowledge proof-of-knowledge such that these set of accumulator values $(V_1, \ldots, V_{L+1})$ is correctly formed. This approach, however, is inefficient. Another approach is to apply the cut-and-choose method in a straight-forward manner. Specifically, the user prepares $k$ sets of value, submits them all to the bank who requires the user to reveal $k-1$ of them in random. The bank checks if these $k-1$ sets of value are honestly generated and signs the remaining one if the check is successful. To ensure that a user cannot cheat, $k$ has to be large. Thus, this approach is inefficient as well.

Luckily, bounded accumulator gives us the possibility of a third solution, which is a modification of the cut-and-choose method. Our approach is *statistical*, that is, a cheating user might spend more than what he withdraws for a particular withdrawal protocol but in a long run, the bank is guaranteed that users cannot spend more than they withdraw on average. The idea is derived from the following fact: since the accumulator we use is bounded, the user can only accumulate a predefined number of values regardless of whether they are cheating or not. Naturally, there is an upper bound for which a cheating user might gain. In our scheme, the cheating user can get at most a monetary value of $L2^L$, compared with a value of $2^L$ for an honest user. If the bank inspects the withdrawal protocol every two withdrawal requests and imposes a fine of monetary value $2L2^L$ if a user is found cheating, the bank is guaranteed it will not lost money on average. In Section 3, we will formally define the security model for divisible e-cash schemes that employ this kind of *statistical* approach. In particular, the gain of a cheater cannot be large; since if the gain is large, a cheater might not be able to pay the fine if he is caught. Secondly, a large gain gives extra incentive for people to cheat.

*Our Contributions.* We propose a practical offline divisible e-cash without a trusted third party which is truly anonymous (unlinkable). We formalize the security model of divisible e-cash scheme that employs a *statistical* approach and prove that our construction is secure under this model. We compare the efficiency of our construction to that of [12] and shows that our system can be

more than 50 to 100 times more efficient, in terms of time and space, in the spending protocol.

*Paper Outline.* The rest of the paper is organized as follows. In Section 2 we present preliminary information on the various cryptographic tools and assumptions used in our construction. Security model of divisible e-cash is presented in Section 3. We present our construction in Section 4 and its efficiency analysis in Section 5. Finally we conclude in Section 6.

## 2 Preliminaries

### 2.1 Pairing

A pairing is a bilinear mapping from two group elements to a group element. Let $\hat{e}$ be a bilinear map such that $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ and the following holds.

- $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic multiplicative groups of prime order $p$.
- Each element of $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_3$ has unique binary representation.
- $g$, $h$ are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.
- (*Bilinear*) $\forall x \in \mathbb{G}_1$, $y \in \mathbb{G}_2$ and $a$, $b \in \mathbb{Z}_p^*$, $\hat{e}(x^a, y^b) = \hat{e}(x, y)^{ab}$.
- (*Non-degenerate*)$\hat{e}(g, h) \neq 1$.

$\mathbb{G}_1$ and $\mathbb{G}_2$ can be the same or different groups. We say that two groups $(\mathbb{G}_1, \mathbb{G}_2)$ are a bilinear group pair if the group action in $\mathbb{G}_1$, $\mathbb{G}_2$ and the bilinear mapping $e$ are all efficiently computable.

### 2.2 Mathematical Assumptions

Security of our construction depends on the following existing mathematical assumptions, namely, Decisional Diffie-Hellman, Symmetric External Diffie-Hellman [1], $q$-Strong Diffie-Hellman [8] and AWSM [4]. Their definition is given below.

**Definition 1 (Decisional Diffie-Hellman).** *The Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is defined as follows: On input a quadruple $(g, g^a, g^b, g^c) \in \mathbb{G}^4$, output 1 if $c = ab$ 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 [1]).** *The Symmetric External Diffie-Hellman (SXDH) Assumption states that the DDH problem is hard in both $\mathbb{G}_1$ and $\mathbb{G}_2$ of a bilinear group pair $(\mathbb{G}_1, \mathbb{G}_2)$. It implies that there is no efficiently computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ or vice versa.*

**Definition 3 ($q$-Strong Diffie-Hellman [8]).** *The $q$-Strong Diffie-Hellman ($q$-SDH) problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follows: On input a $(q+2)$-tuple $(g_0, h_0, h_0^x, h_0^{x^2}, \cdots, h_0^{x^q}) \in \mathbb{G}_1 \times \mathbb{G}_2^{q+1}$, output a pair $(A, c)$ such that $A^{(x+c)} = g_0$ where $c \in \mathbb{Z}_p^*$. 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 $(\mathbb{G}_1, \mathbb{G}_2)$.*

**Definition 4 (AWSM [4][2]).** *The AWSM problem in a bilinear group pair* $(\mathbb{G}_1, \mathbb{G}_2)$ *is defined as follows: Let* $A \in \mathbb{G}_3$, $Y \in \mathbb{G}_2$, $g, g_M, g_m, g_0 \in \mathbb{G}_1$, $h \in \mathbb{G}_2$ *and an oracle* $\mathcal{O}$ *that on input* $M_i, t_i$, $\mathcal{O}$ *outputs* $(a_{1,i}, a_{2,i}, a_{3,i}, s_i)$ *that satisfy* $\hat{e}(a_{1,i}, h) = A\hat{e}(M_i g_M^{t_i}, a_{3,i})$ *and* $\hat{e}(a_{2,i}, a_{3,i}Y) = \hat{e}(gg_m^{t_i}g_0^{s_i}, h)$. *The problem is to output* $a_1, a_2, M \in \mathbb{G}_1$, $a_3 \in \mathbb{G}_2$, $s, t \in \mathbb{Z}_p^*$ *that satisfy* $\hat{e}(a_1, h) = A\hat{e}(Mg_M^t, a_3)$ *and* $\hat{e}(a_2, a_3 Y) = \hat{e}(gg_m^t g_0^s, h)$, *such that* $(a_1, a_2, a_3, s)$ *is not equal to the output of* $\mathcal{O}$ *on input* $M, t$. *We say that the AWSM assumption holds in* $\mathbb{G}_1, \mathbb{G}_2$ *if no PPT algorithm has non-negligible advantage in solving the AWSM problem.*

In [4], the AWSM assumption is proven in the generic group model for a bilinear group pair such that the SXDH holds. On the other hand, AWSM assumption does not hold in groups where SXDH assumption does not hold.

### 2.3 Useful Tools

*Zero-Knowledge Proof of Knowledge.* In zero-knowledge proof of knowledge [19], a prover proves to a verifier that a statement is true without revealing anything other than the veracity of the statement. Our construction involves statements related to knowledge of discrete logarithms constructed over a cyclic group $\mathbb{G}$ of prime order $p$. These proofs can also be used non-interactively by using the Fiat-Shamir heuristic [18]. The non-interactive counter part is referred to as signature proof of knowledge, or SPK for short. They are secure in the random oracle model [6]. Following the notation introduced by Camenisch and Stadler [11], $PK\{(x) : y = g^x\}$ denotes a zero-knowledge proof of knowledge protocol between a prover and a verifier such that the prover knows some $x \in \mathbb{Z}_p$ such that $y = g^x \in \mathbb{G}$. Construction of this proof first appeared in the Schnorr Identification[26]. The corresponding non-interactive signature proof of knowledge shall be denoted as $SPK\{(x) : y = g^x\}(M)$.

*ESS+ Signature.* Extended special signature (ESS) was introduced in [4]. It allows signing a block of messages, one of which being an element in a cyclic group $\mathbb{G}$. The authors also proposed two protocols, namely, signature generation protocol and signature possession protocol. The signature generation protocol allows a user to obtain a signature from the signer on message $M$ in $\mathbb{G}$, together with a block of messages $m_1, \ldots, m_L$ in a commitment. The signer learns nothing about $m_1, \ldots, m_L$ while he knows $M$. The signature possession protocol allows a user to conduct a zero-knowledge proof of knowledge on a message signature pair. ESS scheme is uf-cma secure[20] under the AWSM assumption. We modify the signing protocol of ESS so that the signer learns nothing on the block of messages to be signed as well. We refer this modified signature scheme as ESS+ Signature, which is outlined in Appendix A.

ESS+ signature is uf-cma secure in the standard model under the AWSM assumption. We would like to remark that AWSM is a strong assumption, as it requires bilinear group pair where the SXDH assumption [1] holds.

[2] Au *et. al. proposed in [4] a signature scheme called ESS and proved its security in the generic group model [27]. Here, we model their scheme as an oracle-based assumption called AWSM. That is, ESS is secure if and only if the AWSM assumption holds.*

*Bounded Accumulator.* The notion, bounded accumulator was introduced in [4] as an accumulator with a limit $q$ as the maximum number of elements that can be accumulated. We briefly review their construction here.

Let $\mathbb{G}_1, \mathbb{G}_2$ be a bilinear group pair. Let $u_0$ be a random element in $\mathbb{G}_1$ and $v_0$ be a random element in $\mathbb{G}_2$. Let $q$ be the bound of the accumulator. The generation algorithm randomly selects $\alpha \in \mathbb{Z}_p^*$ and computes $u_i = {u_0}^{\alpha^i}$ for $i = 1 \ldots, q$. Compute $v_1 = v_0^{\alpha}$. The public parameters is $(u_0, \ldots, u_q, v_0, v_1)$.

To accumulate a set of $q$ values $(e_1, \ldots, e_k)$, the evaluation algorithm computes the accumulator value $V = u_0^{\prod_{k=1}^{k=q}(e_k+\alpha)}$. This operation does not require knowledge of $\alpha$ since the $u_i$'s are published. A witness $w_i$ such that value $e_i$ is accumulated in the accumulator $V$ is computed by $w_i = u_0^{\prod_{k=1,k\neq i}^{k=q}(e_k+\alpha)}$. The witness-value pair shall satisfy $\hat{e}(w_i, v_1 v_0^{e_i}) = \hat{e}(u_0, v_0)$. Construction of Zero-knowledge proof of knowledge on a value-witness pair can be found in [22].

## 3 Syntax

A (statistical) divisible e-cash is a tuple (BankSetup, UserSetup, WithdrawalProtocol, SpendProtocol, DepositProtocol, RevokeDoubleSpender, VerifyGuilt) of seven polynomial time algorithms/protocols between three entities the bank $\mathcal{B}$, the merchant $\mathcal{M}$ and the user $\mathcal{U}$.

- BankSetup. On input an unary string $1^\lambda$, where $\lambda$ is the security parameter, the algorithm outputs $\mathcal{B}$'s key pair $bpk, bsk$, which includes wallet size $L$, punishment $P$ if a user is found cheating in Inspection Routine(to be discussed) and frequency of which Inspection Routine is carried out $K$.
- UserSetup. On input $bpk$, the algorithm outputs a key pair $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$ (resp. $(pk_{\mathcal{M}}, sk_{\mathcal{M}})$) for $\mathcal{U}$ (resp. $\mathcal{M}$).
- WithdrawalProtocol. $\mathcal{U}$ with input $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$ wishes to withdraws a wallet $\mathcal{W}$ of $2^L$ coins from $\mathcal{B}$ (with input $(bpk, bsk)$). This protocol consists of two routines, namely, Withdrawal Routine and Inspection Routine, respectively. These two routines share the same steps in the beginning such that the user is not aware which routine the bank selects. At a particular point in the protocol, the bank chooses one of these two routines.
  - Withdrawal Routine. With probability $\frac{K-1}{K}$, Withdrawal Routine is executed. The user obtains a wallet $\mathcal{W}$ after executing the protocol, while the bank (possibly) retains certain information $\tau_w$, called the trace information.
  - Inspection Routine. With probability $\frac{1}{K}$, Inspection Routine is executed. Inspection Routine outputs `pass`/`cheat`. If the output is `cheat`, a fine of $P$ shall be deducted from the user account. If the output is `pass`, the user is asked to restart WithdrawalProtocol from the beginning.
- SpendProtocol. This is the protocol when $\mathcal{U}$(with input $\mathcal{W}$, $pk_{\mathcal{M}}$) spends a divisible coin of value $2^\ell$ ($\ell \leq L$ and is decided by the user) to $\mathcal{M}$. After the protocol, $\mathcal{M}$ obtains a coin serial number $S_\ell$, a proof of validity $\pi_S$,

and possibly some auxiliary information $aux$, and outputs 0/1, depending whether the payment is accepted. $\mathcal{U}$'s output is an updated wallet $\mathcal{W}'$.
- DepositProtocol. $\mathcal{M}$ submits ($S_\ell$, $\pi_S$, $aux$) to $\mathcal{B}$ for deposit in this protocol. $\mathcal{B}$ outputs 0/1, indicating whether the deposit is accepted. $\mathcal{B}$ computes, from $S_\ell$, $2^\ell$ serial numbers $\tilde{S}_1, \ldots, \tilde{S}_{2^\ell}$. If any of the serial numbers $\tilde{S}_i$ already belongs to $\mathcal{L}$ (the database of spent coins), $\mathcal{B}$ invokes the RevokeDoubleSpender algorithm to find out the double-spender. Otherwise, it adds $\tilde{S}_i, S_\ell, \pi_S$, $aux$ to $\mathcal{L}$.
- RevokeDoubleSpender. Formally, on input two spending protocol transcripts involving the same coin, the algorithm outputs the public key $pk$ of the double-spender.
- VerifyGuilt. This algorithm allows the public to verify that the user with public key $pk$ is guilty of double-spending. In particular, when the bank uses RevokeDoubleSpender and outputs $\pi_D$ and $pk$ of the double-spender, everyone can check if the bank is honest.

*Requirements:*

- (Correctness for User.) It is required whenever an honest user obtains $\mathcal{W}$ from the bank *who might be dishonest*, an honest merchant shall output 1 when the user engage with the merchant in SpendProtocol.
- (Correctness for Merchant.) It is required whenever an honest merchant obtains ($S_\ell$, $\pi_S$, $aux$) from some execution of SpendProtocol with some user *who might be dishonest*, there is a guarantee that this transaction will be accepted by the honest bank.[3]
- (Practicality.) It is required that $P$ should be small enough so that the fine is payable. For example, if $P = (2^L)^2$, it is very likely that even when a user is found cheating in Inspection Routine, he is unable to pay the fine. In practice, we suggest $P \leq KL2^L$.

### 3.1 Security Notions

We describe informally the security requirements of a statistical divisible e-cash system. A *secure* statistical divisible e-cash scheme should possess, *statistical balance*, *IdentificationOfDoubleSpender*, *anonymity* and *exculpability*, introduced as follows. The reader may refer to Appendix C for the formal version of these definitions.

- *Statistical Balance.* This is the most important requirement from the bank's point of view. Roughly speaking, *balance* means that no collusion of users and merchants together can deposit more than they withdraw without being identified. *Statistical Balance* means that, in a long run, the *balance* property

[3] It can be seen that it is the bank's responsibility to identify the double-spender. The rationale behind is that a user can always spend the same coin to different merchants in an offline e-cash system and the merchant have no way to detect such a double-spending.

is guaranteed. *Statistical Balance* is a relaxation of *balance* since it does not rule out the possibility that a user might cheat without being detected and gain a certain advantage within a small number of times. However, in a long run, no successful strategy would allow collusion of users and merchants to deposit more than they withdraw without being identified.
In particular, what we wish to model is the following situation. The bank does not check every withdrawal request. However, if the user cheats during the withdrawal, at most he can gain a monetary value $P$. If the bank only checks once every $K$ transactions and imposes a fine of $KP$ for each caught cheating, the *Statistical Balance* property will be achieved. It turns out that this relaxation greatly increase the efficiency of our system.

- *Anonymity.* It is required that no collusion of users, merchants and the bank can ever learn the spending habit of an honest user. In particular, spending of the same user *cannot* be linked.
- *Exculpability.* It is required that an honest user cannot be proven to have double-spent, even all other users, merchants and the bank collude.

A statistical divisible e-cash is said to be *secure* if it has Statistical Balance, Anonymity and Exculpability.

## 4 Construction

In this section, we describe our cryptographic construction in detail and assess its security, after giving a high level description.

### 4.1 High Level Description

Following the terminology of [9, 12], spending a single electronic coin consists of generating a serial number $S$, which is used to detect double-spending, a security tag $T$, which is used to reveal identity of the double-spender should the underlying coin is being spent twice. The spender has to prove to the merchant that the pair $(S, T)$ is well-formed. Nonetheless, we provide an overview of our system as follows.

**The Setup Procedure.** The bank $\mathcal{B}$ generates $\mathbf{ESS+}.pk, \mathbf{ESS+}.sk$ pair of the ESS+ Signature. The bank also generates the public parameters of the bounded accumulator as $\mathbf{Acc}_1, \ldots, \mathbf{Acc}_{L+1}$. Let $LF$, $RF$ be two secure cryptographic hash functions. Let $H$ be another secure cryptographic hash function. Let $\mathbb{G} = \langle g \rangle$ be a cyclic group of prime order $p$ such that DDH assumption holds. Let $g_U$, $h$ be additional generators of $\mathbb{G}$.

**The Account Establishment Procedure.** User Alice establishes an account with the bank $\mathcal{B}$ by selecting $x \in \mathbb{Z}_p^*$ and computes $PK_{Alice} := g_U^x$. She sends $PK_{Alice}$ to $\mathcal{B}$, along with a zero-knowledge proof-of-knowledge of the corresponding secret key $x$.

**The Withdrawal Procedure.** Suppose user Alice, who has already established an account with the bank, wishes to withdraw a wallet containing $2^L$ coins. She first randomly chooses a wallet secret $w$ and computes a binary tree of $L+1$ level as follows. The root note $N_{0,0}$ is assigned the node key value $k_{0,0} := w$. For all nodes $N_{i,j}$, the left children, $N_{i+1,2j}$, is assigned a node key value $k_{i+1,2j} := LF(g^{k_{i,j}})$. Similarly, the right children, $N_{i+1,2j+1}$, is assigned a node key value $k_{i+1,2j+1} := RF(g^{k_{i,j}})$. Let $T_w$ be the resulting binary tree computed by Alice. Fig.4.1 illustrates a construction of a binary tree with $L = 3$.

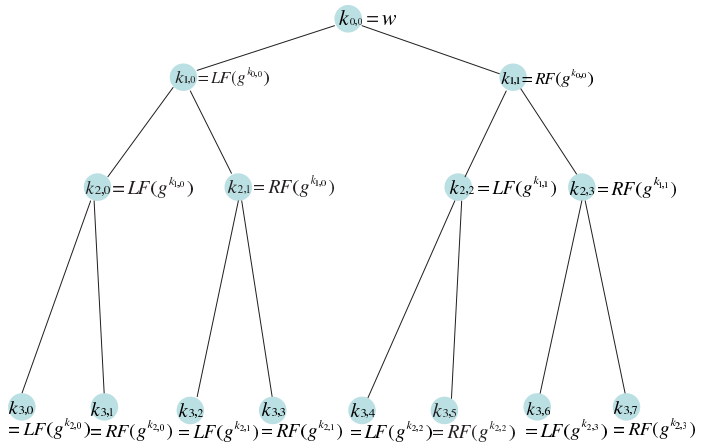


**Fig. 1.** Construction of A Binary Tree (L=3)

For $i = 0$ to $L$, compute $V_i := \mathbf{Acc}_i.\mathsf{Accumulate}(k_{i,0}, \ldots, k_{i,2^i-1})$. Alice then tries to obtain $L+1$ ESS+ Signature on block of messages $(V_i, x)$ using the signature generation protocol of ESS+ Signature.

$\mathcal{B}$ flips a fair coin $b$ and if $b == 1$, $\mathcal{B}$ generates signatures $\sigma_i = \mathbf{ESS+}.\mathsf{Sign}(V_i, x)$ using the signature generation protocol of ESS+ Signature (so that $\mathcal{B}$ learns nothing about $V_i$, $x$ as discussed.) $\mathcal{B}$ sends $\sigma := \{\sigma_0, \ldots, \sigma_L\}$ back to Alice. Alice stores $(\sigma, T_w)$ as her wallet $\mathcal{W}$.

Otherwise if $b == 0$, $\mathcal{B}$ asks Alice to reveal her binary tree. $\mathcal{B}$ tests if the $V_i$'s are honestly generated (that is, checks whether $V_i$ is the accumulation of $k_{i,0}, \ldots, k_{i,2^i-1}$). If yes, $\mathcal{B}$ asks Alice to restart the withdrawal procedure. Otherwise, a fine of $2L2^L$ is deducted from Alice's account.

**The Spending Procedure.** Suppose user Alice with wallet $\mathcal{W}$ wishes to spend to merchant Bob $2^\ell$ dollar where $\ell \leq L$. Alice and Bob agree on certain transaction information $I$ which contains identity of Bob and the monetary value $2^\ell$. Bob also sends Alice a random challenge $R$.

She first chooses a node from the binary tree $T_w$ at level $L-\ell$ which has not been marked as used. Let $N_{i,j}$ be the node chosen (that is, $i = L-\ell$). Compute serial number $S = g^{k_{i,j}}$. Compute security tag $T = g_U^x h^{k_{i,j} R}$.

Alice sends to Bob $S, T$ together with a proof $\pi$ which is a non-interactive zero-knowledge proof-of-knowledge of the following statement:

Alice is in possession of quantities $V_i$, $k_{i,j}$, $x$, $\sigma_i$ which satisfy the following relationship:

1. **ESS+**.$\mathsf{Verify}(\sigma_i, V_i, x) = 1$ (using the signature possession algorithm of ESS+ Signature.)
2. $k_{i,j}$ is a value inside the accumulator $V_i$
3. $S = g^{k_{i,j}}$
4. $T = g_U^x h^{k_{i,j} R}$

Bob verifies if $\pi$ is a valid proof. It accepts the payment if the proof is valid. If Bob accepts the payment, Alice marked down $N_{i,j}$ and all its children, as well as ancestors, from $T_w$ as used node.

*Example: Spending of 4 dollars.* We shall using the binary tree in Fig.4.1 as an example. Alice's wallet includes $\mathcal{W} = \{\sigma_0, \sigma_1, \sigma_2, \sigma_3, V_0, V_1, V_2, V_3, T_w\}$ and wishes to pay 4 dollars to merchant Bob.

Suppose the node $N_{1,0}$ is being chosen. The corresponding node key $k_{1,0}$ is $LF(g^{(w)})$. Compute the serial number $S := g^{LF(g^w)}$ and the security tag $T := g_U^x h^{LF(g^w)R}$ where $R$ is the random challenge given by merchant Bob. She sends $(S, T, \pi_S, 2)$[4] to Bob, where $\pi_S$ is an non-interactive zero-knowledge proof-of-knowledge of the following:

$$SPK_{Spend}\Big\{(\sigma_1, V_1, x, LF(g^w)) : \mathbf{ESS+}.\mathsf{Verify}(\sigma_1, V_1, x) = 1 \wedge LF(g^w) \text{ is an element accumulated in } V_1 \wedge S := g^{LF(g^w)} \wedge T := g_U^x h^{LF(g^w)R}\Big\}(R)$$

Bob accepts the payment if $\pi_S$ is a valid proof. Alice marked $N_{0,0}$, $N_{1,0}$, $N_{2,0}$, $N_{2,1}$, $N_{3,0}$, $N_{3,1}$, $N_{3,2}$, $N_{3,3}$ as used nodes.

**The Deposit Procedure.** Bob sends $(S, T, \pi, R, \ell)$ to the bank for deposit. The bank checks if $R$ is fresh (that is, if $R$ has been used before by Bob). If the check is successful, then credit Bob's account.

The bank then tries to detect if the coin $S$ has been double-spent. Let $S$ be the serial number of a coin of monetary value $2^\ell$. Let $N_{i,j}$ be the corresponding node of the binary tree. From $S$, the bank computes the $2^\ell$ serial numbers corresponding to the leaves of subtree of node $N_{i,j}$ by repeatedly applying the functions $LF(\cdot)$, $RF(\cdot)$ and $g^{(\cdot)}$.

[4] The 2 here indicates the monetary value of the coin is $2^2$.

For each serial number $S_i$, the bank checks if it exists in the database. If not, it stores $(S_i, S, T, R, \pi)$ in its database. Suppose there exists another entry in the database $(S'_i, S', T', R', \pi')$, the bank runs the identify procedure discussed in the following subsection.

*Example: Depositing a coin of 4 dollars.* Continuing our example, Bob submits $(S, T, \pi_S, R, 2)$ to the bank for deposit after getting 4 dollars from Alice. The bank verifies is a valid proof and $R$ is fresh and credits Bob. From $S$, the bank computes $S_0, \ldots, S_3$ as follows. Compute intermediate value $\tilde{S}_0 = g^{LF(S)}$ and $\tilde{S}_1 = g^{RF(S)}$. Compute $S_0 = g^{LF(\tilde{S}_0)}$, $S_1 = g^{RF(\tilde{S}_0)}$, $S_2 = g^{LF(\tilde{S}_1)}$, $S_3 = g^{LF(\tilde{S}_1)}$. The bank checks if $S_0, S_1, S_2, S_3$ exists in its database of spent-coins. If not, it stores $(S_0, S_1, S_2, S_3, S, T, \pi_S, R)$ in the database.

**The Identify (Double-Spender) Procedure.** On input two entries ($S_i$, $S$, $T$, $R$, $\pi$) and ($S'_i$, $S'$, $T'$, $R'$, $\pi'$), the bank computes the identity of the double-spender as follows. If $S$ and $S'$ are the same, compute $PK_{\text{cheater}} := (\frac{T^{R'}}{T'^{R}})^{\frac{1}{R'-R}}$.

On the other hand, if $S$ and $S'$ are different, $S$ and $S'$ must be of different monetary value. Without loss of generality, assume the monetary value of coin with serial number $S$ is greater than that of $S'$. The bank can compute the node key $k_{i,j}$ such that $S' = g^{k_{i,j}}$ from $S$ by repeatedly applying the $LF(\cdot)$, $RF(\cdot)$, $g^{(\cdot)}$ in suitable order. From $k_{i,j}$, the bank computes $pk_{\text{cheater}} = \frac{T'}{h^{R' k_{i,j}}}$ and obtains identity of the double-spender.

*Example: Catching Double-Spender.* Continuing our example. Assuming Alice spends another 2 dollars to Bob, this time using the node $N_{2,1}$ in an attempt to over-spends her wallet. Assume the resulting transcript is $(S', T', \pi_{S'}, R', 1)$. When Bob submits this transcript for deposit, the bank will identify it as a double-spent coin since $S'_1 := g^{LF(S')}$ will be equal to $S_3$ in the above example. The bank can then compute, from $S$ in the above example, $k_{2,1} = RF(S)$. From $k_{2,1}$, the bank computes $PK_{\text{cheater}} := \frac{T'}{h^{R' k_{2,1}}}$ such that $PK_{\text{cheater}} = PK_{Alice}$.

This completes the high-level description of our system.

### 4.2 System Construction.

**Bank's Setup.** Let $2^L$ be the size of a wallet in the system. Let $\lambda$ be a security parameter. On input $\lambda$, generate a $\lambda$-bit prime $p$. Generate a bilinear group pair of order $p$. That is, $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3$ is a bilinear map such that $|\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_3| = p$. Let $g, g_A, g_B, g_0, g_1, g_2, g_3, g_4, u_0, g_U, g_S, g_T$ be random elements in $\mathbb{G}_1$, $h, h_1, h_2, h_3, v$ be random elements in $\mathbb{G}_2$. Since $\mathbb{G}_1$, $\mathbb{G}_2$ are of prime orders, all the above random elements are generators. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ be a secure cryptographic hash function. Let $H_0 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$, $H_1 : \{0,1\}^* \rightarrow \mathbb{Z}_p^*$ be two other secure cryptographic hash function.

The bank randomly chooses $X \in \mathbb{G}_1$, $y, \alpha_0, \ldots, \alpha_L \in \mathbb{Z}_p^*$. Compute $Y = h^y$ and $Z = \hat{e}(X, h)$. For $i = 0$ to $L$ and for $j = 1$ to $2^i$, compute $u_{i,j} = u_0^{\alpha_i^j}$. Compute $v_i = v^{\alpha_i}$ for $i = 0$ to $L$.

The public key of the bank is $bpk := \big(\lambda, \hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, p, H, H_1, L, g, g_A, g_B, g_0, g_1, g_2, g_3, g_4, u_0, g_U, g_T, [u_{i,1}, \ldots, u_{i,2^i}]_{i=0}^{i=L} \in \mathbb{G}_1, Y, h, h_1, h_2, h_3, v, v_0, \ldots, v_L \in \mathbb{G}_2, Z \in \mathbb{G}_3\big)$. The private key of the bank is $bsk := \big(X \in \mathbb{G}_1, y \in \mathbb{G}_2\big)$. *Remarks:* The $\alpha_i$'s are no longer needed and the bank shall delete them. Later we shall see knowledge of $\alpha_i$ helps breaking the balance property of the scheme, while, it does not help breaking anonymity or exculpability. Thus, we shall believe the bank to delete those values since keeping them is exactly against its interest.

**User Account Establishment.** User Alice chooses $x$ as her private key and computes $PK_{Alice} = \{g_U^x\}$. She sends $PK_{Alice}$ to the bank, along with the proof of correctness. The bank stores $PK_{Alice}$ as the identity of Alice in its database. Alice stores $(PK_{Alice}, x)$ as her key pair.

**Withdrawal Protocol.** To withdraw a wallet $\mathcal{W}$ from the bank, Alice first prepares a binary tree $T_w$ as follows. Randomly chooses $w \in \mathbb{Z}_p^*$. Set $k_{0,0} := w$. Execute algorithm ComputeAllNodeKey on $w, L$ and obtain all the node key $k_{i,j}$ of binary tree $T_w$. Then, she computes the accumulation of the node keys of each levels as follows. For $i = 0$ to $L$, she computes $V_{w,i} = u_0^{\prod_{j=0}^{2^i-1}(\alpha_i + k_{i,j})}$[5]. She computes the commitment of the binary tree $T_w$ and her private key $x$. This is done by randomly choosing $a_i, b_i' \in \mathbb{Z}_p^*$, computes $C_{w,i} = V_{w,i} g_A^{a_i}$, $D_{w,i} = g_0^{b_i'} g_B^{a_i}$. She sends $\big[C_{w,i}, D_{w,i}\big]_{i=0}^{i=L}$ to the bank.

With probability $1/2$, the bank will ask Alice to execute Inspection Routine. Alice has to reveal $T_w$, $a_i$, $b_i'$ for $i = 0, \ldots, L$ to the bank. The bank checks if Alice computes the values $V_{w,i}$'s honestly. If Alice is found dishonest, a fine of $2L2^L$ is deducted from Alice account. Otherwise, the withdrawal protocol is repeated from the beginning.

If Inspection Routine is not chosen to be carried out, Alice is required to send a proof of knowledge of representation of $D_{w,i}$ to the bank. The bank verifies the proof, randomly chooses $b_i'', c_i \in \mathbb{Z}_p^*$ for $i = 0$ to $L$ and computes $A_i = X(C_{w,i})^{c_i}$, $B_i = (g g_0^{b_i''} PK_{Alice} D_{w,i})^{\frac{1}{y+c_i}}$, $C_i = h^{c_i}$. Then bank sends $\big[(A_i, B_i, C_i, a_i, b_i'',)\big]_{i=0}^{i=L}$ to Alice.

Alice computes $b_i = b_i' + b_i''$ for $i = 0$ to $L$, checks, for $i = 0$ to $L$, if

$$\hat{e}(A_i, h) \stackrel{?}{=} Z_i \hat{e}(V_{w,i} {g_A}^{a_i}, C_i),$$
$$\hat{e}(B_i, C_i Y) \stackrel{?}{=} \hat{e}(g, h)\hat{e}(g_B, h)^{a_i}\hat{e}(g_0, h)^{b_i}\hat{e}(g_U, h)^x,$$

and set $\mathcal{W} := \Big(T_w, \big[(A_i, B_i, C_i, a_i, b_i)\big]_{i=0}^{i=L}\Big)$.

[5] This computation does not require knowledge of $\alpha_i$. It can be computed using $u_0^{\alpha_i}, \ldots, u_0^{\alpha_i^{2^i}}$

| Algorithm ComputeAllNodeKey<br>Input: $w$, $L$<br>Output: $k_{0,0}$, $k_{1,0}$, $k_{1,1}$, …, $k_{L,0}$, …, $k_{L,2^L-1}$ |
|---|
| $K_{0,0} := w$<br>For $i = 1$ to $L$<br>  For $j = 0$ to $2^i - 1$<br>  if $j$ mod $2 == 0$ \\ Left children of parent node<br>  $k_{i,j} := H_0(g_S^{K_{i-1,j/2}})$ \\ $H_0$ for left children<br>  else \\ Right children of parent node<br>  $k_{i,j} := H_1(g_S^{K_{i-1,j/2}})$ \\ $H_1$ for right children |

**Fig. 2.** Withdrawal Protocol - Computation of a Binary Tree

**Spend Protocol.** Alice with wallet $\mathcal{W}$ wishes to pay merchant Bob with $2^\ell$ ($\ell < L$) dollar in the spend protocol. Alice and Bob first agree on the transaction information $I$ which includes $\ell$ and Bob's identity. Alice chooses an unused node key of level $i := L - \ell$. Let $k_{i,j}$ be the node key being chosen.

1. Bob sends to Alice a random challenge $m$.
2. Alice computes $M = H(I, m)$. She computes serial number of the coin $S = g_S^{k_{i,j}}$ and security tag $T = PK_{Alice} g_T^{Mk_{i,j}}$. She also computes a proof of correctness $\Pi_S$ such that $S$, $T$ are correctly formed as follows:

$$SPK_{Spend}\Big\{(A_i, B_i, C_i, a_i, b_i, x, k_{i,j}, V_{w,i}, W_{i,j}) :$$

$$\hat{e}(A_i, h) = Z_i \hat{e}(V_{w,i} g_A^{a_i}, C_i) \ \wedge \ \hat{e}(B_i, C_i Y) = \hat{e}(g g_B^{a_i} g_0^{b_i} g_U^{x}, h) \ \wedge$$
$$S = g_S^{k_{i,j}} \ \wedge \ T = g_U^{x} g_T^{Mk_{i,j}} \ \wedge \ \hat{e}(W_{i,j}, v_i v^{k_{i,j}}) = \hat{e}(V_{w,i}, v)\Big\}(M),$$

   where $W_{i,j} = u_0^{\prod_{k=0, k\neq j}^{k=2^i-1}(\alpha_i + k_{i,k})}$. She sends \$ $:= (S, T, \Pi_S, I, m)$ to Bob.
3. Bob accepts the payment \$ if $\Pi_S$ is a valid proof statement.
4. Alice marks the node $N_{i,j}$, its ancestors and all its children in $T_w$ as used nodes.

*Remarks: Instantiation of SPK $\Pi_S$ is shown in Appendix B.*

**Deposit Protocol.** Bob with \$ from Alice approaches the bank in the deposit protocol. He submits \$ to the bank, who checks if $I$ matches the merchant identity and checks if $m$ has been used before. The bank credits Bob if both checks passes.

Let $2^\ell$ be the value of the coin and $i := L - \ell$. The bank executes algorithm ComputeAllSerials on $S$, $\ell$ and obtains $S_{L,0}, \ldots, S_{L,2^\ell}$. The bank then checks if $S_{L,0}$, $S_{L,2^\ell}$ is in its database of spent-coin serial numbers. If yes, it runs the RevokeDoubleSpender algorithm described below. Otherwise, it stores $S_{L,0}, S_{L,2^\ell}$, together with \$ in its database of spent-coin serial numbers.

Algorithm ComputeAllSerials
Input: $S$, $L$, $\ell$
Output: $S_{L,0}, \ldots, S_{L,2^\ell-1}$

For $j = 0$ to $2^\ell - 1$
  $Temp := S; index := j;$
  For $i = \ell + 1$ to $L$
    if $index$ mod 2 == 0 \\ Left children of parent node
      $Temp := g_S^{H_0(Temp)}$ \\ $H_0$ for left children
    else \\ Right children of parent node
      $Temp := g_S^{H_1(Temp)}$ \\ $H_1$ for right children
    $index := index/2;$
  $S_{L,j} := Temp;$

**Fig. 3.** Deposit Protocol - Computation of All Serial Numbers Associated with a Particular Coin

**RevokeDoubleSpender** Let $\$ := (S, T, \Pi_S, I, m)$ and $\$' := (S', T', \Pi_{S'}, I', m')$ be two coins such that one of the output from algorithm ComputeAllSerials is the same. Denote $M := H(I, m)$ and $M' = H(I', M')$. If both coins are of the same value, compute $PK := (\frac{T^{M'}}{T'^M})^{\frac{1}{M'-M}}$ and output $PK$ as the identity of the double-spender.

Without loss of generality, assume value of coin $\$$ is $2^\ell$ and value of coin $\$'$ is $2^{\ell'}$ such that $\ell > \ell'$. Let $S_{L,\alpha}$, $0 \leq \alpha \leq 2^\ell - 1$, be the output from ComputeAllSerials on $(S, L, \ell)$ such that $S_{L,\alpha}$ equals to one of the output serial numbers from ComputeAllSerials on $(S', L, \ell')$. Execute algorithm GetNodeKey with input $S, \alpha, L, \ell, \ell'$ and obtain $k$. Compute $PK := \frac{T'}{h^{M'k}}$ and output $PK$ as the identity of the double-spender.

Algorithm GetNodeKey
Input: $S$, $\alpha$, $L$, $\ell$, $\ell'$
Output: $K$

$Temp := S; index := \alpha;$
For $i = 1$ to $\ell - \ell'$
  if $index$ mod 2 == 0 \\ Left children of parent node
    $K := H_1(Temp, 0)$ \\ 0 for left children
    else \\ Right children of parent node
    $K := H_1(Temp, 1)$ \\ 1 for right children
  $index := index/2;$
  $Temp := g_S^K;$

**Fig. 4.** RevokeDoubleSpender Algorithm - Computation of a Node Key from a Parent Serial Number

**VerifyGuilt** The algorithm RevokeDoubleSpender can be executed by the public. Thus, a proof that the bank is outputting the double-spender honestly is to publish two double-spent transcript.

### 4.3 Security Analysis

Regarding the security of our construction, we have the follow theorem whose proof can be found in Appendix D.

**Theorem 1.** *Our construction is secure under the q-SDH assumption and the AWSM assumption in the random oracle model.*

## 5 Efficiency Analysis

Table 1 summarizes the complexities of different protocols of our scheme and the scheme in [12]. The cost of the protocol with pre-processing of our scheme is listed as a reference. It is somehow hard to quantify the exact cost of the spend protocol in [12] as the instantiation of the SPK is very complex. Furthermore, it involves $L + 1$ cyclic groups of different orders. We simplify the comparison by stating the total number of group elements needed. If the Strong RSA-based CL signature [10] is used, as stated in [12], the group $\mathbb{G}$ in the paper would be the group of quadratic residue modulus a safe-prime product $n$, which would be of 1024-bit. $t$ is the security parameter controlling the cheating probability of the proof-of-knowledge of double-discrete logarithm. For example, $t = 80$ would give the protocol a cheating probability of $2^{-80}$.

For a moderate value $L = 10$ and $t = 40$, spending a coin of monetary value 1 in [12] requires 816 and 857 multi-based exponentiations from the user and the merchant respectively, and a total bandwidth of 981 elements in $\mathbb{Z}^*_{|\mathbb{G}|}$ and 28 elements in $\mathbb{G}$. If the base group is of order $n$ which is 1024-bit, each of the above elements is at least 1024-bit in size. On a contrary, spending a coin of any monetary value in our scheme requires a constant cost of 21 and 13 multi-based exponentiations from the user and the merchant respectively. And a total bandwidth of 9 elements in $\mathbb{G}$ and 21 elements in $\mathbb{Z}^*_{|\mathbb{G}|}$ is needed.

## 6 Conclusion

We presented an efficient off-line divisible e-cash scheme which is *truly anonymous*. While [12] shows that *truly anonymous* off-line divisible e-cash can be constructed, in this paper, we provided one step further by providing an affirmative answer whether a practical and efficient off-line divisible e-cash can be constructed. Our scheme is very efficient and practical (c.f. [12]).

## Acknowledgments.

We would like to thank Qiong Huang and the anonymous reviewers of FC 2008 for their helpful comments and suggestions.

| Time Complexities | | | | | |
|---|---|---|---|---|---|
| | This paper | | | Canard *et. al.*[12] | |
| WithdrawalProtocol | User w/o Preproc. | User w/ Preproc. | Bank | User | Bank |
| multi-EXP | $2^{L+1}+9L+5$ | $2L+2$ | $2^{L+1}+8L+6$ | 2 | 3 |
| Pairing | $2L+2$ | $2L+2$ | 0 | 0 | 0 |
| SpendProtocol (coin of value $2^{L-i}$) | User w/o Preproc. | User w/ Preproc. | Merchant | User | Merchant |
| multi-EXP | 21 | 1 | 13 | $6+2ti+i$ | $2ti+i+t+7$ |
| Pairing | 6 | 0 | 8 | 0 | 0 |

| Space Complexities | | |
|---|---|---|
| WithdrawalProtocol | Total Bandwidth Required | Total Bandwidth Required |
| $\mathbb{G}$ element | $7L+7$ | 3 |
| $\mathbb{Z}^*_{\lvert\mathbb{G}\rvert}$ element | $7L+8$ | 2 |
| SpendProtocol (coin of value $2^{L-i}$) | Total Bandwidth Required | Total Bandwidth Required |
| $\mathbb{G}$ element | 9 | $2(i+1)+6$ |
| $\mathbb{Z}^*_{\lvert\mathbb{G}\rvert}$ element | 21 | $2ti+4t+i+11$ |

**Table 1.** Time and Space Complexities of this paper and [12].

## References


1. G. Ateniese, J. Camenisch, and B. de Medeiros. Untraceable rfid tags via insubvertible encryption. In *ACM Conference on Computer and Communications Security*, pages 92–101, 2005.
2. M. H. Au, W. Susilo, and Y. Mu. Practical compact e-cash. In *ACISP*, pages 431–445, 2007.
3. M. H. Au, W. Susilo, and Y. Mu. Practical anonymous divisible e-cash from bounded accumulators. In *Financial Cryptography and Data Security*, 2008. to appear.
4. M. H. Au, Q. Wu, W. Susilo, and Y. Mu. Compact e-cash from bounded accumulator. In *CT-RSA*, pages 178–195, 2007.
5. N. Bari and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.
6. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
7. J. C. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In *EUROCRYPT*, pages 274–285, 1993.
8. D. Boneh and X. Boyen. Short signatures without random oracles. In *EUROCRYPT*, pages 56–73, 2004.
9. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *EUROCRYPT*, pages 302–321, 2005.
10. J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In *SCN*, pages 268–289, 2002.
11. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO*, pages 410–424, 1997.

12. S. Canard and A. Gouget. Divisible e-cash systems can be truly anonymous. In *EUROCRYPT*, pages 482–497, 2007.
13. S. Canard and J. Traoré. On fair e-cash systems based on group signature schemes. In *ACISP*, pages 237–248, 2003.
14. A. H. Chan, Y. Frankel, and Y. Tsiounis. Easy come - easy go divisible cash. In *EUROCRYPT*, pages 561–575, 1998.
15. D. Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203. Plenum, New York, 1983.
16. S. D'Amiano and G. D. Crescenzo. Methodology for digital money based on general cryptographic tools. In *EUROCRYPT*, pages 156–170, 1994.
17. T. Eng and T. Okamoto. Single-term divisible electronic coins. In *EUROCRYPT*, pages 306–319, 1994.
18. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
19. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, pages 291–304, 1985.
20. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, 1988.
21. T. Nakanishi and Y. Sugiyama. Unlinkable divisible electronic cash. In *ISW*, pages 121–134, 2000.
22. L. Nguyen. Accumulators from Bilinear Pairings and Applications. In *CT-RSA*, pages 275–292, 2005.
23. T. Okamoto. An efficient divisible electronic cash scheme. In *CRYPTO*, pages 438–451, 1995.
24. T. Okamoto and K. Ohta. Universal electronic cash. In *CRYPTO*, pages 324–337, 1991.
25. J. C. Pailles. New protocols for electronic money. In *ASIACRYPT*, pages 263–274, 1992.
26. C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.
27. V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

## A ESS+ Signature.

Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair as discussed. Let $g, g_A, g_B, g_0, g_1, \ldots, g_k$ be random elements of $\mathbb{G}_1$, $h$ be random elements in $\mathbb{G}_2$. The signing key is a pair $(X, y) \in (\mathbb{G}_1 \times \mathbb{Z}_p^*)$. The corresponding public key is $(Z = \hat{e}(X, h), Y = h^y) \in (\mathbb{G}_3 \times \mathbb{G}_2)$.

To sign a block of messages $(M, m_1, \ldots, m_k) \in (\mathbb{G}_1 \times (\mathbb{Z}_p)^k)$, the signer randomly chooses $a, b, c \in \mathbb{Z}_p^*$ and computes $A = X(Mg_A{}^a)^c \in \mathbb{G}_1$, $B = (gg_B{}^a g_0^b g_1^{m_1} \ldots g_k{}^{m_k})^{\frac{1}{y+c}} \in \mathbb{G}_1$, $C = h^c \in \mathbb{G}_2$. The signature on $(M, m_1, \ldots, m_k)$ is $(A, B, C, a, b)$.

To verify a signature, the verifier checks if the following holds:

$$\hat{e}(A, h) \stackrel{?}{=} Z\hat{e}(Mg_A{}^a, C)$$

$$\hat{e}(B, CY) \stackrel{?}{=} \hat{e}(g, h)\hat{e}(g_B, h)^a \hat{e}(g_0, h)^b \hat{e}(g_1, h)^{m_1} \ldots \hat{e}(g_k, h)^{m_k}$$

Note that the pairings $\hat{e}(g,h)$, $\hat{e}(g_i,h)$ $(i=0,\ldots,k)$ can be pre-computed or regarded as public parameters.

In the signature generation protocol, a user obtains a signature on ($M$, $m_1$, …, $m_k$) such that the signer learns nothing about the messages being signed. This protocol is different from the one in [4]. In [4], the protocol is somehow "hybrid" such that $M$, but not $m_1,\ldots,m_k$, is known to the signer. The protocol consists of three steps.

1. The user computes $(C_1,C_2)=(M(g_A)^a, {g_B}^a g_0^{b'} {g_1}^{m_1}\ldots{g_k}^{m_k})$ for some randomly generated $a,b'\in\mathbb{Z}_p^*$. The user sends $(C_1,C_2,\pi_{C_2})$ to the signer, where $\pi_{C_2}$ is an non-interactive proof of knowledge of the presentation of $C_2$ to bases $g_B,g_0,g_1,\ldots,g_k$[6].
2. The signer verifies $\pi_{C_2}$ and proceeds only if it is valid. it selects $c,b''\in\mathbb{Z}_p^*$ at random, computes $A=X(Mg_A{}^a)^c\in\mathbb{G}_1$, $B=(gg_0^{b''}C_2)^{\frac{1}{y+c}}\in\mathbb{G}_1$ and $C=h^c\in\mathbb{G}_2$. He sends $(A,B,C,b'')$ to the user.
3. The user computes $b=b'+b''$ and sets the signature as $(A,B,C,a,b)$

The signature possession protocol allows a user to demonstrate possession of a message signature pair to any third parties without revealing the signature nor the messages signed. It consists of three steps and can be converted into non-interactive form using Fiat-Shamir heuristic [18]. The details of the construction can be found in [4].

## B Instantiation of SPK

The details of SPK $\Pi_S$ in SpendProtocol is presented as follows.

*$SPK\Pi_S$ Signing.* To produce a proof $\Pi_S$ for on message $M$, do the following.

1. Produce auxiliary commitments ($\tilde{T}_A$, $\tilde{T}_B$, $\tilde{T}_C$, $\tilde{T}_V$, $\tilde{T}_W$, $\tilde{T}_1$, $\tilde{T}_2$) by randomly picking $\rho_1$, $\rho_2$, $\rho_3$, $\rho_4$, $\rho_5$, $\rho_6$, $\rho_7\in_R\mathbb{Z}_p^*$ and computing $\tilde{T}_A=A_i{g_1}^{\rho_1}$, $\tilde{T}_B=B_i g_2^{\rho_2}$, $\tilde{T}_C=C_i h_1^{\rho_3}$, $\tilde{T}_V=V_{w,i}g_3^{\rho_4}$, $\tilde{T}_W=W_{i,j}g_4^{\rho_5}$, $\tilde{T}_1=h_2^{\rho_3}h_3^{\rho_6}$ and $\tilde{T}_2=h_2^{\rho_5}h_3^{\rho_7}$.
2. Return $\Pi_S$ as ($\tilde{T}_A$, $\tilde{T}_B$, $\tilde{T}_C$, $\tilde{T}_V$, $\tilde{T}_W$, $\tilde{T}_1$, $\tilde{T}_2$, $\Pi_I$), where $\Pi_I$ is a signature proof of knowledge of:

$$SPK_I\Big\{(a_i,b_i,x,k_{i,j},\rho_1,\rho_2,\rho_3,\rho_4,\rho_5,\rho_6,\rho_7,\beta_1,\beta_2,\beta_3,\beta_4,\beta_5,\beta_6,\beta_7,\beta_8):$$

$$\tilde{T}_1=h_2^{\rho_3}h_3^{\rho_6}\ \wedge\ \tilde{T}_2=h_2^{\rho_5}h_3^{\rho_7}\ \wedge\ 1=\tilde{T}_1^{-\rho_4}h_2^{\beta_1}h_3^{\beta_2}\ \wedge$$
$$1=\tilde{T}_1^{-a_i}h_2^{\beta_3}h_3^{\beta_4}\wedge 1=\tilde{T}_1^{-\rho_2}h_2^{\beta_5}h_3^{\beta_6}\wedge 1=\tilde{T}_2^{-k_{i,j}}h_2^{\beta_7}h_3^{\beta_8}\ \wedge$$
$$\frac{\hat{e}(\tilde{T}_A,h)}{Z_i\hat{e}(\tilde{T}_V,\tilde{T}_C)}=\hat{e}_{1h}^{\rho_1}\hat{e}_{31}^{\beta_1}\hat{e}(g_A,\tilde{T}_C)^{a_i}\hat{e}(\tilde{T}_V,h_1)^{-\rho_3}\hat{e}(g_3,\tilde{T}_c)^{-\rho_4}\hat{e}_{A1}^{-\beta_3}\ \wedge$$
$$\hat{e}(\tilde{T}_B,\tilde{T}_C)\hat{e}(\tilde{T}_B,Y)\hat{e}_{gh}^{-1}=\hat{e}_{Bh}^{a_i}\hat{e}_{0h}^{b_i}\hat{e}_{Uh}^{x}\hat{e}(g_2,\tilde{T}_C)^{\rho_2}\hat{e}(\tilde{T}_B,h_1)^{\rho_3}\hat{e}_{2Y}^{\rho_2}\hat{e}_{21}^{-\beta_5}\ \wedge$$

[6] Formally, $SPK\{(b',m_1,\ldots,m_k):C_2={g_B}^a g_0^{b'}{g_1}^{m_1}\ldots{g_k}^{m_k}\}(R)$ for some randomly generated $R$ will serve the purpose.

$$S = g_S^{k_{i,j}} \ \wedge \ T = g_U^x (g_T^M)^{k_{i,j}} \ \wedge \ \frac{\hat{e}(\tilde{T}_V, v)}{\hat{e}(\tilde{T}_W, v_i)} = \hat{e}(\tilde{T}_W, v)^{k_{i,j}} \hat{e}_{3v}^{\rho_4} \hat{e}_{4v}^{-\beta_7} \hat{e}_{4v_i}^{-\rho_5} \Big\} (M)$$

on message $M$, which can be computed using the knowledge of $a_i$, $b_i$, $x$, $k_{i,j}$, $\rho_1$, $\rho_2$, $\rho_3$, $\rho_4$, $\rho_5$, $\rho_6$, $\rho_7$, $\beta_1$, $\beta_2$, $\beta_3$, $\beta_4$, $\beta_5$, $\beta_6$, $\beta_7$ and $\beta_8$, where $\beta_1 = \rho_3\rho_4$, $\beta_2 = \rho_4\rho_6$, $\beta_3 = \rho_3 a_i$, $\beta_4 = \rho_6 a_i$, $\beta_5 = \rho_3\rho_2$, $\beta_6 = \rho_6\rho_2$, $\beta_7 = \rho_5 k_{i,j}$ and $\beta_8 = \rho_7 k_{i,j}$. In the above, we denoted $\hat{e}(g_1, h)$ as $\hat{e}_{1h}$, $\hat{e}(g_3, h_1)$ as $\hat{e}_{31}$, $\hat{e}(g_A, h_1)$ as $\hat{e}_{A1}$, $\hat{e}(g, h)$ as $\hat{e}_{gh}$, $\hat{e}(g_B, h)$ as $\hat{e}_{Bh}$, $\hat{e}(g_0, h)$ as $\hat{e}_{0h}$, $\hat{e}(g_U, h)$ as $\hat{e}_{Uh}$, $\hat{e}(g_2, Y)$ as $\hat{e}_{2Y}$, $\hat{e}(g_2, h_1)$ as $\hat{e}_{21}$, $\hat{e}(g_3, v)$ as $\hat{e}_{3v}$, $\hat{e}(g_4, v)$ and $\hat{e}_{4v}$, $\hat{e}(g_4, v_i)$ as $\hat{e}_{4v_i}$.[7]

*$SPK\Pi_I$ Signing.* To produce a signature proof of knowledge $\Pi_I$ on message $M$, do the following:

1. *(Commit.)* Pick $r_a, r_b, r_x, r_k, r_{\rho_1}, r_{\rho_2}, r_{\rho_3}, r_{\rho_4}, r_{\rho_5}, r_{\rho_6}, r_{\rho_7}, r_{\beta_1}, r_{\beta_2}, r_{\beta_3}, r_{\beta_4}, r_{\beta_5}, r_{\beta_6}, r_{\beta_7}, r_{\beta_8} \in_R \mathbb{Z}_p^*$ uniformly at random. Compute $T_1 = h_2^{r_{\rho_3}} h_3^{r_{\rho_6}}$, $T_2 = h_2^{r_{\rho_5}} h_3^{r_{\rho_7}}$, $T_3 = \tilde{T}_1^{-r_{\rho_4}} h_2^{r_{\beta_1}} h_3^{r_{\beta_2}}$, $T_4 = \tilde{T}_1^{-r_a} h_2^{r_{\beta_3}} h_3^{r_{\beta_4}}$, $T_5 = \tilde{T}_1^{-r_{\rho_2}} h_2^{r_{\beta_5}} h_3^{r_{\beta_6}}$, $T_6 = \tilde{T}_2^{-r_k} h_2^{r_{\beta_7}} h_3^{r_{\beta_8}}$,
   $T_7 = \hat{e}_{1h}^{r_{\rho_1}} \hat{e}_{31}^{r_{\beta_1}} \hat{e}(g_A, \tilde{T}_C)^{r_a} \hat{e}(\tilde{T}_V, h_1)^{-r_{\rho_3}} \hat{e}(g_3, \tilde{T}_C)^{-r_{\rho_4}} \hat{e}_{A1}^{-r_{\beta_3}}$,
   $T_8 = \hat{e}_{Bh}^{r_a} \hat{e}_{0h}^{r_b} \hat{e}_{Uh}^{r_x} \hat{e}(g_2, \tilde{T}_C)^{r_{\rho_2}} \hat{e}(\tilde{T}_B, h_1)^{r_{\rho_3}} \hat{e}_{2Y}^{r_{\rho_2}} \hat{e}_{21}^{-r_{\beta_5}}$, $T_9 = g_S^{r_k}$, $T_{10} = g_U^{r_x} (g_T^M)^{r_k}$ and $T_{11} = \hat{e}(\tilde{T}_W, v)^{r_k} \hat{e}_{3v}^{r_{\rho_4}} \hat{e}_{4v}^{-r_{\beta_7}} \hat{e}_{4v_i}^{-r_{\rho_5}}$.
2. *(Challenge.)* Compute $c$ as: $H(\tilde{T}_A, \tilde{T}_B, \tilde{T}_C, \tilde{T}_V, \tilde{T}_W, \tilde{T}_1, \tilde{T}_2, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10}, T_{11}, S, T, M)$.
3. *(Response.)* Compute $z_a = r_a - ca_i$, $z_b = r_b - cb_i$, $z_x = r_x - cx$, $z_k = r_k - ck_{i,j}$, $z_{\rho_i} = r_{\rho_i} - c\rho_i$ for $i = 1$ to 7 and $z_{\beta_i} = r_{\beta_i} - c\beta_i$ for $i = 1$ to 8.
4. *(Output.)* The signature proof of knowledge $\Pi_I$ on $M$ is

   $$\Pi_I = (c, z_a, z_b, z_x, z_k, z_{\rho_1}, z_{\rho_2}, z_{\rho_3}, z_{\rho_4}, z_{\rho_5}, z_{\rho_6}, z_{\rho_7}, z_{\beta_1}, z_{\beta_2}, z_{\beta_3}, z_{\beta_4}, z_{\beta_5}, z_{\beta_6}, z_{\beta_7}, z_{\beta_8}).$$

*$SPK\Pi_S$ Verification.* To verify a proof $\Pi_S := (\tilde{T}_A, \tilde{T}_B, \tilde{T}_C, \tilde{T}_V, \tilde{T}_W, \tilde{T}_1, \tilde{T}_2, \Pi_I)$ on message $M$, do the following:

1. Parse $\Pi_I$ as $(c, z_a, z_b, z_x, z_k, z_{\rho_1}, z_{\rho_2}, z_{\rho_3}, z_{\rho_4}, z_{\rho_5}, z_{\rho_6}, z_{\rho_7}, z_{\beta_1}, z_{\beta_2}, z_{\beta_3}, z_{\beta_4}, z_{\beta_5}, z_{\beta_6}, z_{\beta_7}, z_{\beta_8})$.
2. Compute $T_1' = \tilde{T}_1^c h_2^{z_{\rho_3}} h_3^{z_{\rho_6}}$, $T_2' = \tilde{T}_2^c h_2^{z_{\rho_5}} h_3^{z_{\rho_7}}$, $T_3' = \tilde{T}_1^{-z_{\rho_4}} h_2^{z_{\beta_1}} h_3^{z_{\beta_2}}$, $T_4' = \tilde{T}_1^{-z_a} h_2^{z_{\beta_3}} h_3^{z_{\beta_4}}$, $T_5' = \tilde{T}_1^{-z_{\rho_2}} h_2^{z_{\beta_5}} h_3^{z_{\beta_6}}$, $T_6' = \tilde{T}_2^{-z_k} h_2^{z_{\beta_7}} h_3^{z_{\beta_8}}$, $T_9' = S^c g_S^{z_k}$, $T_{10}' = T^c g_U^{z_x} (g_T^M)^{z_k}$,

   $$T_7' = \left( \frac{\hat{e}(\tilde{T}_A, h)}{Z_i \hat{e}(\tilde{T}_V, \tilde{T}_C)} \right)^c \hat{e}_{1h}^{z_{\rho_1}} \hat{e}_{31}^{z_{\beta_1}} \hat{e}(g_A, \tilde{T}_C)^{z_a} \hat{e}(\tilde{T}_V, h_1)^{-z_{\rho_3}} \hat{e}(g_3, \tilde{T}_C)^{-z_{\rho_4}} \hat{e}_{A1}^{-z_{\beta_3}},$$

   $$T_8' = \left( \frac{\hat{e}(\tilde{T}_B, \tilde{T}_C) \hat{e}(\tilde{T}_B, Y)}{\hat{e}_{gh}} \right)^c \hat{e}_{Bh}^{z_a} \hat{e}_{0h}^{z_b} \hat{e}_{Uh}^{z_x} \hat{e}(g_2, \tilde{T}_C)^{z_{\rho_2}} \hat{e}(\tilde{T}_B, h_1)^{z_{\rho_3}} \hat{e}_{2Y}^{z_{\rho_2}} \hat{e}_{21}^{-z_{\beta_5}}, \text{ and}$$

[7] Note that all these pairings can be pre-computed or regarded as part of the bank's public key.

$$T'_{11} = \left(\frac{\hat{e}(\tilde{T}_V, v)}{\hat{e}(\tilde{T}_W, v_i)}\right)^c \hat{e}(\tilde{T}_W, v)^{z_k} \hat{e}_{3v}^{z_{\rho_4}} \hat{e}_{4v}^{-z_{\beta_7}} \hat{e}_{4v_i}^{-z_{\rho_5}}.$$

3. Return `valid` if $c$ equals: $H(\ \tilde{T}_A,\ \tilde{T}_B,\ \tilde{T}_C,\ \tilde{T}_V,\ \tilde{T}_W,\ \tilde{T}_1,\ \tilde{T}_2,\ T'_1,\ T'_2,\ T'_3,\ T'_4,\ T'_5,\ T'_6,\ T'_7,\ T'_8,\ T'_9,\ T'_{10},\ T'_{11},\ S,\ T,\ M)$. Return `invalid` otherwise.

## C Security Games

We use a game-based approach to define the security formally. The adversary's capabilities are modeled by arbitrary and adaptive queries to oracles. The oracles are defined as follows.

- UserSetup. This oracle allows the adversary $\mathcal{A}$ to add a user into the system. Upon invocation, this oracle act as the bank and engage with the adversary $\mathcal{A}$ in the UserSetup protocol. The public key $pk$ presented by $\mathcal{A}$ is added to a set $\mathcal{X}_A$
- Withdrawal. This oracle allows the adversary $\mathcal{A}$ to withdraw money from an honest bank. Upon invocation, $\mathcal{A}$ presents a public key $pk \in \mathcal{X}_A$ and engages in the WithdrawalProtocol as a user to obtain a wallet. $\mathcal{A}$ can choose whether Inspection Routine is to be run or not.
- Spend. This oracle allows the adversary $\mathcal{A}$ to act as a merchant to receive payment from an honest user. Upon invocation, $\mathcal{A}$ specific the monetary value to be paid. The oracle stores the total monetary value paid to $\mathcal{A}$ in a counter $q_O$.
- Hash. $\mathcal{A}$ can ask for the values of the hash functions for any input.

*Statistical Balance.* What we wish to model in statistical balance is that the maximum gain of an adversary $\mathcal{A}$ is $P$ for an cheating WithdrawalProtocol request. (A cheating WithdrawalProtocol request is an attempt to run WithdrawalProtocol with the bank such that if Inspection Routine is being executed, the output will be `cheat`). Specifically, we wish to model the case when the adversary runs $q_1$ honest WithdrawalProtocol and $q_2$ cheating WithdrawalProtocol, he can only get $2^L q_1 + P q_2$ monetary value. To model this fact, we introduce a new entity called a moderator $\mathcal{M}$ whose only purpose is to decide if an withdrawal request is valid or not. By valid we mean if Inspection Routine is being run, the output shall be `pass`. Formally, the game statistical Balance is defined as follows.

**Definition 5 (Game Statistical Balance).**

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter $\lambda$ and runs BankSetupto generate bpk and also a master secret key bsk. $\mathcal{C}$ keeps bsk to itself and sends bpk to adversary $\mathcal{A}$ and moderator $\mathcal{M}$. $\mathcal{M}$ initialize a counter $W$ to* 0.
- (Probing Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles in an adaptive manner. For each Withdrawal query, $\mathcal{A}$ is also required to run Inspection Routine(in parallel) with moderator $\mathcal{M}$. If the output of Inspection Routine is `pass`, $\mathcal{M}$ sets $W := W + 2^L$. Otherwise, $\mathcal{M}$ sets $W := W + P$.*

– (End Game Phase.) *$\mathcal{M}$ outputs the value $W$. Let $q_O$ be the monetary value $\mathcal{A}$ obtains from oracle* Spend. *$\mathcal{A}$ wins the game if it can deposit $W + 1 + q_O$ dollar to $\mathcal{C}$ such that, on input any two of these deposits transcript,the* RevokeDoubleSpender *algorithm does not output any of the public keys presented during the Withdrawal Oracle query.*

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

*Anonymity.* The following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ formally defines anonymity.

**Definition 6 (Game Anonymity).**

– (Initialization Phase.) *The challenger $\mathcal{C}$ gives a sufficiently large security parameter $\lambda$ to $\mathcal{A}$. $\mathcal{A}$ generates bpk and bsk. $\mathcal{A}$ gives bpk to $\mathcal{C}$. Since $\mathcal{A}$ is in possession of bsk, only Hash oracle query is allowed in Game Anonymity.*
– (Challenge Phase.) *$\mathcal{C}$ chooses two public keys $PK$ and $PK'$ and presents them to $\mathcal{A}$. $\mathcal{C}$ runs the* WithdrawalProtocol *with $\mathcal{A}$ acting as bank to obtain several wallets $w_0, \cdots, w_t$ and $w'_0, \cdots, w'_t$ on behalf of the two public keys. $\mathcal{A}$ then acts as merchant and ask for spending from $\mathcal{C}$. $\mathcal{A}$ is allowed to specify which wallet $\mathcal{C}$ uses, with the restriction that it cannot ask $\mathcal{C}$ to over-spend any of the wallets. Finally, $\mathcal{C}$ randomly chooses one wallet $w$ from user $PK$ and one wallet $w'$ from user $PK'$ from the set of wallets that are legal for the challenge, flip a fair coin to decide to use $w$ or $w'$ for the challenge spending.*
– (End Game Phase.) *The adversary $\mathcal{A}$ decides which public key $\mathcal{C}$ uses.*

*$\mathcal{A}$ wins the above game if it guesses correctly. The advantage of $\mathcal{A}$ is defined as the probability that $\mathcal{A}$ wins minus $\frac{1}{2}$.*

*Exculpability.* The following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ formally defines exculpability.

**Definition 7 (Game Exculpability).**

– (Initialization Phase.) *The challenger $\mathcal{C}$ gives a sufficiently large security parameter $\lambda$ to $\mathcal{A}$. $\mathcal{A}$ then generates bpk and bsk. $\mathcal{A}$ gives bpk to $\mathcal{C}$. Since $\mathcal{A}$ is in possession of bsk, only Hash oracle query is allowed in Game Exculpability.*
– (Challenge Phase.) *$\mathcal{C}$ runs the* WithdrawalProtocol *for $q_j$ times with $\mathcal{A}$ acting as bank to obtain wallets $w_1, \cdots, w_{q_j}$. $\mathcal{A}$ then act as merchant and ask for spending from $\mathcal{C}$. $\mathcal{A}$ is allowed to specific which wallet $\mathcal{C}$ uses, with the restriction that it cannot ask $\mathcal{C}$ to over-spend any of the wallets. $\mathcal{A}$ can also ask to corrupt any of the user in the above withdrawal protocol. A corrupted user need to surrender its private key as well as the wallet to $\mathcal{A}$.*
– (End Game Phase.) *$\mathcal{A}$ outputs $(pk, \pi_D)$. $\mathcal{A}$ wins the game if* VerifyGuilt *on $(pk, \pi_D)$ outputs 1 such that pk is one of the public key presented during* WithdrawalProtocol *from $\mathcal{C}$ in the challenge phase and pk has not been corrupted.*

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

## D Proof of Theorem 1

We sketch the proof for Theorem 1 in the following three subsections, one for each security requirement.

*Statistical Balance.* Let $\mathcal{A}$ be an adversary that makes $q_O$ spend queries. We outline why the success probability of $\mathcal{A}$ is negligible under the $q_{2^L}$-SDH assumption and the $AWSM$ assumption, by constructing a simulator $\mathcal{S}$ acting as challenger $\mathcal{C}$ (and Moderator $\mathcal{M}$).

For each UserSetup query, $\mathcal{C}$ rewinds and extracts the secret key corresponding to the public key. For each withdrawal query, $\mathcal{A}$ is required to present a public key $pk \in \mathcal{X}_A$. Rewind and extract the representation of $D_{w,i}$. $\mathcal{C}$ (also act as moderator $\mathcal{M}$) obtains the values $w$ and randomness used in $C_{w,i}$ and $D_{w,i}$ that is used to compute the accumulator values. If the values does not match (or $\mathcal{A}$ refuse to give out those values), set $W := W + L2^L$. Otherwise, set $W := W + 2^L$. Since in both cases, representation of $D_{w,i}$ is known, and $x$ such that $pk = g_U^x$ is known, invoke the signing oracle of ESS+ Signature to finish the view of the withdrawal protocol of $\mathcal{A}$. If $w$ is correctly formed, $\mathcal{C}$ stores the $2^L$ serial numbers associated with in in $\mathcal{S}$.

For each spending query, $\mathcal{C}$ simply simulates the HVZK protocol using a value $w$ as a wallet secret. The simulation is perfect since the protocol is HVZK.

Finally, $\mathcal{A}$ deposits $W + q_O$ dollars. If the deposit transcript consists of coins of value $2^\ell$, generate the $2^\ell$ serial numbers associated with it. $\mathcal{A}$ wins the game either by (1) all the $W + q_O$ serial numbers during deposit are unique or (2) some of the serial numbers are duplicated but RevokeDoubleSpender on the corresponding deposit attempt does not output a $pk \in \mathcal{X}_A$. Now we are to analyze these two cases separately.

Case (1): Since that only $q_O$ serial numbers are given to $\mathcal{A}$ during the spend queries, $\mathcal{A}$ must have produce another $W$ serial numbers. Due to the soundness of the underlying proof of knowledge protocols, each valid withdrawal query only gives $\mathcal{A}$ $2^L$ valid serial numbers. Due to the bound of the bounded accumulator (under the $2^L$-SDH assumption), each cheating withdrawal query only gives $\mathcal{A}$ $L2^L$ valid serial numbers. Thus, $\mathcal{A}$ must have conducted a *false* proof as part of the signature of knowledge such that one of the following is fake:

1. Possession of an ESS+ Signature on block of messages $(V, b, x)$.
2. $k$ is a value in accumulator $V$
3. $S = g_S^k$.

The fake proof of possession of ESS+ Signature happens with negligible probability under the $AWSM$ assumption. The fake proof that $S$ (and $T$) are well-formed happens with negligible probability under the discrete logarithm assumption. Thus $\mathcal{A}$'s success probability is negligible in Case (1).

Case (2): We have shown in case (1) that $\mathcal{A}$ cannot convince an $\mathcal{S}$ to accept an invalid serial number with non-negligible probability. We now suppose duplicated $S$ are accepted. Due to the zero-knowledge property of the Spend Protocol, $\mathcal{A}$ learns nothing about the message-signature pair. Thus, $\mathcal{A}$ cannot produce valid

deposit using the same set of values from spend query twice except using identical transcripts, which shall be rejected.

It remains to show the associated $T$ is bounded by specification except with negligible probability so that the correctness of the RevokeDoubleSpender implies the recovering of $PK$. Due to the soundness of the proof of knowledge protocol, $T = g_U^x g_T^{Mk}$ is the only valid $T$ to accompany serial number $S = g_S^k$. Since $M$ is chosen by the random oracle, the two $M$'s shall be different in the two transactions. To deviate from these valid tags, $A$ must fake the proof during Spend Protocol which we have already shown to happen with negligible probability only.

Thus, $\mathcal{A}$ can only win if he can forge an ESS+ Signature or break the security property (bound) of the bounded accumulator. It happens with negligible probability under the $AWSM$ assumption and the $q$-SDH assumption.

*Anonymity.* To tell if two transactions are linked, the adversary needs to distinguish if two serial numbers are from the same wallet. Let $g, g^a, g^b, Z$ be the DDH problem instance. The simulator can set $g_T = g^a$[8] and during the challenge spending, set $(S, T)$ as $(g_S^k := g^b, g_U^x Z^R)$ and simulate the protocol. The protocol is indistinguishable from the real world for user with secret key $x$ if $Z = g^{ab}$ and it contains no information on any user if $Z$ is just a random element. Thus, using the adversary as a black-box, $\mathcal{S}$ can solve the DDH problem.

*Exculpability.* In RevokeDoubleSpender, both transcripts contain the proof of correctness of $T$, which involves proving knowledge of the user secret $x$. To slander an honest user, adversary without knowledge of user secret $x$ has to fake the knowledge of $T$ which involve knowledge of $x$ to base $g_U$. This happens with negligible probability under the discrete logarithm assumption.

[8] This requires the random elements in $bpk$ to be set as the output of some hash functions. This is a common practice since it ensure the bank does not know the relative discrete logarithm of those random elements.