

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 work is supported by ARC Linkage Project LP0667899 and ARC Discovery Grant DP0877123.

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^ℓ , $\ell \leq L$, coins together can be done more efficiently than repeating the spend protocol for 2^ℓ times.

A lot of divisible e-cash schemes exist 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 [9,4,3] 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 efficient withdrawal of 2^L coins, the construction in [12] requires a series of $L+2$ cyclic groups

($\mathbb{G} = \langle g \rangle, \mathbb{G}_1 = \langle g_1 \rangle, \dots, \mathbb{G}_{L+1} = \langle g_{L+1} \rangle$) such that $\mathbb{G}_i \subset \mathbb{Z}_{|\mathbb{G}_{i+1}|}^*$ for $i = 1$ to $L + 1$ and $\mathbb{G} \subset \mathbb{Z}_{|\mathbb{G}_1|}^*$ 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, \dots, 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, \dots, 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 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. Moreover, it requires a communication cost of more than 800 group elements (each group element shall be of size greater than 1kb). 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 [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 $O(1)$

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.

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, \dots, V_{L+1}) and obtains $L+1$ signatures. In the spending protocol, if a node of level ℓ 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_ℓ . 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, \dots, 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 definitions can be found in the full version of the paper [2].

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, \dots, m_L in a commitment. The signer learns nothing about m_1, \dots, 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 the full version of the paper [2].

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.

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 $i = 1 \dots, q$. Compute $v_1 = v_0^\alpha$. The public parameters is $(u_0, \dots, u_q, v_0, v_1)$.

To accumulate a set of q values (e_1, \dots, 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 α 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^λ , where λ 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 τ_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^ℓ ($\ell \leq L$ and is decided by the user) to \mathcal{M} . After the protocol, \mathcal{M} obtains a coin serial number S_ℓ , a proof of validity π_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_ℓ, π_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_ℓ , 2^ℓ serial numbers $\tilde{S}_1, \dots, \tilde{S}_{2^\ell}$. 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 π_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_ℓ, π_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.²
- (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 $P \leq KL2^L$.

² 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.

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 the full version [2] 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 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 $\mathbf{Acc}_1, \dots, \mathbf{Acc}_{L+1}$. 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 node $N_{0,0}$ 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.

For $i = 0$ to L , compute $V_i := \mathbf{Acc}_i.\mathbf{Accumulate}(k_{i,0}, \dots, k_{i,2^i-1})$. 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 $\sigma_i = \mathbf{ESS}+.\mathbf{Sign}(V_i, x)$ using the signature generation protocol of ESS+ Signature (so that \mathcal{B} learns nothing about V_i, x as discussed.) \mathcal{B} sends $\sigma := \{\sigma_0, \dots, \sigma_L\}$ back to Alice. Alice stores (σ, 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 $k_{i,0}, \dots, k_{i,2^i-1}$). 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^ℓ dollar where $\ell \leq L$. Alice and Bob agree on certain transaction information I which contains identity of Bob and the monetary value 2^ℓ . 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 π 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. $\mathbf{ESS}+.\mathbf{Verify}(\sigma_i, V_i, x) = 1$ (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 π 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.

The Deposit Procedure. Bob sends (S, T, π, R, ℓ) 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^ℓ . Let $N_{i,j}$ be the corresponding node of the binary tree. From S , the bank computes the 2^ℓ serial numbers corresponding to the leaves of subtree of node $N_{i,j}$ by repeatedly applying the functions $LF(\cdot)$, $RF(\cdot)$ and $g^{(\cdot)}$.

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

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

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.

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 λ be a security parameter. On input λ , generate a λ -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 $X \in \mathbb{G}_1$, $y, \alpha_0, \dots, \alpha_L \in \mathbb{Z}_p^*$. 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 $bpk := (\lambda, \hat{e}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, p, H, H_1, L, g, g_A, g_B, g_0, g_1, g_2, g_3, g_4, u_0, g_U, g_T, [u_{i,1}, \dots, u_{i,2^i}]_{i=0}^{i=L} \in \mathbb{G}_1, Y, h, h_1, h_2, h_3, v, v_0, \dots, v_L \in \mathbb{G}_2, Z \in \mathbb{G}_3)$. The private key of the bank is $bsk := (X \in \mathbb{G}_1, y \in \mathbb{G}_2)$. *Remarks:* The α_i 's are no longer needed and the bank shall delete them. Later we shall see knowledge of α_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 $k_{0,0} := w$ and obtain all node key $k_{i,j}$ of the binary tree T_w . The algorithm, denoted as `ComputeAllNodeKey`, can be found in the full version of the paper[2]. Then, she computes the accumulation of the node keys of each levels as follows. For $i = 0$ to L , she computes $V_{w,i} = u_0^{\prod_{j=0}^{2^i-1} (\alpha_i + k_{i,j})} g_3$. 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 $[C_{w,i}, D_{w,i}]_{i=0}^{i=L}$ 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 $i = 0, \dots, L$ 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 $[(A_i, B_i, C_i, a_i, b''_i)]_{i=0}^{i=L}$ 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 $\mathcal{W} := (T_w, [(A_i, B_i, C_i, a_i, b_i)]_{i=0}^{i=L})$.

³ This computation does not require knowledge of α_i . It can be computed using $u_0^{\alpha_i}, \dots, u_0^{\alpha_{2^i}}$.

Spend Protocol. Alice with wallet \mathcal{W} wishes to pay merchant Bob with 2^ℓ ($\ell < L$) dollar in the spend protocol. Alice and Bob first agree on the transaction information I which includes ℓ 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 $T = PK_{Alice} g_T^{M k_{i,j}}$. She also computes a proof of correctness Π_S such that S, T are correctly formed as follows:

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

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

$$S = g_S^{k_{i,j}} \wedge T = g_U^x g_T^{M k_{i,j}} \wedge \hat{e}(W_{i,j}, v_i v^{k_{i,j}}) = \hat{e}(V_{w,i}, v) \left. \right\} (M),$$

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 Π_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 Π_S is shown in the full version[2].

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^ℓ be the value of the coin and $i := L - \ell$. The bank compute all serial numbers accompanying S and obtains $S_{L,0}, \dots, S_{L,2^\ell}$. The algorithm, denoted as **ComputeAllSerials**, can be found in the full version of the paper[2]. 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.

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 $PK := \left(\frac{T^{M'}}{T^M}\right)^{\frac{1}{M'-M}}$ and output PK as the identity of the double-spender.

Without loss of generality, assume value of coin $\$$ is 2^ℓ 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, ℓ) such that $S_{L,\alpha}$ equals to one of the output serial numbers from **ComputeAllSerials** on (S', L, ℓ') . Compute K by applying H_0 or H_1 suitably such that $S = g_S^K$. The algorithm, denoted as **GetNodeKey** can be found in the full version of the paper[2]. Compute $PK := \frac{T'}{h^{M'k}}$ and output PK as the identity of the double-spender.

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

Time Complexities					
	This paper			Canard <i>et. al.</i> [12]	
Withdrawal Protocol	User		Bank	User	Bank
	w/o Preproc.	w/ Preproc.			
multi-EXP	$2^{L+1} + 9L + 5$	$2L + 2$	$2^{L+1} + 8L + 6$	2	3
Pairing	$2L + 2$	$2L + 2$	0	0	0
Spend Protocol (coin of value 2^{L-i})	User		Merchant	User	Merchant
	w/o Preproc.	w/ Preproc.			
multi-EXP	21	1	13	$6 + 2ti + i$	$2ti + i + t + 7$
Pairing	6	0	8	0	0
Space Complexities					
Withdrawal Protocol	Total Bandwidth Required			Total Bandwidth Required	
\mathbb{G} element	$7L + 7$			3	
$\mathbb{Z}_{ \mathbb{G} }^*$ element	$7L + 8$			2	
Spend Protocol (coin of value 2^{L-i})	Total Bandwidth Required			Total Bandwidth Required	
\mathbb{G} element	9			$2(i + 1) + 6$	
$\mathbb{Z}_{ \mathbb{G} }^*$ element	21			$2ti + 4t + i + 11$	

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 the full version of the paper[2].

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.

References

1. Ateniese, G., Camenisch, J., de Medeiros, B.: Untraceable rfid tags via insubvertible encryption. In: ACM Conference on Computer and Communications Security, pp. 92–101 (2005)
2. Au, M.H., Susilo, W., Mu, Y.: Practical anonymous divisible e-cash from bounded accumulators. Cryptology ePrint Archive, Report, 2007/459 (2007), <http://eprint.iacr.org/>
3. Au, M.H., Susilo, W., Mu, Y.: Practical compact e-cash. In: Pieprzyk, J., Ghodsi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 431–445. Springer, Heidelberg (2007)
4. Au, M.H., Wu, Q., Susilo, W., Mu, Y.: Compact e-cash from bounded accumulator. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 178–195. Springer, Heidelberg (2006)
5. Bari, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 480–494. Springer, Heidelberg (1997)
6. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: ACM Conference on Computer and Communications Security, pp. 62–73 (1993)
7. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures (extended abstract). In: EUROCRYPT, pp. 274–285 (1993)
8. Boneh, D., Boyen, X.: Short signatures without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 56–73. Springer, Heidelberg (2004)

9. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 302–321. Springer, Heidelberg (2005)
10. Camenisch, J., Lysyanskaya, A.: A Signature Scheme with Efficient Protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 268–289. Springer, Heidelberg (2003)
11. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: CRYPTO, pp. 410–424 (1997)
12. Canard, S., Gouget, A.: Divisible e-cash systems can be truly anonymous. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 482–497. Springer, Heidelberg (2007)
13. Canard, S., Traoré, J.: On fair e-cash systems based on group signature schemes. In: Safavi-Naini, R., Seberry, J. (eds.) ACISP 2003. LNCS, vol. 2727, pp. 237–248. Springer, Heidelberg (2003)
14. Chan, A.H., Frankel, Y., Tsiounis, Y.: Easy come - easy go divisible cash. In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 561–575. Springer, Heidelberg (1998)
15. Chaum, D.: Blind Signatures for Untraceable Payments. In: McCurley, K.S., Ziegler, C.D. (eds.) Advances in Cryptology 1981 - 1997. LNCS, vol. 1440, pp. 199–203. Springer, Heidelberg (1999)
16. D’Amiano, S., Crescenzo, G.D.: Methodology for digital money based on general cryptographic tools. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 156–170. Springer, Heidelberg (1995)
17. Eng, T., Okamoto, T.: Single-term divisible electronic coins. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 306–319. Springer, Heidelberg (1995)
18. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
19. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: STOC, pp. 291–304 (1985)
20. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. 17(2), 281–308 (1988)
21. Nakanishi, T., Sugiyama, Y.: Unlinkable divisible electronic cash. In: Okamoto, E., Pieprzyk, J.P., Seberry, J. (eds.) ISW 2000. LNCS, vol. 1975, pp. 121–134. Springer, Heidelberg (2000)
22. Nguyen, L.: Accumulators from Bilinear Pairings and Applications. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 275–292. Springer, Heidelberg (2005)
23. Okamoto, T.: An efficient divisible electronic cash scheme. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 438–451. Springer, Heidelberg (1995)
24. Okamoto, T., Ohta, K.: Universal electronic cash. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 324–337. Springer, Heidelberg (1992)
25. Pailles, J.C.: New protocols for electronic money. In: ASIACRYPT, pp. 263–274 (1992)
26. Schnorr, C.-P.: Efficient signature generation by smart cards. J. Cryptology 4(3), 161–174 (1991)