Practical Compact E-Cash

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. Compact e-cash schemes allow a user to withdraw a wallet containing k coins in a single operation, each of which the user can spend unlinkably. One big open problem for compact e-cash is to allow multiple denominations of coins to be spent efficiently without executing the spend protocol a number of times. In this paper, we give a (*partial*) solution to this open problem by introducing two additional protocols, namely, compact spending and batch spending. Compact spending allows spending all the k coins in one operation while batch spending allows spending any number of coins in the wallet in a single execution.

We modify the security model of compact e-cash to accommodate these added protocols and present a generic construction. While the spending and compact spending protocol are of constant time and space complexities, complexities of batch spending is linear in the number of coins to be spent together. Thus, we regard our solution to the open problem as *partial*.

We provide two instantiations under the q-SDH assumption and the LRSW assumption respectively and present security arguments for both instantiations in the random oracle model.

Keywords: E-Cash, constant-size, compact, bilinear pairings

1 Introduction

Electronic cash (e-cash) was invented by Chaum[12] in 1982. In its simplest form, an *e-cash* system consists of three parties (the bank $\mathcal{B}$, the user $\mathcal{U}$ and the shop $\mathcal{S}$) and four main procedures (account establishment, withdrawal, payment and deposit). The user $\mathcal{U}$ first performs an account establishment protocol with the bank $\mathcal{B}$. The currency circulating around is quantized as 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 $\mathcal{S}$. To deposit a coin, $\mathcal{S}$ performs a deposit protocol with $\mathcal{B}$.

Security of e-cash refers to the fact that only the bank $\mathcal{B}$ can produce a coin and for offline schemes, users who double-spent should be identified. The problem of double-spending only occurs in the electronic world due to easy duplication of digital coins. On the other hand, honest spenders cannot be slandered to have double spent (*exculpability*), and when the shops deposit the money from the payee, the bank should not be able to trace who the actual spender is (*anonymity).* Many e-cash systems that provide the function of identifying double-spenders have been proposed, but most of them rely on a trusted third party (TTP) to *revoke* the anonymity so as to identify the double-spenders [7, 18, 11]. While the TTP cannot slander an honest user, its existence in fact implies that even honest users are not anonymous.

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, payment and deposit protocols; (2) the size of an electronic coin; and (3) the size of the bank's database.

Camenisch, Hohenberger and Lysyanskaya [8] proposed a secure offline anonymous e-cash scheme (which we shall refer to as CHL scheme from now on) which is compact to address the efficiency issue. In their scheme, a wallet containing k coins can be withdrawn and stored in complexity $O(\lambda + \log(k))$ for a security parameter λ, where each coin can be spent unlinkably with complexity $O(\lambda + \log(k))$ as well. Au *et al.* [2] construct compact e-cash from another approach by using a bounded accumulator. However, both schemes involve extensive use of proof-of-knowledge and the exact cost of each operation is somehow hard to quantify.

Related Results. Compact e-cash scheme is closely related to k-TAA[21] and itself can be regarded as a multi-show credential system[13]. The main difference between a compact e-cash and a k-TAA is that in the former case, a token can only be used for a total of k times while in the latter, a token can be shown for k-times to each application provider where k is specified by each application provider independently. In some sense a k-TAA is more general. If the authentication of the k-TAA can be done non-interactively, that k-TAA scheme can be used as a compact e-cash system as follows. All shops play the role of a single application provider with k being specified by the bank, while the bank plays the role of a GM. A user withdraws a coin by obtaining a credential from the bank and spend the coin by authenticating himself to the shop non-interactively. The shop deposits by submitting the authentication transcript back to the bank.

CHL's Compact E-Cash. There are two versions of CHL's scheme. We describe the first one since it is simpler and much more efficient than the other. When a user withdraws a wallet, it obtains a signature σ from the bank on the commitment of user's secret key and two secret random numbers, s and t. To spend a coin, the user proves to the merchant that it possesses such a signature σ, and uses s to generate a link tag. By using a verifiable random function on s and counter i, the user generates a link tag S_i (called serial number). By the same method, the user also generates a blinding value B_i from t and counter i. For the counter i running from 0 to $2^\ell - 1$, the wallet can be spent 2^ℓ times unlinkably. The blinding value is used to compute the value $T_i = g^u B_i^R$, where g^u is the secret key of the user and R is provided by the merchant. T_i is called a double-spending equation. Should a user double-spent any of his coins, two double-spending equations can be used to reveal the public key of the user. It can be extended to incorporate the coin tracing protocol, such that all spendings of the double-spender can be traced, by making the XDH assumption.

Overview of Our Scheme. One big open problem mentioned in the CHL's paper is how to spend several coins in the wallet together efficiently. *For the first time*, we address this open problem and propose two protocols, namely, *batch spending* and *compact spending*. In *batch spending*, the user pays an arbitrary number of coins to the merchant while in *compact spending*, the user spends the whole wallet in a single execution. Our result is based on the CHL scheme. Our main idea is that, submitting s and t directly essentially gives away all the coins in the wallet since everybody could then compute all the S_i and T_i. Now the remaining problem is to prevent the user from *double compact spends*. We solve the problem by introducing a third random number y during withdrawal. Whenever the user executes compact spend, we use y to compute a blinding value C_i and uses it to compute $T_c = g^u C_i^R$. Double compact spend shall then be identified with duplicated (s, t) and from two double-spending equation T_c, the double-spender will be caught. A user commits double-spending by first doing a normal spending followed by a compact spending shall be identified and caught as follows. Firstly, from the value s, the bank can compute all S_i and uses it to identify the double-spent coin. Then from the value t and i, the bank computes B_i, the blinding value of the user's public key in T_i, and uses it to identify the double-spender.

We optimize several spend protocols together into batch spending protocol. Finally, we make use of the idea from [22] to further improve the efficiency of the scheme from $O(\lambda + \log(k))$ to $O(\lambda)$. CHL scheme restricts the coin to be k-spendable by requiring the user to prove that the counter i is within 1 to k with the smart use of the result from Boudot[6]. Boudot's result is of complexity $O(\log(k))$ and works in groups of unknown order. Consequently, a natural choice of the bank's signature in CHL's scheme is the strong RSA-based CL signature[9]. With the idea from [22], we are able to make use of more efficient signature schemes from elliptic curve group such as CL+[10] or BBS+[1]. We shall see in the efficiency analysis section that, all our protocols except batch spending, are of constant size and computational cost. While the time and space complexities of batch spending is linear in the number of coins spent, the number of pairing computation required is constant. In fact, spending an extra coin in the batch spending protocol only adds 43 bytes and 4 multi-based exponentiation in the overhead.

Concerns About q-SDH Assumption. Due to recent concern about the q-SDH assumption[14], we also present a construction from the LRSW-based CL+ signature[10]. The scheme, however, is less efficient computationally.

Our Contributions. Specifically, we make the following contributions

- We solve an open problem stated in the CHL paper by introducing the idea of compact spending and batch spending into compact e-cash systems.
- We present generic construction of compact e-cash system with these two added protocols and propose two instantiations
- We formalize a model to accommodate batch spending and compact spending protocols into compact e-cash schemes and present security arguments for our schemes.
- We outline how size of the wallet can be chosen arbitrarily by users while preserving user privacy during spending.

Organization. We discuss related works and technical preliminaries in the next section. A security model is shown in Section 3. The construction is shown in Section 4, accompanied by security analysis. Finally we conclude in Section 5.

2 Preliminaries

2.1 Notations

Let $\hat{e}$ be a bilinear map such that $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

- $\mathbb{G}_1$ and $\mathbb{G}_2$ are cyclic multiplicative groups of prime order p.
- each element of $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ has unique binary representation.
- g_0, h_0 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.
- $\psi : \mathbb{G}_2 \rightarrow \mathbb{G}_1$ is a computable isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$, with $\psi(h_0) = g_0$.
- (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)$\hat{e}(g_0, h_0) \neq 1$.

$\mathbb{G}_1$ 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 $\mathbb{G}_1$, $\mathbb{G}_2$, the isomorphism ψ and the bilinear mapping $\hat{e}$ are all efficiently computable.

2.2 Mathematical Assumptions

Definition 1 (Decisional Diffie-Hellman). *The Decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ is defined as follow: 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 (q-Strong Diffie-Hellman[3]). *The q-Strong Diffie-Hellman (q-SDH) problem in $(\mathbb{G}_1, \mathbb{G}_2)$ is defined as follow: 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 3 (y-Decisional Diffie-Hellman Inversion Assumption[15, 8]). *The y-Decisional Diffie-Hellman Inversion problem (y-DDHI) in prime order group $\mathbb{G}$ is defined as follow: On input a $(y+2)$-tuple $g, g^x, g^{x^2}, \cdots, g^{x^y}, g^c \in \mathbb{G}^{y+2}$, output 1 if $c = 1/x$ and 0 otherwise. We say that the y-DDHI assumption holds in $\mathbb{G}$ if no PPT algorithm has non-negligible advantage over random guessing in solving the y-DDHI problem in $\mathbb{G}$.*

Definition 4 (LRSW Assumption[17]). *The LRSW problem in prime order group $\mathbb{G}$ is defined as follow: Let $\mathbb{G} = \langle g \rangle$ be a prime order cyclic group of order p and $u = g^x, v = g^y$. Define $O_{u,v}(\cdot)$ as an oracle such that on input a value $m \in \mathbb{Z}_p$, output (a, a^y, a^{x+mxy}) for a randomly chosen $a \in \mathbb{G}$. The problem is on input g, u, v, and the oracle $O_{u,v}(\cdot)$, output (m, a, b, c) such that $m \neq 0 \ \wedge \ a \in \mathbb{G} \ \wedge \ b = a^y \ \wedge \ c = a^{x+mxy}$ and m has not been input to $O_{u,v}(\cdot)$. We say that the LRSW assumption holds in $\mathbb{G}$ if no PPT algorithm has non-negligible advantage in solving the LRSW problem in $\mathbb{G}$.*

Definition 5 (eXternal Diffie-Hellman). *The eXternal Diffie-Hellman (XDH) problem in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ is defined as solving the DDH problem in $\mathbb{G}_1$ given the following three efficient oracles*

1. *solving DDH problem in $\mathbb{G}_2$,*
2. *computing the isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$,*
3. *and computing the bilinear mapping of groups $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$.*

We say that the XDH assumption holds in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ if no PPT algorithm has non-negligble advantage in solving the XDH problem in $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$.

The above assumption implies that the isomorphism is computationally one-way, i.e. there does not efficient way to complete $\psi^{-1} : \mathbb{G}_1 \rightarrow \mathbb{G}_2$. This has proven to be false in supersingular curves while it is conjectured to hold over MNT curves. See [4] for a more throughout discussion.

2.3 Building Blocks

Verifiable Random Function. One of the building blocks of our e-cash system is the verifiable random function (VRF) from [15], which we shall refer to as DY VRF. The notion VRF was introduced in [19]. Roughly speaking, a VRF is a pseudo-random function with non-interactive proof of correctness of its output. The VRF defined in [15] is described as follow. The function f is defined by a tuple $(\mathbb{G}_p, p, g, s)$, where $\mathbb{G}_T$ is a cyclic group of prime order p, g a generator of $\mathbb{G}_p$ and s is a seed in $\mathbb{Z}_p$. On input x, $f_{\mathbb{G}_p,p,g,s}(x) = g^{\frac{1}{s+x+1}}$. Efficient proof such that the output is correctly formed (with respect to s and x in some commitment scheme such as Pedersen Commitment [20]) exists and the output of f is indistinguishable from random elements in $\mathbb{G}_p$ if the y-DDHI assumption in $\mathbb{G}_p$ holds.

Signature with Efficient Protocols. A signature scheme with efficient protocols refers to signature scheme with the following two protocols: (1) a protocol between a user and a signer with keys (pk, sk). Both the user and the signer agreed on a commitment scheme such as Pedersen commitment. The user input is a block of messages $(m_1, \cdots, m_L)$ and a random value r such that C=Commit$(m_1, \cdots, m_L, r)$. After executing the protocol, the user obtains a signature on $(m_1, \cdots, m_L)$ from the signer while the signer learns nothing about the block of messages; (2) a protocol to prove the knowledge of a signature. This allows the user to prove to a verifier that he is in possession of a signature. Examples include CL signature, CL+ signature [9, 10] and a modification of the short group signature from Boneh *et al.*[4] that is called BBS+[1].

3 Security Model

3.1 Syntax

A compact e-cash system with compact spending and batch spending is a tuple (BankSetup, UserSetup, WithdrawalProtocol, SpendProtocol, BSpendProtocol, CSpendProtocol, DepositProtocol, RevokeDoubleSpender, VerifyGuilt) of nice polynomial time algorithms/protocols between three entities, namely Bank, Merchant and User. The following enumerates the syntax.

– BankSetup. On input an unary string 1^λ, where λ is the security parameter, the algorithm outputs the bank's master secret bsk and the public parameter bpk.

- UserSetup. On input bpk, outputs a key pair (pk, sk). Since merchants are a subset of users, they may use this algorithm to obtain keys as well.
- WithdrawalProtocol. The user with input (pk, sk) withdraws a wallet w of k coins from the bank. The bank's input is the master secret bsk. After executing the protocol, the user obtains a wallet w while the bank (possibly) retains certain information τ_w, called the trace information.
- SpendProtocol. This is the normal spend protocol when the user spends a single coin to a merchant. The user input is w and the merchant's identity. After the protocol, the merchant obtains a transcript including a proof of validity π of a coin from the wallet, and possibly some auxiliary information aux, and outputs 0/1, depending whether the payment is accepted. The user's output is an updated wallet w'.
- BSpendProtocol. This is the batch spend protocol when the user spends n coins, $n < k$, together to a merchant. The user input is w and the merchant's identity. After the protocol, the merchant obtains a transcript including a proof of validity π of n coin from the wallet, and possibly some auxiliary information aux, and outputs 0/1, depending whether the payment is accepted. The user's output is an updated wallet w'.
- CSpendProtocol. This is the compact spend protocol when the user spends all k coins in his wallet w together to a merchant. The user input is w and the merchant's identity. After the protocol, the merchant obtains a transcript including a proof of validity π of a wallet w, and possibly some auxiliary information aux, and outputs 0/1, depending whether the payment is accepted.
- DepositProtocol. In a deposit protocol, the merchant submits (π, aux) to the bank for deposit. The bank outputs 0/1, indicating whether the deposit is accepted. It is required whenever an honest merchant obtains (π, aux) by running any of the spend protocols with some user, there is a guarantee that this transaction will be accepted by the bank. The bank adds (π, aux) to the database of spent coins.
- RevokeDoubleSpender. Whenever a user double spends, this algorithm allows the bank to identify the double spender. Formally, on input two spending protocol transcripts involving the same coin, the algorithm outputs the public key pk of the double-spender. Intuitively, there are three possible cases for a user to double-spend, namely, normal spend twice (or batch spend involving same coin), compact spend twice, or normal spend (or batch spend) and then compact spend or vice versa. The bank also output a proof π_D to prove that user pk indeed double-spends.
- 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 output π_D and pk of the double-spender, everyone can check if the bank is honest.

SpendProtocol, BSpendProtocol and CSpendProtocol shall be collectively called spend protocols. In situations where ambiguity may arise, we shall refer to executing SpendProtocol as normal spending.

Remarks: We omit the Trace and VerifyOwnership algorithm defined in the CHL paper because our system does not support it, just as the first version in the CHL paper. We should remark, however, we can extend our system using the same technique as in the CHL paper to support these two algorithms. Details of extension can be found in the appendix.

3.2 Security Notions

We first provide an informal description of the security requirements. A *secure* compact e-cash scheme should possess *correctness*, *balance*, *anonymity* and *exculpability*, introduced as follows.

- *Correctness.* If an honest user runs WithdrawalProtocol with an honest bank and runs any of the spend protocols with an honest merchant, the merchant accepts the payment. The merchant later runs Deposit with the bank, which will accept the transaction.
- *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. More precisely, we require that collusion of users and merchants, having run the withdrawal

protocol for n times, cannot deposit more than nk coins back to the bank. In case they do deposit $nk + 1$ coins, at least one of the colluders must be identified. A related notion is revocability, which means identity of the double-spender must be revoked. It is straight forward to see that revocability is implied by the definition of *balance*.
- *Anonymity.* It is required that no collusion of users, merchants and the bank can ever learn the spending habit of an honest user.
- *Exculpability.* It is required that an honest user cannot be accused of having double-spent, even all other users, merchants and the bank colludes.

From our definition, it can be seen that it is the bank's responsibility to identify the double-spender. The rationale behind this is that a user can always spend the same coin to different merchants in an offline e-cash system and the merchant has no way to detect such double-spending.

Next we are going to formally define the security model. While the model in CHL uses the UC framework, our model is game-based.

The capability of an adversary $\mathcal{A}$ is modeled as oracles.

- Withdrawal Oracle: $\mathcal{A}$ presents a public key pk and engages in the WithdrawalProtocol as user and obtains a wallet. The oracle stores pk in a set $\mathbb{X}_A$.
- Spend Oracle: $\mathcal{A}$ now acts as a merchant and request users to spend coins with it. It can request for CSpend, BSpend or normal Spend for any user of its choice.
- Hash Oracle: $\mathcal{A}$ can ask for the values of the hash functions for any input.

We require that the answers from the oracles are indistinguishable from the view as perceived by an adversary in real world attack.

Definition 6 (Game Balance).

- (Initialization Phase.) *The challenger $\mathcal{C}$ takes a sufficiently large security parameter λ and runs* BankSetup *to generate bpk and also a master secret key bsk. $\mathcal{C}$ keeps bsk to itself and sends bpk to $\mathcal{A}$.*
- (Probing Phase.) *The adversary $\mathcal{A}$ can perform a polynomially bounded number of queries to the oracles in an adaptive manner.*
- (End Game Phase.) *Let q_w be the number of queries to the Withdrawal Oracle and q_s be the number of queries to the Spend Oracle. Note that a compact spending query to the Spend Oracle is counted as k queries and a batch spending of n coins query is counted as n queries. $\mathcal{A}$ wins the game if it can run kq_w+q_s+1 deposit to $\mathcal{C}$ such that $\mathcal{C}$ cannot point to any of the users during the Withdrawal Oracle query by running* RevokeDoubleSpender.

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

Definition 7 (Game Anonymity).

- (Initialization Phase.) *The challenger $\mathcal{C}$ gives a sufficiently large security parameter λ 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 Anonymity.*
- (Challenge Phase.) *$\mathcal{C}$ then 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, where t and t' are specified by $\mathcal{A}$. $\mathcal{A}$ 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, $\mathcal{A}$ chooses a type of spending (normal spend, BSpend or CSpend) as challenge. $\mathcal{A}$ also 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 (for example, if wallet w_0 has spent $k-1$ times already and BSepnd 2 coins is chosen as the challenge, $\mathcal{A}$ cannot specific wallet w_0). $\mathcal{C}$ then flips 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 $\frac{1}{2}$.

Definition 8 (Game Exculpability).

– (Initialization Phase.) *The challenger $\mathcal{C}$ gives a sufficiently large security parameter λ 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 needs to surrender its private key as well as the wallet to $\mathcal{A}$.*
– (End Game Phase.) *$\mathcal{A}$ runs two deposit protocol with $\mathcal{C}$. $\mathcal{A}$ wins the game if* RevokeDoubleSpender *on this two deposit protocol points to a user in* any *of the withdrawal protocol during initialization and that user has not been corrupted.*

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

A compact e-cash scheme with compact spending is *secure* if no PPT adversary can win in Game Balance, Game Anonymity and Game Exculpability with non-negligible advantage.

4 Our Constructions

4.1 Generic Construction

BankSetup. Let $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme with efficient protocols as discussed. Let $\mathrm{Vrf}(\cdot)$ be an verifiable random function as discussed. The bank generates the parameter of a signature scheme with efficient protocols using KeyGen and is in possession of the signing key. It also publishes, preferably using another key pair of the signature scheme, $\sigma_1 = \mathsf{Sign}(1), \cdots, \sigma_k = \mathsf{Sign}(k)$. Each user is in possession of a DL type key pair (x, u^x).

Withdrawal. To withdraw, the user obtains a signature $\sigma_x = \mathsf{Sign}(s, t, x, y, r)$ using the signature generation protocol. The banks learns nothing about the block of messages (s, t, x, y, r). The User keeps $(\sigma_x, s, t, x, y, r)$ as its wallet secret and sets the counter $J = 1$.

Spend Protocols. For payment, the user and the merchant with identity $I \in \{0,1\}^*$ first agree on the transaction information info. Then, they compute $R = H(\text{info}, I)$ locally, for some cryptographic hash function H.

Spend. To spend a single coin, the user then sends to the merchant C which is a commitment of (s, t, x, y, r, J) and also $S = \mathsf{Vrf}(s, J)$, $T = PK\mathsf{Vrf}(t, J)^R$. Note that $PK = u^x$ and $\mathsf{Vrf}(s, x)$ denotes the verifiable random function as discussed on input x with respect to seed s. It then sends the following signature of knowledge to the merchant.

$$\Pi_{\mathsf{Spend}} : SPK\Big\{(\sigma_x, s, t, x, y, r, \sigma_J, J) : \mathsf{Verify}(\sigma_x, s, t, x, y, r) = 1 \ \wedge \ \mathsf{Verify}(\sigma_J, J) = 1 \ \wedge \ S = \mathsf{Vrf}(s, J) \ \wedge \ T = u^x \mathrm{Vrf}(t, J)^R \ \wedge \ C = \mathrm{Commit}(s, t, x, y, r, J)\Big\}(R)$$

If Π_{Spend} is a valid SPK, the merchant accepts the payment. Finally, the user increases the counter J of his wallet by 1. When J is bigger than k, the user can no longer spend his wallet unlinkably.

Compact Spend. To spend the whole wallet, the user then sends to the merchant C which is the commitment of (s, t, x, y, r) and also $T_c = PK\mathsf{Vrf}(y, 0)^R$. Then, it sends the following signature of knowledge to the merchant.

$$\Pi_{\mathsf{CSpend}} : SPK\Big\{(\sigma_x, s, t, x, y, r) : \mathsf{Verify}(\sigma_x, s, t, x, y, r) = 1 \ \wedge \ T_c = u^x \mathsf{Vrf}(y, 0)^R \ \wedge \ C = \mathrm{Commit}(s, t, x, y, r)\Big\}(R)$$

Finally, the user discloses s, t to the merchant. If Π_{CSpend} is valid and s, t is indeed the value in the commitment, the merchant accepts the whole payment.

Batch Spend. To spend n coins together, the user then sends to the merchant C which is the commitment of (s, t, x, y, r, J) and also $S_i = \mathsf{Vrf}(s, J + i)$, $T_i = PK\mathsf{Vrf}(t, J + i)^R$ for $i = 0, \cdots, n - 1$. Then, it sends the following signature of knowledge to the merchant.

$$\begin{aligned}\Pi_{\mathsf{BSpend}} : SPK\Big\{(\sigma_x, s, t, x, y, r, \sigma_J, J, \sigma_{J+n-1}) : \\ \mathsf{Verify}(\sigma_x, s, t, x, y, r) = 1 \ \wedge \ \mathsf{Verify}(\sigma_J, J) = 1 \ \wedge \ S_0 = \mathsf{Vrf}(s, J) \ \wedge \ T_0 = u^x \mathrm{Vrf}(t, J)^R \ \wedge \\ \cdots \ \wedge \ S_i = \mathsf{Vrf}(s, J + i) \ \wedge \ T_i = u^x \mathrm{Vrf}(t, J + i)^R \ \wedge \ \cdots \ \wedge \\ S_{n-1} = \mathsf{Vrf}(s, J + n - 1) \ \wedge \ T_{n-1} = u^x \mathrm{Vrf}(t, J + n - 1)^R \ \wedge \\ \mathsf{Verify}(\sigma_{J+n-1}, J + n - 1) = 1 \ \wedge \ C = \mathrm{Commit}(s, t, x, y, r, J)\Big\}(R)\end{aligned}$$

If Π_{BSpend} is a valid SPK, the merchant accepts the payment. Finally, the user increases the counter J of his wallet by n.

Remarks: S is called a serial number. For each wallet, only k valid serial numbers can be generated. Should a user attempt to double-spend, he must use a duplicated serial number. On the other hand, during CSpend, the user submits s to the merchant and this is equivalent to submitting all k possible serial numbers. This is the main technique we used to achieve compact spending. Once double-spending is identified, T is the component used to revoke identity of double-spender, as shown in the RevokeDoubleSpender algorithm.

We achieve *constant-size compact e-cash*, due to the idea from [22], by having the bank publishes k signatures on 1 to k. User proving possession of these signatures on counter j indirectly proves counter j has not reached the limit k. Proving j is within 1 to k directly require a complexity of $O(logk)$ while with this technique, constant-size is achieved. The price is that public parameter size is increased to k. Note that if the bank is dishonest and gives signature on $k + 1$ to a user, the user is able to spend the wallet for $k+1$ times without being noticed. However, this does not compromise the security since this only breaks the *balance* property which is exactly against the interest of the bank. Thus, it gives no incentive for the bank to behave dishonestly in this way.

Deposit. To deposit, the merchant simply gives the bank the whole communication transcript during the spend protocol. The bank verifies the transcript exactly as the merchant did. In addition, the bank has to verify that I is indeed the identity of the merchant and $R = H(\mathrm{info}, I)$ is not used before by that merchant. This is to prevent colluding users and merchants from submitting a double spent coin (which have identical transcripts). It also prevents a malicious merchant from eavesdropping an honest transaction and depositing it (in that case, identity of the malicious merchant does not match with I). In case the check is successful, the bank stores S, T, R to the database. In case it is CSpend,

the bank computes $S_i = \mathsf{Vrf}(s, i)$ for $i = 1, \cdots, k$. The bank then stores all (S, T, R) $\big((S, T_c, s, t, R)$ in case it is CSpend$\big)$ for each spending in the database.

RevokeDoubleSpender. When a new spending transcript is received, the bank checks if S exists in the database. If yes, then it is a double-spent coin. The bank identifies the double-spender as follows. There are three cases:

- (Double-spending of a single coin.) Let the entry in the database be (S, T', R') and the current transcript be (S, T, R). The bank computes PK as $(\frac{T^{R'}}{T'^{R}})^{1/(R'-R)}$.
- (CSpend and spend a single coin.) Suppose the entry in database is (S, T_c, s, t) and the current transcript is (S, T, R). The bank checks for an i such that $S = \mathsf{Vrf}(s, i)$ and computes $PK = T/(\mathsf{Vrf}(t, i)^R)$.
- (Double CSpend.) Suppose the two entries are (s, t, T_c, R) and (s, t, T_c', R'). The bank computes $PK = (\frac{T_c^{R'}}{T_c'^{R}})^{1/(R'-R)}$.

Remarks: Double spending can be falsely identified if there exists $J, J' \leq k$ such that $J + s = J' + s'$ for two different wallets. However, the probability is negligible if k is much smaller than the security parameter. This applies to the CHL scheme too. The proof π_D such that bank is honest is the two double-spend transcripts.

VerifyGuilt. The bank outputs the double-spent transcripts as well as the public key of the double-spender. Everyone can check if the bank is honest by invoking the algorithm RevokeDoubleSpender on the two transcripts since it does not require any of the bank's secret.

4.2 Scheme 1 (Instantiation Using BBS+ Signature and DY VRF)

Following the generic construction, efficient compact e-cash can be constructed readily by choosing a suitable signature scheme with efficient protocols and VRF. One additional criterion is that $PK\{(t, x, j) : T = u^x \mathrm{Vrf}(t, x, j)^R\}$ can be efficiently done since that may not be efficient for *any* VRF. Below we instantiate a q-SDH based compact e-cash using BBS+ signature and DY VRF.

BankSetup. Let λ be the security parameter. Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with computable isomorphism ψ as discussed such that $|\mathbb{G}_1| = |\mathbb{G}_2| = p$ for some prime p of λ bits. Also assume $\mathbb{G}_p$ is a group of order p where DDH is intractable. Let $H : \{0,1\}^* \to \mathbb{Z}_p$ be a cryptographic hash function. Let $g_0, g_1, g_2, g_3, g_4, g_5$ be generators of $\mathbb{G}_1$, $h_0, h_1, h_2, h_3, h_4, h_5$ be generators of group $\mathbb{G}_2$ such that $\psi(h_i) = g_i$ and u_0, u_1, u_2, u_3 be generators of $\mathbb{G}_p$ such that related discrete logarithm of the generators are unknown. This can be done by setting the generators to be output by a hash function of some publicly known seed. The bank randomly selects $\gamma, \gamma_r \in_R \mathbb{Z}_p^*$ and computes $w = {h_0}^{\gamma}, w_r = {h_0}^{\gamma_r}$. The bank's public key is $bpk = (g_0, g_1, g_2, g_3, g_4, g_5, h_0, w, w_r, u_0, u_1, u_2, u_3, k)$ and the bank's secret key is $bsk = (\gamma, \gamma_r)$. It also publishes $\sigma_i = (B_i, d_i)$ s.t. $\hat{e}(B_i, w_r h_0^{d_i}) = \hat{e}(g_0, h_0)\hat{e}(g_1, h_0)^i$ for $i = 1, \cdots, k$. These are the BBS+ signature on i for $i = 1, \cdots, k$. k has to be much smaller than 2^λ. For efficiency consideration, it also publishes $E_j = \hat{e}(g_j, h_0)$ for $j = 0, \cdots, 5$ and $E_w = \hat{e}(g_2, w)$, $E_{wr} = \hat{e}(g_2, w_r)$ as part of the public key.

UserSetup. We assume PKI is implemented, that is, each user is equipped with a discrete logarithm type public and private key pair $({u_0}^x, x) \in \mathbb{G}_p \times \mathbb{Z}_p^*$.

WithdrawalProtocol. A user randomly selects $s', t, y, r \in_R \mathbb{Z}_p^*$ and sends $C' = g_1^{s'} g_2^t g_3^x g_4^y g_5^r$, along with the proof $\Pi_0 = PK\{(s', t, x, y, r) : C' = g_1^{s'} g_2^t g_3^x g_4^y g_5^r \wedge PK = u_0^x\}$ to the bank. The bank verifies that Π_0 is valid and randomly selects $s'' \in_R \mathbb{Z}_p^*$. It computes $C = C' g_1^{s''}$ and selects $e \in_R \mathbb{Z}_p^*$. It then computes $A = (g_0 C)^{\frac{1}{e+\gamma}}$ and sends (A, e, s'') to the user. User computes $s = s' + s''$ and checks if

$\hat{e}(A, wh_0^e) = \hat{e}(g_0 g_1^s g_2^t g_3^x g_4^y g_5^r, h_0)$. He then stores (A, e, s, t, x, y, r) as his wallet secret and sets counter $J = 1$.

Spend Protocols. Let the user wallet be (A, e, s, t, x, y, r, J) such that $J \leq k$. The merchant with identity I and the user first agree on the transaction information info and compute $R = H(\text{info}, I)$ locally.

Single Coin Spend Protocol. The user computes $S = u_1^{\frac{1}{s+J+1}}$, $T = u_0^x u_1^{\frac{R}{t+J+1}}$. The user also computes the following quantities $A_1 = g_1^{r_1} g_2^{r_2}$, $A_2 = A g_2^{r_1}$, $A_3 = g_1^J g_2^t g_3^{r_3}$, $A_4 = g_1^{r_4} g_2^{r_5}$, $A_5 = B_J g_2^{r_4}$, for $r_1, r_2, r_3, r_4, r_5 \in_R \mathbb{Z}_p^*$, in $\mathbb{G}_1$. Recall that (B_J, d_J) is the BBS+ signature on J published by the bank. The following SPK Π_1 is then computed.

$$\Pi_1 : SPK\Big\{(r_1, r_2, r_3, r_4, r_5, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_J, \delta_t, e, d_J, s, t, x, y, r, J) :$$
$$A_1 = g_1^{r_1} g_2^{r_2} \ \wedge \ A_1^e = g_1^{\delta_1} g_2^{\delta_2} \ \wedge \ \tfrac{\hat{e}(A_2, w)}{E_0} = E_1^s E_2^t E_3^x E_4^y E_5^r E_2^{\delta_1} E_w^{r_1} \hat{e}(A_2, h_0)^{-e} \ \wedge$$
$$\tfrac{u_1}{S} = S^J S^s \ \wedge \ A_3 = g_1^J g_2^t g_3^{r_3} \ \wedge \ A_3^x = g_1^{\delta_J} g_2^{\delta_t} g_3^{\delta_3} \ \wedge \ \tfrac{u_1^R}{T} = T^J T^t u_0^{-\delta_J} u_0^{-\delta_t} u_0^{-x} \ \wedge$$
$$A_4 = g_1^{r_4} g_2^{r_5} \ \wedge \ A_4^{d_J} = g_1^{\delta_4} g_2^{\delta_5} \ \wedge \ \tfrac{\hat{e}(A_5, w_r)}{E_0} = E_1^J E_2^{\delta_4} E_{wr}^{r_4} \hat{e}(A_5, h_0)^{-d_J}\Big\}(R)$$

where $\delta_1 = r_1 e, \delta_2 = r_2 e, \delta_4 = r_4 d_J, \delta_5 = r_5 d_J, \delta_J = Jx, \delta_t = tx, \delta_3 = r_3 x$.

The user sends $S, T, A_1, A_2, A_3, A_4, A_5$ along with Π_1 to the merchant for payment. The merchant then verifies Π_1 and accepts the payment if it is valid.

CSpend Protocol. To spend the whole wallet, the user computes $T_c = u_0^x u_1^{\frac{R}{y+1}}$. He also computes the following quantities $A_1 = g_1^{r_1} g_2^{r_2}$, $A_2 = A g_2^{r_1}$, $A_3 = g_1^y g_2^{r_3}$ for $r_1, r_2, r_3 \in_R \mathbb{Z}_p^*$, in $\mathbb{G}_1$. The following SPK Π_2 is then computed.

$$\Pi_2 : SPK\Big\{(r_1, r_2, r_3, \delta_1, \delta_2, \delta_3, \delta_y, e, x, y, r) :$$
$$A_1 = g_1^{r_1} g_2^{r_2} \ \wedge \ A_1^e = g_1^{\delta_1} g_2^{\delta_2} \ \wedge \ \tfrac{\hat{e}(A_2, w)}{E_0 E_1^s E_2^t} = E_3^x E_4^y E_5^r E_2^{\delta_1} E_w^{r_1} \hat{e}(A_2, h_0)^{-e} \ \wedge$$
$$A_3 = g_1^y g_2^{r_3} \ \wedge \ A_3^x = g_1^{\delta_y} g_2^{\delta_3} \ \wedge \ \tfrac{u_1^R}{T_c} = T_c^y u_0^{-\delta_y} u_0^{-x}\Big\}(R)$$

where $\delta_1 = r_1 e, \delta_2 = r_2 e, \delta_y = yx, \delta_3 = r_3 x$.

The user sends T_c, s, t, A_1, A_2, A_3, along with Π_2 to the merchant for payment. The merchant then verifies Π_2 and accepts the payment if it is valid.

BSpend Protocol. To spend n coins such that $J + n - 1 \leq k$, the user computes $S_i = u_1^{\frac{1}{s+J+i}}$, $T_i = u_0^x u_1^{\frac{R}{t+J+i}}$ for $i = 1$ to n. Denotes $I = J + n - 1$. The user also computes the following quantities $A_1 = g_1^{r_1} g_2^{r_2}$, $A_2 = A g_2^{r_1}$, $A_3 = g_1^J g_2^t g_3^{r_3}$, $A_4 = g_1^{r_4} g_2^{r_5}$, $A_5 = B_J g_2^{r_4}$, $A_6 = g_1^{r_6} g_2^{r_7}$, $A_7 = B_I g_2^{r_6}$, for $r_1, r_2, r_3, r_4, r_5, r_6, r_7 \in_R \mathbb{Z}_p^*$, in $\mathbb{G}_1$. Recall that $(B_J, d_J), (B_I, d_I)$ are the BBS+ signatures published by the bank on J and I respectively. The following SPK Π_3 is then computed.

$$\Pi_3 : SPK\{(r_1, r_2, r_3, r_4, r_5, r_6, r_7, \delta_1, \delta_2, \delta_3, \delta_4, \delta_5, \delta_6, \delta_7, \delta_J, \delta_t, e, d_J, d_I, s, t, x, y, r, J) :$$
$$A_1 = g_1^{r_1} g_2^{r_2} \wedge A_1^e = g_1^{\delta_1} g_2^{\delta_2} \ \wedge \ \tfrac{\hat{e}(A_2, w)}{E_0} = E_1^s E_2^t E_3^x E_4^y E_5^r E_2^{\delta_1} E_w^{r_1} \hat{e}(A_2, h_0)^{-e} \ \wedge$$
$$\tfrac{u_1}{S_1^1} = S_1^J S_1^s \ \wedge \ \cdots \ \wedge \ \tfrac{u_1}{S_i^i} = S_i^J S_i^s \ \wedge \ \cdots \ \wedge \ \tfrac{u_1}{S_n^n} = S_n^J S_n^s \ \wedge$$
$$A_3 = g_1^J g_2^t g_3^{r_3} \ \wedge \ A_3^x = g_1^{\delta_J} g_2^{\delta_t} g_3^{\delta_3} \ \wedge \ \tfrac{u_1^R}{T_1^1} = T_1^J T_1^t u_0^{-\delta_J} u_0^{-\delta_t} (u_0^1)^{-x} \ \wedge \ \cdots \ \wedge$$
$$\tfrac{u_1^R}{T_i^i} = T_i^J T_i^t u_0^{-\delta_J} u_0^{-\delta_t} (u_0^i)^{-x} \ \wedge \ \cdots \ \wedge \ \tfrac{u_1^R}{T_n^n} = T_n^J T_n^t u_0^{-\delta_J} u_0^{-\delta_t} (u_0^n)^{-x} \ \wedge$$
$$A_4 = g_1^{r_4} g_2^{r_5} \ \wedge \ A_4^{d_J} = g_1^{\delta_4} g_2^{\delta_5} \ \wedge \ \tfrac{\hat{e}(A_5, w_r)}{E_0} = E_1^J E_2^{\delta_4} E_{wr}^{r_4} \hat{e}(A_5, h_0)^{-d_J} \ \wedge$$
$$A_6 = g_1^{r_6} g_2^{r_7} \ \wedge \ A_6^{d_I} = g_1^{\delta_6} g_2^{\delta_7} \ \wedge \ \tfrac{\hat{e}(A_7, w_r)}{E_0 E_1^{n-1}} = E_1^J E_2^{\delta_6} E_{wr}^{r_6} \hat{e}(A_7, h_0)^{-d_I}\Big\}(R)$$

where $\delta_1 = r_1 e, \delta_2 = r_2 e, \delta_4 = r_4 d_J, \delta_5 = r_5 d_J, \delta_6 = r_5 d_I, \delta_7 = r_7 d_I, \delta_J = Jx, \delta_t = tx, \delta_3 = r_3 x$.

The user sends $S_1, T_1, \cdots, S_n, T_n, A_1, A_2, A_3, A_4, A_5, A_6, A_7$ along with Π_3 to the merchant for payment. The merchant then verifies Π_3 and accepts the payment if it is valid.

Deposit, RevokeDoubleSpender and VerifyGuilt have been described in the generic construction.

4.3 Scheme 2 (Instantiation Using CL+ Signature and DY VRF)

BankSetup. Let λ be the security parameter. Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group pair with computable isomorphism ψ as discussed such that $|\mathbb{G}_1| = |\mathbb{G}_2| = p$ for some prime p of λ bits. Also assume $\mathbb{G}_p$ is a group of order p where DDH is intractable. Let $H : \{0,1\}^* \rightarrow \mathbb{Z}_p$ be a cryptographic hash function. Let h_0 be a generator of $\mathbb{G}_2$ and $g_0 = \psi(h_0)$. Let also g_1, g_2, g_3 be generators of $\mathbb{G}_1$ Let u_0, u_1 be generator of $\mathbb{G}_p$. The bank chooses $\alpha, \beta, \gamma_1, \cdots, \gamma_4 \in_R \mathbb{G}_p$. The bank then computes $X = h_0^{\alpha}, Y = h_0^{\beta}, Z_1 = h_0^{\gamma_1}, \cdots, Z_4 = h_0^{\gamma_4}, W_1 = Y^{\gamma_1}, \cdots, W_4 = Y^{\gamma_4}$. The secret key of the bank is $(\alpha, \beta, \gamma_1, \cdots, \gamma_4)$ and the public key is $(h_0, g_0, g_1, g_2, g_3, u_0, u_1, X, Y, Z_1, \cdots, Z_4, W_1, \cdots, W_4)$. Also, the bank computes $X_r = h_0^{\alpha_r}, Y_r = h_0^{\beta_r}$ for some randomly selected $\alpha_r, \beta_r \in_R \mathbb{Z}_p$. The bank then publishes CL+ signatures $\sigma_i = (a_i, b_i, c_i)$ on $1 \cdots, k$ such that $(a_i, b_i, c_i) = (a_i, a_i^{\beta_r}, a_i^{\alpha_r + k\alpha_r\beta_r})$ for some randomly generated a_i.

UserSetup. A user is equipped with public/private key pair (u_0^x, x).

WithdrawalProtocol. A user with key pair (u_0^x, x) randomly chooses $s, t, y, r \in_R \mathbb{Z}_p$ and computes $C = h_0^s Z_1^t Z_2^x Z_3^y Z_4^r \in \mathbb{G}_2$ and sends it to the bank, along with proof $\Pi_{0'} = PK\{(s,t,x,y,r) : C = h_0^s Z_1^t Z_2^x Z_3^y Z_4^r \wedge PK = u_0^x\}$. The banks verifies $\Pi_{0'}$ is valid and randomly chooses $d \in_R \mathbb{G}_p$. It computes $a = h_0^d$, $A_1 = a^{\gamma_1}, \cdots, A_4 = a^{\gamma_4}$. It then computes $b = a^{\beta}$ and $B_1 = A_1^{\beta}, \cdots, B_4 = A_4^{\beta}$. Finally, it computes $c = a^{\alpha} C^{d\alpha\beta}$. The bank then sends $(a, b, c, A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4)$ to the user as his wallet. The user then checks if $\hat{e}(\psi(a), Z_i) = \hat{e}(g, A_i)$, $\hat{e}(\psi(a), Y) = \hat{e}(g, b)$, $\hat{e}(\psi(A_i), Y) = \hat{e}(g, B_i)$ for $i = 1, \cdots, 4$ and $\hat{e}(g, c) = \hat{e}(\psi(X), ab^s B_1^t B_2^x B_3^y B_4^r)$.

Spend Protocols. Let the user wallet be $(a,b,c,A_1,A_2,A_3,A_4,B_1,B_2,B_3,B_4,s,t,x,y,r,J)$ such that $J \leq k$. The merchant with identity I and the user first agree on the transaction information info and computes $R = H(\text{info}, I)$ locally.

Single Coin Spend Protocol. The user computes $S = u_1^{\frac{1}{s+J+1}}$, $T = u_0^x u_1^{\frac{R}{t+J+1}}$. The user chooses $r_1, r_2, r_3, r_4, r_5 \in_R \mathbb{Z}_p$ and computes the following quantities $A_T = g_1^J g_2^t g_3^{r_5}$, $\tilde{a} = a^{r_1}$, $\tilde{b} = b^{r_1}$, $\tilde{c} = c^{r_1}$, $\tilde{A}_i = A_i^{r_1}$, $\tilde{B}_i = B_i^{r_1}$ for $i = 1, \cdots, 4$ and computes $\hat{c} = \tilde{c}^{r_2}$. Recall that (a_J, b_J, c_J) is the bank's CL+ signature on J. The user also computes $(\tilde{a_J}, \tilde{b_J}, \hat{c_J}) = (a_J^{r_3}, b_J^{r_3}, c_J^{r_3 r_4})$. The following SPK $\Pi_{1'}$ is then computed. For simplicity we denote $E_{\hat{c}} = \hat{e}(g_0, \hat{c})$, $E_{\tilde{a}} = \hat{e}(\psi(X), \tilde{a})$, $E_{\tilde{b}} = \hat{e}(\psi(X), \tilde{b})$, $E_{\tilde{B}_i} = \hat{e}(\psi(X), \tilde{B}_i)$ for $i = 1, \cdots, 4$ and also $E_{\tilde{c_J}} = \hat{e}(g_0, \hat{c_J})$, $E_{\tilde{a_J}} = \hat{e}(\psi(X_r), \tilde{a_J})$, $E_{\tilde{b_J}} = \hat{e}(\psi(X_r), \tilde{b_J})$.

$$\Pi_{1'} : SPK\Big\{(s,t,x,y,r,\delta_2,J,\delta_4,r_5,\delta_J,\delta_t,\delta_5) :$$

$$E_{\tilde{a}} = E_{\hat{c}}^{\delta_2} E_{\tilde{b}}^{-s} E_{\tilde{B}_1}^{-t} E_{\tilde{B}_2}^{-x} E_{\tilde{B}_3}^{-y} E_{\tilde{B}_4}^{-r} \ \wedge \ E_{\tilde{a_J}} = E_{\tilde{c_J}}^{\delta_4} E_{\tilde{b_J}}^{-J} \ \wedge \ \tfrac{u_1}{S} = S^J S^s \ \wedge$$

$$A_T = g_1^J g_2^t g_3^{r_5} \ \wedge \ A_T^x = g_1^{\delta_J} g_2^{\delta_t} g_3^{\delta_5} \ \wedge \ \tfrac{u_1^R}{T} = T^J T^t u_0^{-\delta_J} u_0^{-\delta_t} u_0^{-x}\Big\}(R)$$

where $\delta_2 = 1/r_2 \bmod p, \delta_4 = 1/r_4 \bmod p, \delta_J = Jx, \delta_t = tx, \delta_5 = r_5 x$.

The user then sends $S, T, A_T, \tilde{a}, \tilde{b}, \hat{c}, \tilde{A}_1, \tilde{B}_1, \cdots, \tilde{A}_4, \tilde{B}_4, \tilde{a_J}, \tilde{b_J}, \hat{c_J}$, along with $\Pi_{1'}$ to the merchant. The merchant verifies that $\hat{e}(\psi(\tilde{a}), Z_i) = \hat{e}(g_0, \tilde{A}_i)$, $\hat{e}(\psi(\tilde{A}_i), Y) = \hat{e}(g_0, \tilde{B}_i)$ for $i = 1, \cdots, 4$ and $\hat{e}(\psi(\tilde{a}), Y) = \hat{e}(g_0, \tilde{b})$, $\hat{e}(\psi(\tilde{a_J}), Y_r) = \hat{e}(g_0, \tilde{b_J})$. Then the merchant verifies that $\Pi_{1'}$. It accepts the payment if all are valid.

CSpend Protocol. To spend the whole wallet, the user computes $T_c = u_0^x u_1^{\frac{R}{y+1}}$. The user chooses $r_1, r_2, r_3 \in_R \mathbb{Z}_p$ and computes the following quantities $A_T = g_1^y g_2^{r_3}$, $\tilde{a} = a^{r_1}$, $\tilde{b} = b^{r_1}$, $\tilde{c} = c^{r_1}$, $\tilde{A}_i = A_i^{r_1}$, $\tilde{B}_i = B_i^{r_1}$ for $i = 1, \cdots, 4$ and computes $\hat{c} = \tilde{c}^{r_2}$. The following SPK $\Pi_{2'}$ is then computed, using the same notations we use above.

$$\Pi_{2'} : SPK\Big\{(x, y, r, \delta_2, r_3, \delta_y, \delta_3) : \frac{E_{\tilde{a}}}{E_{\tilde{b}}^s E_{\tilde{B}_1}^t} = E_{\hat{c}}^{\delta_2} E_{\tilde{B}_2}^{-x} E_{\tilde{B}_3}^{-y} E_{\tilde{B}_4}^{-r} \ \wedge \ A_T = g_1^y g_2^{r_3} \ \wedge \ A_T^x = g_1^{\delta_y} g_2^{\delta_3} \ \wedge \ \frac{u_1^R}{T_c} = T_c{}^y u_0^{-\delta_y} u_0^{-x}\Big\}(R)$$

where $\delta_2 = 1/r_2 \bmod p, \delta_y = yx, \delta_3 = r_3 x$.

The user then sends $s, t, T_c, A_T, \tilde{a}, \tilde{b}, \hat{c}, \tilde{A}_1, \tilde{B}_1, \cdots, \tilde{A}_4, \tilde{B}_4$, along with $\Pi_{2'}$ to the merchant. The merchant verifies that $\hat{e}(\psi(\tilde{a}), Z_i) = \hat{e}(g_0, \tilde{A}_i)$, $\hat{e}(\psi(\tilde{A}_i), Y) = \hat{e}(g_0, \tilde{B}_i)$ for $i = 1, \cdots, 4$ and $\hat{e}(\psi(\tilde{a}), Y) = \hat{e}(g_0, \tilde{b})$. Then the merchant verifies that $\Pi_{2'}$. It accepts the payment if all are valid.

BSpend Protocol. To spend n coins such that $J + n - 1 \leq k$, the user computes $S_i = u_1^{\frac{1}{s+J+i}}$, $T_i = u_0^x u_1^{\frac{R}{t+J+i}}$ for $i = 1$ to n. Denotes $I = J + n - 1$. The user chooses $r_1, r_2, r_3, r_4, r_5, r_6, r_7 \in_R \mathbb{Z}_p$ and computes the following quantities $A_T = g_1^J g_2^t g_3^{r_7}$, $\tilde{a} = a^{r_1}$, $\tilde{b} = b^{r_1}$, $\tilde{c} = c^{r_1}$, $\tilde{A}_i = A_i^{r_1}$, $\tilde{B}_i = B_i^{r_1}$ for $i = 1, \cdots, 4$ and computes $\hat{c} = \tilde{c}^{r_2}$. Recall that $(a_J, b_J, c_J), (a_I, b_I, c_I)$ are the bank's CL+ signatures on J and I respectively. The user also computes $(\tilde{a_J}, \tilde{b_J}, \tilde{c_J}) = (a_J^{r_3}, b_J^{r_3}, c_J^{r_3 r_4})$ and $(\tilde{a_I}, \tilde{b_I}, \tilde{c_I}) = (a_I^{r_5}, b_J^{r_5}, c_J^{r_5 r_6})$. The following SPK $\Pi_{3'}$ is then computed, using the same notations we use above.

$$\Pi_{3'} : SPK\Big\{(s, t, x, y, r, \delta_2, J, \delta_4, \delta_6, r_7, \delta_J, \delta_t, \delta_7) :$$
$$E_{\tilde{a}} = E_{\hat{c}}^{\delta_2} E_{\tilde{b}}^{-s} E_{\tilde{B}_1}^{-t} E_{\tilde{B}_2}^{-x} E_{\tilde{B}_3}^{-y} E_{\tilde{B}_4}^{-r} \ \wedge \ E_{\tilde{a_J}} = E_{\tilde{c_J}}^{\delta_4} E_{\tilde{b_J}}^{-J} \ \wedge \ \frac{E_{\tilde{a_I}}}{E_{\tilde{b_I}}^{n-1}} = E_{\tilde{c_I}}^{\delta_6} E_{\tilde{b_I}}^{-J} \ \wedge$$
$$\frac{u_1}{S_1^1} = S_1^J S_1^s \ \wedge \ \cdots \ \wedge \ \frac{u_1}{S_i^i} = S_i^J S_i^s \ \wedge \ \cdots \ \wedge \ \frac{u_1}{S_n^n} = S_n^J S_n^s \ \wedge$$
$$A_T = g_1^J g_2^t g_3^{r_5} \ \wedge \ A_T^x = g_1^{\delta_J} g_2^{\delta_t} g_3^{\delta_5} \ \wedge \ \frac{u_1^R}{T_1^1} = T_1^J T_1^t u_0^{-\delta_J} u_0^{-\delta_t} (u_0^1)^{-x} \ \wedge \ \cdots \ \wedge$$
$$\frac{u_1^R}{T_i^i} = T_i^J T_i^t u_0^{-\delta_J} u_0^{-\delta_t} (u_0^i)^{-x} \ \wedge \ \cdots \ \wedge \ \frac{u_1^R}{T_n^n} = T_n^J T_n^t u_0^{-\delta_J} u_0^{-\delta_t} (u_0^n)^{-x}\Big\}(R)$$

where $\delta_2 = 1/r_2 \bmod p, \delta_4 = 1/r_4 \bmod p, \delta_6 = 1/r_6, \delta_J = Jx, \delta_t = tx, \delta_7 = r_7 x$.

The user then sends $A_T, S_1, T_1, \cdots, S_n, T_n, \tilde{a}, \tilde{b}, \hat{c}, \tilde{A}_1, \tilde{B}_1, \cdots, \tilde{A}_4, \tilde{B}_4, \ \tilde{a_J}, \tilde{b_J}, \tilde{c_J}, \ \tilde{a_I}, \tilde{b_I}, \tilde{c_I}$, along with $\Pi_{3'}$ to the merchant. The merchant verifies that $\hat{e}(\psi(\tilde{a}), Z_i) = \hat{e}(g_0, \tilde{A}_i)$, $\hat{e}(\psi(\tilde{A}_i), Y) = \hat{e}(g_0, \tilde{B}_i)$ for $i = 1, \cdots, 4$ and $\hat{e}(\psi(\tilde{a}), Y) = \hat{e}(g_0, \tilde{b})$, $\hat{e}(\psi(\tilde{a_J}), Y_r) = \hat{e}(g_0, b_J)$, $\hat{e}(\psi(\tilde{a_I}), Y_r) = \hat{e}(g_0, b_I)$. Then the merchant verifies that $\Pi_{1'}$. It accepts the payment if all are valid.

Deposit, RevokeDoubleSpender and VerifyGuilt have been described in the generic construction.

Following the parameters suggested by Boneh *et al.*[5, 4], we can take $p = 170$ bits and each group element in $\mathbb{G}_T$, $\mathbb{G}_2$ can be represented by 171 bits. Assume elements in $\mathbb{G}_p$ are represented by 171 bits (using another elliptic curve group where pairing is not available[16]). We list the time and space complexity of our schemes and the CHL scheme in Fig. 1. For the CHL scheme, we take the public modulus N to be 1024 bits.

4.4 Extensions

Our schemes can be extended to support full coin tracing using the same method as in [8]. It can also be extended to support arbitrary wallet size. Due to space limitation, these extensions are shown in appendix A.

4.5 Security Analysis

Proofs of the following theorems can be found in appendix B.

Theorem 1. *Our first scheme is secure under the q-SDH assumption and the k-DDHI assumption in the random oracle model.*

Theorem 2. *Our second scheme is secure under the LRSW assumption and the k-DDHI assumption in the random oracle model.*

	CHL	this paper(scheme 1)	this paper (scheme 2)
Withdrawal	704 bytes	213 bytes	384 bytes
Single Spend	1.9 kB	596 bytes	640 bytes
Batch Spend ($n > 1$ coins)	N/A	$702 + 43n$ bytes	$682 + 43n$ bytes
Compact Spend (k coins)	N/A	383 bytes	491 bytes
Deposit	Same as respective Spend protocols		
Bank's Store (per spent coin)	0.3 kB	64 bytes	64 bytes

Fig. 1. Space Efficiency of different protocols.

	CHL	this paper (scheme 1)	this paper (scheme 2)
Single Spend			
User	18ME	17ME + 2P	24ME +8P
Merchant	11ME	10ME + 4P	6ME +20P
Bank	11ME	10ME + 4P	6ME +20P
Batch Spend ($n > 1$ coins)			
User	N/A	$(4n + 18)$ME + 2P	$(4n + 11)$ME + 10P
Merchant	N/A	$(2n + 11)$ME + 6P	$(2n + 5)$ME + 25P
Bank	N/A	$(2n + 11)$ME + 6P	$(2n + 5)$ME + 25P
Compact Spend			
User	N/A	10ME + 1P	17ME + 4P
Merchant	N/A	6ME + 2P	4ME + 13P
Bank	N/A	6ME + 2P	4ME + 13P

Fig. 2. Computational Cost of Spend protocols. (ME=Multi-based Exponentiation, P=Pairing)

5 Concluding Remarks

We introduced the idea of compact spending and batch spending into compact e-cash, presented security model to accommodate the new idea, and gave efficient and secure constructions. One problem of our system is that since BBS+/CL+ (or CL) signatures do not support concurrent signature generation, withdrawal must be done in a sequential manner. The same drawback is also present in the original compact e-cash [8]. It remains an open problem to design a secure compact e-cash scheme which supports concurrent withdrawal.

Acknowledgments.

We would like to thank Colin Boyd and the anonymous reviewers of ACISP 2007 for their helpful comments and suggestions.

References

1. M. H. Au, W. Susilo, and Y. Mu. Constant-Size Dynamic k-TAA. In *SCN 2006*, volume 4116 of *LNCS*, pages 111–125. Springer-Verlag, 2006.
2. M. H. Au, Q. Wu, W. Susilo, and Y. Mu. Compact e-cash from bounded accumulator. In *CT-RSA*, 2007.
3. D. Boneh and X. Boyen. Short signatures without random oracles. In *EUROCRYPT*, pages 56–73, 2004.
4. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.
5. D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *ASIACRYPT*, volume 2248, pages 514–532, 2001.
6. F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, pages 431–444, 2000.
7. E. Brickell, P. Gemmell, and D. Kravitz. Trustee-based Tracing Extensions to Anonymous Cash and the Making of Anonymous Change. In *SODA '95: Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 457–466. Society for Industrial and Applied Mathematics, 1995.
8. J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Compact e-cash. In *EUROCRYPT*, pages 302–321, 2005.
9. J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. In *SCN 2002*, volume 2576 of *LNCS*, pages 268–289. Springer-Verlag, 2002.

10. J. Camenisch and A. Lysyanskaya. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO*, volume 3152, pages 56–72, 2004.
11. S. Canard and J. Traoré. On fair e-cash systems based on group signature schemes. In *ACISP*, pages 237–248, 2003.
12. D. Chaum. Blind Signatures for Untraceable Payments. In *Advances in Cryptology: Proceedings of CRYPTO '82*, pages 199–203. Plenum, New York, 1983.
13. D. Chaum. Security without identification: Transaction systems to make big brother obsolete. In *Communications of the ACM, 28(10)*, pages 1030–1044, 1985.
14. J. H. Cheon. Security analysis of the strong Diffie-Hellman problem. In *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2006.
15. Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *PKC 2005*, volume 3386 of *LNCS*, pages 416 – 431, 2005.
16. J. Furukawa and H. Imai. An efficient group signature scheme from bilinear maps. In *ACISP*, pages 455–467, 2005.
17. A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Selected Areas in Cryptography*, pages 184–199, 1999.
18. G. Maitland and C. Boyd. Fair Electronic Cash Based on a Group Signature Scheme. In *Information and Communications Security, Third International Conference, ICICS 2001, Xian, China, November 13-16, 2001*, volume 2229 of *Lecture Notes in Computer Science*, pages 461–465. Springer, 2001.
19. S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.
20. T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.
21. I. Teranishi, J. Furukawa, and K. Sako. k-Times Anonymous Authentication (Extended Abstract). In *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 308–322. Springer-Verlag, 2004.
22. I. Teranishi and K. Sako. *k*-times anonymous authentication with a constant proving cost. In *Public Key Cryptography*, pages 525–542, 2006.

A Extensions

A.1 Extending Our System to Support Full Tracing.

Our system extracts the public key u_0^x of the double-spender. Using the same method as in [8], we can turn our scheme to support full coin tracing. That is, all coins from the double-spender can be traced, regardless it has been spent honestly or not. The idea is to make use of bilinear encryption, where secret key is of the form u_0^x. During withdrawal protocol, user is required to verifiably encrypt all the serial numbers for the wallet. Should a user double-spent, his secret key is revealed and the bank then decrypt and get all the serial numbers associated with the user.

One problem is that if bilinear encryption is used, the VRF we used may no longer secure since now we need to move G_p to G_1 where DDH may not hold. One solution is to make the XDH assumption such that DDH in G_1 still holds. Another solution is to make use of the sum-free VRF as in [8]. The price is that the protocols will become less efficient.

A.2 Extending Our System to Support Arbitrary Wallet Size.

We outline how to modify our system so that wallet size k is to be chosen by user arbitrarily during the withdrawal protocol while coins from wallet of different size are indistinguishable during normal spending. The idea is that during the withdrawal protocol, the value k chosen by the user is also signed by the bank. During spending, the user prove, in zero knowledge manner that, the counter j is within 1 to k where k is hidden from the merchant. However, inefficient exact range proof[6] has to be employed. Besides, compact spend of different wallet must be distinguishable. Specifically, the change to make is that the user obtain $\sigma_x = \mathsf{Sign}(s, t, x, y, r, k)$ during the withdrawal protocol while change of Π_{Spend} is shown as follow. Changes of Π_{CSpend} and Π_{BSepnd} are omitted.

$$\Pi_{\mathsf{Spend}} : SPK\Big\{(\sigma_x, s, t, x, y, r, k, J) : \mathsf{Verify}(\sigma_x, s, t, x, y, r, k, J) = 1 \ \wedge \ 1 \leq J \leq k \ \wedge \ T = u^x \mathrm{Vrf}(t, J)^R \ \wedge \ C = \mathrm{Commit}(s, t, x, y, r, k, J)\Big\}(R)$$

B Proof of Theorem 1 and 2

Proof.
Balance. Let $\mathcal{A}$ be an adversary that makes q_w withdrawal queries and q_s spend queries. We outline why the success probability of $\mathcal{A}$ is negligible under the q_w-SDH assumption (for scheme 1) or the LRSW assumption (for scheme 2), by constructing a simulator $\mathcal{S}$ acting as challenger $\mathcal{C}$.

For each withdrawal query, $\mathcal{A}$ is required to present a public key $PK = u_0^x$ and proof the knowledge of representation of C' with Π_0 (or C with $\Pi_{0'}$ in scheme 2). $\mathcal{S}$ acts as if an honest bank would, except during the proof of knowledge of C' (resp. C) where $\mathcal{S}$ runs a rewind simulation and extracts (s', t, y, x, r) (or (s, t, y, x, r) in scheme 2). Such an extraction is possible due to the soundness of the proof of knowledge. If Π_0 (resp. $\Pi_{0'}$) is conducted non-interactively, then $\mathcal{S}$ is given control over the random oracle as well. After each execution, $\mathcal{S}$ computes $w = (S_1, \cdots, S_k, s, t, x, y, r)$ where $S_i = u_1^{\frac{1}{i+s+1}}$ for $i = 1, \cdots, k$. The set of $\Gamma = \{S_{i,j} | 1 \leq i \leq q_w, 1 \leq j \leq k\}$ corresponds to all the serial numbers after q_w executaions of the withdrawal protocol.

For each payment query, $\mathcal{A}$ presents a public key PK, a transaction information info, merchant identity I and request a legal number of Spend Protocol, be it normal Spend, CSpend or BSpend for that user. Suppose it is normal Spend of BSpend, $\mathcal{S}$ responds by randomly choosing (s, t) and computing $S = u_1^{\frac{1}{s+j+1}}$, $T = PKu_1^{\frac{R}{t+j+1}}$ such that $R = H(\text{info}, I)$ and $j = i$ to $i + n - 1$ for some $1 \in_R [1, k]$ such that $i + n - 1 < k$. $\mathcal{S}$ then generates a simulated proof Π_1(resp. $\Pi_{1'}$) for Spend or Π_3(resp. $\Pi_{3'}$) for BSpend. Such simulated proof is perfect due to the HVZK property of the underlying SPK.

For CSpend, $\mathcal{S}$ randomly chooses s, t, y and computes $T_c = PKu_1^{\frac{R}{y+1}}$. $\mathcal{S}$ then gives (s, t) to $\mathcal{A}$ and computes a simulated proof Π_2 (resp.$\Pi_{2'}$). Such simulated proof is perfect due to the HVZK property of the underlying SPK.

Finally, $\mathcal{A}$ runs $kq_w + q_s + 1$ deposit protocol with $\mathcal{S}$. In case it is a CSpend, $\mathcal{S}$ uses the s to calculate the k serial numbers corresponds to that single s.

$\mathcal{A}$ wins the game either by (1) all the $kq_w + q_s + 1$ serial numbers during deposit are unique or (2) some of the serial numbers are duplicated but RevokeDoubleSpender on the corresponding deposit attempt does not point to any of the public key presented during the withdrawal queries. Now we are to analyzes these two cases separately.

Case (1): Since that only q_s serial numbers are given to $\mathcal{A}$ during the spend queries, $\mathcal{A}$ must have produce another $kq_w + 1$ serial numbers. Due to the soundness of the underlying proof of knowledge protocols, Γ contains all valid serial numbers that $\mathcal{A}$ can produce. Thus, $\mathcal{A}$ can only win in Case (1) by convincing $\mathcal{S}$ to accept a serial number $S \notin \Gamma$ and is not the output of a spend query. Then $\mathcal{A}$ must have conducted a *false* proof as part of the signature of knowledge such that one of the following is fake:

1. Possession of BBS+ (resp. CL+) signature on block of messages (s, t, x, y, r).
2. $S = u_1^{\frac{1}{j+s+1}}$.
3. Possession of BBS+ (resp. CL+) signature on j.

In case it is a CSpend, $\mathcal{A}$ have conducted a *false* proof so that one of the following is fake:

1. Possession of BBS+ (resp. CL+) signature on block of messages (s, t, x, y, r).
2. Opening of (s,t) is equal to the pair committed in the above proof.

In case it is a BSpend, one of the following is fake:

1. Possession of BBS+ (resp. CL+) signature on block of messages (s, t, x, y, r).
2. $S_i = u_1^{\frac{1}{i+s+1}}$ for $i = j$ to $j + n - 1$.
3. Possession of BBS+ (resp. CL+) signatures on j and $j + n - 1$.

Fake proof of possession of BBS+ signature (resp. CL+ signature) happens with negligible probability under the q-SDH assumption (resp. LRSW assumption). Fake proof of S (and T) are well-formed happens with negligible probability under the discrete logarithm assumption (which is subsumed by the k-DDHI assumption in the theorem statement). 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 or s are accepted. We first argue that at least one pair of duplicated S are in Γ. Due to the zero-knowledge property of the Spend Protocol, $\mathcal{A}$ learns nothing about the signature on the values (s, t, x, y, r) presented during the spend queries. Thus, $\mathcal{A}$ cannot produce valid deposit using the same set of (s, t, x, y, r) from spend query twice except using identical transcripts, which shall be rejected.

It remains to show the associated T, or (t, T_c) 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, $T = u_0^x u_1^{\frac{R}{j+t+1}}$ is the only valid T to accompany serial number $S = u_1^{\frac{1}{s+j+1}}$. Since R is chosen by the random oracle, the two R shall be different in the two transaction. To deviate from these valid tags, A must fake the proof during Spend Protocol which we already shown to happen with negligible probability only. Similarly, $T_c = u_0^x u_1^{\frac{R}{y+1}}$ is the only valid T_c to accompany the wallet (s, t, x, y, r) for CSpend and faking the tag is only possible with negligible probability. Thus, $\mathcal{A}$'s success probability is negligible.

Anonymity. Assume the pairing group and the group $\mathbb{G}_p$ is given. It first guesses whether the challenge is normal spend/batch spend or compact spend.

If it is compact spend, $\mathcal{S}$ receives a k-DDHI problem instance in group $\mathbb{G}_p$. Using the k-DDHI problem instance, $\mathcal{S}$ is able to generate u_1 such that it knows $u_1^{\frac{1}{t+j+1}}$ for some unknown t for $j = 1 \cdots, k-1$. Since the generators are the output of the hash function, $\mathcal{S}$ can back patch u_1 to such value.

$\mathcal{A}$ needs to choose the bank's secret key (γ, γ_r) for scheme 1 or (α, β, γ_1, γ_2, γ_3, γ_4, α_r, β_r) for scheme 2, and publishes the signatures on $i = 1, \cdots k$.

The proof basically follows the VRF from [15]. $\mathcal{S}$ randomly generates two public keys PK and PK'. It chooses one of the wallets during the withdrawal protocol as w^*. For other wallets, $\mathcal{S}$ simply follows the withdrawal and spend protocols honestly. For wallet w^*, if Spend Protocol is required, it uses the k-DDHI problem instance to generate T. Though t is unknown to $\mathcal{S}$, it can always simulate the proof perfectly. In fact, since during withdrawal, there always exists an r such that $C = \text{Commit}(s, t, x, y, r)$ for any s, t, x, y, such proof is perfect. During the challenge spending, if w^* is chosen, it uses the problem instance to generate T. If the problem instance is a valid k-DDHI tuple, then the simulation is perfect. Otherwise, it contains no information on the public key.

On the other hand, if it is compact spend, a problem instance of the DDHI problem is needed. $\mathcal{S}$ uses the DDHI problem instance to generate u_1 such that it knows $u_1^{\frac{1}{y+1}}$ for some unknown y(if it is a valid DDHI tuple) and the rest is same as above.

Exculpability. In RevokeDoubleSpender, either one or both transcript contains the proof of correctness of T or T_c, which involve 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 u_0. This happens with negligible probability under the discrete logarithm assumption. □