Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 216973, 19 pages
http://dx.doi.org/10.1155/2014/216973

Research Article

Date Attachable Offline Electronic Cash Scheme

Chun-I Fan, Wei-Zhe Sun, and Hoi-Tung Hau

Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan

Correspondence should be addressed to Chun-I Fan; cifan@faculty.nsysu.edu.tw

Received 15 January 2014; Accepted 26 February 2014; Published 18 May 2014

Academic Editors: T. Cao, M. Ivanovic, and F. Yu

Electronic cash (e-cash) is definitely one of the most popular research topics in the e-commerce field. It is very important that e-cash be able to hold the anonymity and accuracy in order to preserve the privacy and rights of customers. There are two types of e-cash in general, which are online e-cash and offline e-cash. Both systems have their own pros and cons and they can be used to construct various applications. In this paper, we pioneer to propose a provably secure and efficient offline e-cash scheme with date attachability based on the blind signature technique, where expiration date and deposit date can be embedded in an e-cash simultaneously. With the help of expiration date, the bank can manage the huge database much more easily against unlimited growth, and the deposit date cannot be forged so that users are able to calculate the amount of interests they can receive in the future correctly. Furthermore, we offer security analysis and formal proofs for all essential properties of offline e-cash, which are anonymity control, unforgeability, conditional-traceability, and no-swindling.

1. Introduction

Due to the rapid growth of the Internet and communication developments, electronic commerce has become much more popular and widely used than ever [1–8]. The mobile telecommunications have been developed from 2 G to 3.5 G. Furthermore, LTE Advanced, 4 G, and 5 G are being implemented to the market in recent years. With the convenience of mobile network, people can do shopping or electronic payments by using any devices with network capability instead of leaving home. As a result, electronic commerce has been emphasized nowadays. Electronic cash (e-cash) is definitely one of the most popular research topics among electronic commerce. E-cash and the traditional cash notes are very much alike except e-cash is digitized and used on Internet transactions; therefore, it is very important that e-cash be able to hold the accuracy, privacy, and all other security concerns.

A typical e-cash system usually consists of payers (customers), payees (shops), and a bank. There are two types of e-cash in general which are online e-cash [9–13] and offline e-cash [14–27]. Online e-cash system involves participation of the bank during transactions (the payment stage). Banks are able to check whether customers have double-spent the e-cash(s) or not, and if yes, banks can terminate the transactions at once. Thus, the bank has to be online during every transaction and it may lead to a bottleneck of the system. On the other hand, while banks do not participate in the payment stage of offline e-cash systems, double-spending check is only held during the deposit stage. Yet, the bank is set to be offline, but the system design is usually much more complicated than the online type and it may lead to a longer transaction time. Since both systems have their own pros and cons, they are used under different circumstances.

Extending online and offline e-cash systems, many e-cash schemes with other different features have been proposed over the years. For instance, e-cash can be stored compactly such that the space to store these e-cash is much reduced [15, 16], e-cash is generated by multiauthorities instead of one bank only [25], exact payments e-cash [13], recoverable e-cash which can be recovered when an e-cash is lost [26], and so on.

Based on the majority of the existing approaches, we summarize that a secure e-cash system should satisfy the following requirements.

(i) *Anonymity*: no one, except the judge, can obtain any information of the e-cash owner's identity from the contents of e-cash.

(ii) *Unlinkability*: no one, except the judge, can link any e-cash payment contents.

(iii) *Unforgeability*: no one, except the bank, can generate a legal e-cash.

(iv) *Double-Spending Control*: banks should have the ability to check if the e-cash is double-spent or not. No e-cash is allowed to be spent twice or more in an e-cash system.

(v) *Conditional-Traceability*: the system should be able to trace and revoke the anonymity of users who violate any of the security rules so that they will receive penalties.

(vi) *No-swindling*: no one, except the real owner, can spend a valid offline e-cash successfully.

In order to perform double-spending checks, banks have to store information of e-cash(s) in their database. Thus, the database of banks grows in direct proportion to the number of e-cash(s) withdrawn. Embedding an expiration date into each e-cash has been considered since it helps the banks to manage the database more easily. On the other hand, customers have to exchange their expired e-cash(s) with banks for new ones so as to keep the validity of the e-cash. Furthermore, customers will receive interest from banks after cash is deposited. In order to guarantee customers will receive the right amount of interest, it is necessary for customers to attach the deposit date to their e-cash(s) and the date cannot be modified by anyone else [11]. So far, there are a number of online e-cash schemes with an expiration date attachment [9, 11, 28]. However, there are very few offline approaches [21].

In this paper, we are going to propose an efficient date attachable offline e-cash scheme and provide formal proofs on essential properties to it in the random oracle model. Considering the practical needs, we pioneer to embed two kinds of date, which are expiration data and deposit date, to the offline e-cash. Moreover, we will offer an *E-cash renewal protocol* in our scheme (Section 3.2.5). Users can exchange their unused expired e-cash for a new one with another valid expiration date more efficiently. Compared with other similar works, our scheme is efficient from the aspect of considering computation cost.

The rest of this paper is organized as follows. In Section 2, we briefly review techniques employed throughout our scheme. Our proposed scheme is described in Section 3 in detail. Security proofs and analysis are covered in Section 4. Features and performance comparisons are made in Section 5, and the conclusion is given in Section 6.

2. Preliminaries

In this section, we briefly review techniques used in our date attachable offline e-cash scheme.

2.1. Chaum's Blind Signature Scheme. Blind signature was first introduced by Chaum [29]. It has been widely used in e-cash protocols since it has been proposed. A signer will not be able to view the content of the message while she/he is signing the message. Afterwards, a user can get a message with the signature of the signer by unblinding the signed message. The protocol is described as follows.

(1) Initialization:

The signer randomly chooses two distinct large primes p and q, then computes $n = pq$ and $\phi(n) = (p-1)(q-1)$. Afterwards, the signer selects two integers e and d at random such that $ed \equiv 1 (\mathrm{mod}\ \phi(n))$. Finally, the signer publishes the public parameters (e, n) and a one-way hash function H.

(2) User $\rightarrow$ Signer: α

The user chooses a message m and a random integer r in $\mathbb{Z}_n^*$, then blinds the message by computing $\alpha = r^e H(m) \bmod n$ and sends it to the signer.

(3) Signer $\rightarrow$ User: t

After receiving α, the signer signs it with her/his private key d and sends it back to the user. The signed message will be $t = \alpha^d \bmod n$.

(4) Unblinding:

After receiving t from the signer, the user unblinds it by computing $s = r^{-1}t \bmod n$. The signature-message pair is (s, m).

(5) Verification:

The (s, m) can be verified by checking if $s^e \equiv H(m) (\mathrm{mod}\ n)$ is true or not.

2.2. Chameleon Hashing Based on Discrete Logarithm. Chameleon hashing was proposed by Krawczyk and Rabin [30]. The chameleon hash function is associated with a one-time public-private key pair; it is a collision resistant function except for users who own a trapdoor for finding collision. Any user who knows the public key can compute the hashing, and for those who do not know the private key (trapdoor), it is impossible for them to find any two inputs which lead to the same hashing output. On the contrary, any user who knows the trapdoor can find the collision of given inputs. The construction of the chameleon hashing based on discrete logarithm is described as follows.

(1) *Setup*:

(i) p, q: two large primes such that $p = kq + 1$,

(ii) g: an element order q in $\mathbb{Z}_p^*$,

(iii) x: private key in $\mathbb{Z}_q^*$,

(iv) y: public key, where $y = g^x \bmod p$.

(2) *The function*: a message $m \in \mathbb{Z}_q^*$ is given and a random integer $r \in \mathbb{Z}_q^*$ is chosen. The hash is defined as $\text{CHAM-HASH}_y(m, r) = g^m y^r \bmod p$.

(3) *Collision*: for a user who knows x, she/he is able to find the collision of the hash for any given m, m' such that $\text{CHAM-HASH}_y(m, r) = \text{CHAM-HASH}_y(m', r')$. The user derives r' in the equation $m + xr = m' + xr' (\mathrm{mod}\ q)$.

3. The Proposed Date Attachable Offline Electronic Cash Scheme

In this section, we will introduce a new date attachable offline e-cash scheme. Considering the issues mentioned in Section 1, we propose a secure offline e-cash scheme with two specific kinds of date attached to the e-cash, which are expiration date and deposit date.

3.1. Outline of the Proposed Scheme. Here we are going to briefly describe the procedures of our scheme. The proposed scheme contains four protocols, *withdrawal protocol, payment protocol, deposit protocol,* and *e-cash renewal protocol.* A user withdraws an e-cash with an expiration date attached to it from the bank. A trusted computing platform (i.e., *judge device*) [31, 32], as stated in the proposed scheme, is installed in the bank to hold the identity information of all users and it will further help trace users when it is needed. It is impossible for anyone except the judge to obtain any information embedded in the device [33]. Nowadays, judge device can be implemented by the technique of Trusted Platform Module (TPM) [32, 34] in practice.

Before an e-cash is deposited, the depositor attaches the deposit date on the e-cash and sends it to the bank during the deposit stage. When the bank receives an e-cash, it will perform double-spending checking to verify whether the e-cash is doubly spent or not. The bank can derive secret parameters of the user who does double-spending and let the judge revoke the anonymity of the user. Besides, when an unused e-cash is expired, a user will be able to exchange it for a new one with a new expiration date. In our scheme, for the efficiency concerns, some of the unused parameters of users can remain unchanged while exchanging for a new valid e-cash. In the following sections, we will describe our scheme in detail.

3.2. The Proposed Scheme. Firstly, we define some notations as follows.

(1) H_1, H_2, H_3: three one-way hash functions, $H_1, H_2, H_3 : \{0,1\}^* \rightarrow \{0,1\}^n$.

(2) H_4, H_5: two one-way hash functions, $H_4, H_5 : \{0,1\}^* \rightarrow \{0,1\}^q$.

(3) $\widetilde{E}_x, \widetilde{D}_x$: a secure symmetric cryptosystem. Plaintext is both encrypted and decrypted with a symmetric key x.

(4) $\widehat{E}_{pk}, \widehat{D}_{sk}$: a secure asymmetric cryptosystem. Plaintext is encrypted with a public key pk and decrypted with the corresponding private key sk.

(5) (pk_j, sk_j): the public-private key pair of the judge.

(6) (e_b, d_b): the public-private key pair of bank.

(7) *Date*: expiration date. It represents an effective spending date of a withdrawn e-cash. Any e-cash withdrawn in the same period will have the same expiration date, and vice versa.

(8) ID_c: the identity of user C.

(9) l_k, l_r: the security parameters.

(10) *A judge device*: a tamper-resistant device which is issued by the judge. It is installed into the system of the bank. It is impossible to intercept or modify any information stored in the device.

3.2.1. Initialization. Initially, the bank randomly chooses two distinct large primes (p_b, q_b) and computes RSA parameters $n_b = p_b q_b$. It selects an integer e_b at random such that $\mathrm{GCD}(\phi(n_b), e_b) = 1$, where $\phi(n_b) = (p_b - 1)(q_b - 1)$ and $1 < e_b < \phi(n_b)$. Then, it finds a d_b such that $e_b d_b \equiv 1 (\mathrm{mod}\ \phi(n_b))$. Secondly, it also chooses two other large primes p and q and two generators g_1 and g_2 of order q in $\mathbb{Z}_p^*$. Then, the bank publishes $(n_b, e_b, p, q, g_1, g_2, pk_j, H_1, H_2, H_3, H_4, H_5, \widetilde{E}, \widetilde{D}, \widehat{E}, \widehat{D})$. Meanwhile, the judge embeds $(n_b, e_b, p, q, g_1, g_2, pk_j, sk_j, H_1, H_2, H_3, H_4, H_5, \widetilde{E}, \widetilde{D}, \widehat{E}, \widehat{D})$ into a judge device and issues it to the bank.

3.2.2. Withdrawal Protocol. Users run the withdrawal protocol with banks to get an e-cash, as shown in Figure 1, yet banks have to obtain information of users' identity, such as ID_c or account numbers, before the withdrawal protocol is proceeded. Therefore, users should perform an authentication with banks beforehand. Users can execute the withdrawal protocol by any devices that have the ability to compute and connect to the network. For instance, users can use mobile phones or computers to perform the withdrawal protocol and store the withdrawn e-cash. The detailed steps of the protocol are as follows.

(1) Bank $\rightarrow$ User: D

Firstly, the user prepares parameters for withdrawing an e-cash. The user chooses integers a, x_1, x_2, r_1, r_2, and r_3 in random, where $a \in_R \mathbb{Z}_{n_b}^*$ and $x_1, x_2, r_1, r_2, r_3 \in_R \{0, 1, \ldots, q-1\}$ and selects a string $k \in_R \{0,1\}^{l_k}$ randomly. The user then computes (y_1, w_1, y_2, w_2), where $y_i = g_i^{x_i} \bmod p$ and $w_i = g_i^{r_i} \bmod p$ for $i = \{1, 2\}$. Secondly, the bank computes parameters for expiration date. It randomly chooses a r in $\mathbb{Z}_n^*$, prepares $D = \mathrm{Date} \parallel r$ for some expiration date *Date*. The bank will send D to the user when she/he requests to withdraw an e-cash.

(2) User $\rightarrow$ Bank: (α, ϵ)

After receiving D, the user prepares $\epsilon = \widehat{E}_{pk_j}(k \parallel \mathrm{ID}_c)$ and

$$\alpha = \left[a^{e_b} H_1^2(m \parallel D)\right]^{-1} \bmod n_b, \tag{1}$$

where $m = (y_1 \parallel w_1 \parallel y_2 \parallel w_2 \parallel r_3)$. Finally, the user sends (α, ϵ) to the bank.

(3) Bank $\rightarrow$ Judge device: (ϵ, μ, D)

The bank sets $\mu = \mathrm{ID}_c$, where ID_c is the identity of user C, and inputs it together with ϵ and D to the judge device.

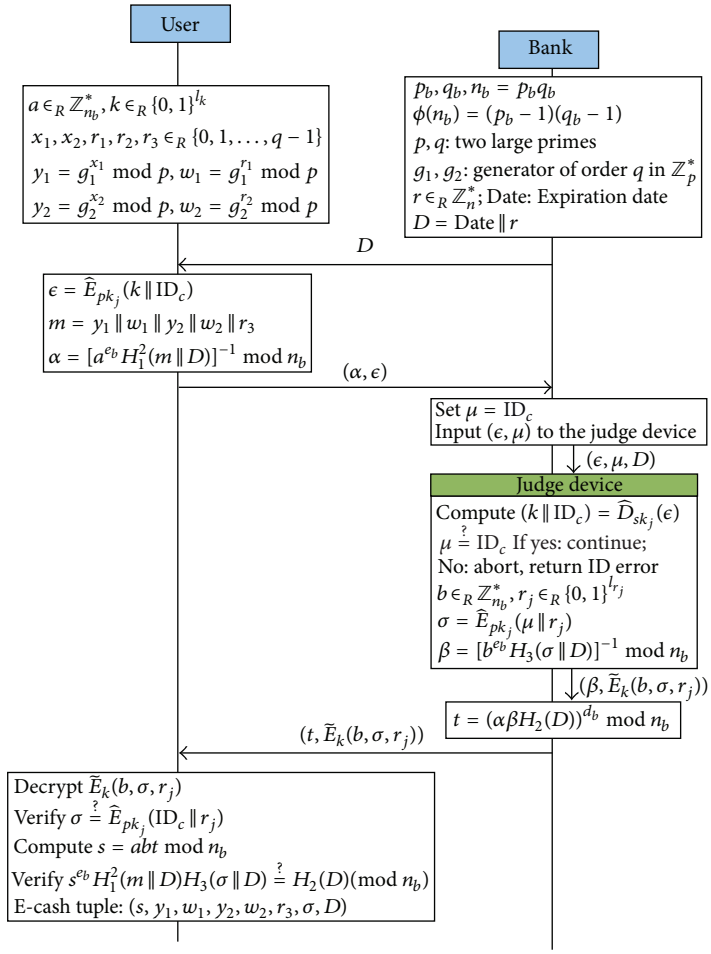

Figure 1: Withdrawal protocol.

(4) Judge device $\rightarrow$ Bank: $(\beta, \widetilde{E}_k(b, \sigma, r_j))$

The judge device decrypts ϵ and checks if $\mu = \text{ID}_c$. If not, it returns "ID error" to the bank; or else, it picks a random integer $b \in_R \mathbb{Z}_{n_b}^*$ and a string $r_j \in_R \{0,1\}^{l_{r_j}}$ randomly. Then it computes $\sigma = \widehat{E}_{pk_j}(\mu \| r_j)$ and

$$\beta = \left[b^{e_b} H_3(\sigma \| D)\right]^{-1} \bmod n_b. \tag{2}$$

Finally, it encrypts (b, σ, r_j) by using the symmetric key k and outputs it together with β to the bank.

(5) Bank $\rightarrow$ User: $(t, \widetilde{E}_k(b, \sigma, r_j))$

After receiving $(\beta, \widetilde{E}_k(b, \sigma, r_j))$ from the judge device, it computes

$$t = \left(\alpha\beta H_2(D)\right)^{d_b} \bmod n_b \tag{3}$$

and sends $(t, \widetilde{E}_k(b, \sigma, r_j))$ to the user.

(6) Verifications

After receiving $(t, \widetilde{E}_k(b, \sigma, r_j))$, the user firstly decrypts the ciphertext by using the symmetric key k in order to obtain (b, σ, r_j). Secondly, she/he checks if his/her ID is embedded correctly by computing if $\sigma = \widehat{E}_{pk_j}(\text{ID}_c \| r_j)$ is true or not. Thirdly, she/he computes

$$s = abt \bmod n_b \tag{4}$$

and verifies s by checking if

$$s^{e_b} H_1^2(m \| D) H_3(\sigma \| D) = H_2(D) \left(\bmod n_b\right) \tag{5}$$

is true or not. Finally, when all verifications are done, the user gets the e-cash tuples (s, m, σ, D) and stores (x_1, x_2, r_1, r_2) for further payment usages.

3.2.3. Payment Protocol. When a user has to spend the e-cash, she/he performs the protocol as shown in Figure 2. The steps of the protocol are described as follows.

(1) User $\rightarrow$ Shop: $(s, m, \sigma, D, x_2, r_2)$

The user sends $(s, m, \sigma, D, x_2, r_2)$ to the shop, where D contains the expiration date of the e-cash.

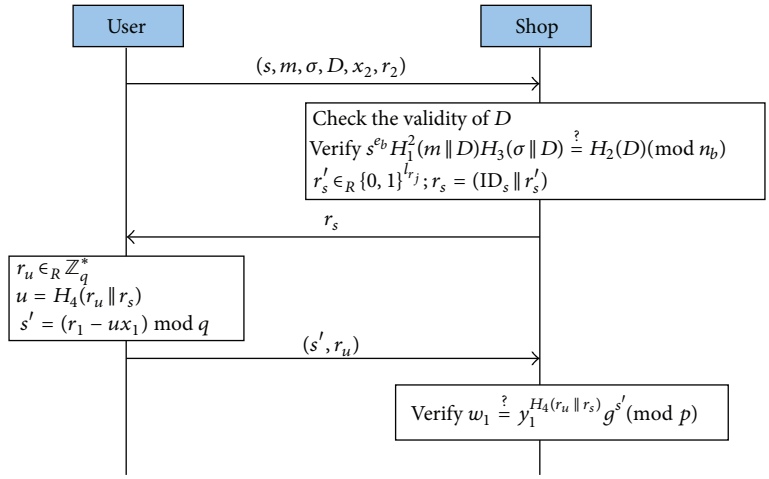

Figure 2: Payment protocol.

(2) Shop $\rightarrow$ User: r_s

The shop first checks D to verify if the e-cash is still within the expiration date or not. If not, it terminates the transaction. Otherwise, it continues to verify $s^{e_b} H_1^2(m \parallel D) H_3(\sigma \parallel D) = H_2(D) (\mathrm{mod}\, n_b)$. If it is not valid, the protocol is aborted; or else, it selects a string $r_s' \in_R \{0,1\}^{l_{r_j}}$ and sets a challenge $r_s = (\mathrm{ID}_s \parallel r_s')$, where ID_s is the identity of the shop. Finally, it sends r_s to the user.

(3) User $\rightarrow$ Shop: (s', r_u)

After receiving r_s from the shop, the user randomly selects a $r_u \in_R \mathbb{Z}_q^*$ and computes a response to the challenge

$$s' = (r_1 - ux_1) \bmod q, \tag{6}$$

where $u = H_4(r_u \parallel r_s)$. Then, the user sends (s', r_u) to the shop.

(4) Verifications

After receiving (s', r_u) from the user, the shop verifies if $w_1 = y_1^{H_4(r_u \| r_s)} g^{s'} (\mathrm{mod}\, p)$ is true or not. If it is true, the shop will accept the e-cash. On the other hand, if it is not, the shop will reject it. Since it is an offline e-cash, the shop does not have to deposit it to the bank immediately. It can store the e-cash and deposit it later together with other received e-cash(s).

3.2.4. Deposit Protocol. As Figure 3 shows, shops attach the deposit date to their e-cash(s) and deposit them to banks in this protocol. Banks perform double-spending checks when they receive these e-cash(s). If any e-cash is double-spent, the bank will revoke the anonymity of the e-cash owner with the help of the judge. The steps are described in detail as follows.

(1) Shop $\rightarrow$ Bank: $(s, m, \sigma, D, d, r_4, s', r_u, r_s)$

The shop computes $r_4 = r_2 - x_2 H_5(d)$, where d is the deposit date, and sends $(s, m, \sigma, D, d, r_4, s', r_u, r_s)$ to the bank.

(2) Verifications

Firstly, the bank checks the correctness of expiration date D and deposit date d, respectively, and also checks if

$$\begin{aligned} w_2 &= y_2^{H_5(d)} g_2^{r_4} \bmod p, \\ w_1 &= y_1^{H_4(r_u \| r_s)} g_2^{s'} \bmod p \end{aligned} \tag{7}$$

are true or not. Secondly, the bank verifies if $s^{e_b} H_1^2(m \parallel D) H_3(\sigma \parallel D) = H_2(D) (\mathrm{mod}\, n_b)$ and checks the uniqueness of (s, m, σ, D). Finally, if all of the above facts are verified successfully, the bank will accept and store the e-cash in its database and record $H_1(m \parallel D)$ in *exchange list*. Otherwise, it will reject this transaction and trace the owner of the e-cash.

3.2.5. E-Cash Renewal Protocol. In order to reduce the unlimited growth database problem of the bank, we have expiration date and renewal protocol in our scheme to achieve it, as shown in Figure 4. When an unused e-cash is expired, the user has to exchange it for another e-cash with a new expiration date from the bank.

(1) User $\rightarrow$ Bank: (s, ρ, σ, D)

The user recalls $m = (y_1, w_1, y_2, w_2, x_2, r_3)$ and prepares

$$\rho = H_1 (m \parallel D) \tag{8}$$

and sends it together with the unused (s, σ, D) to the bank.

(2) Verifications

Firstly, the bank checks the correctness of expiration date D and makes sure ρ does not exist in the *exchange list*. Secondly, the bank verifies if $s^{e_b} H_1(\rho) H_3(\sigma \parallel D) \equiv H_2(D) (\mathrm{mod}\, n_b)$. Finally, if all of the above facts are verified successfully, the bank will accept to

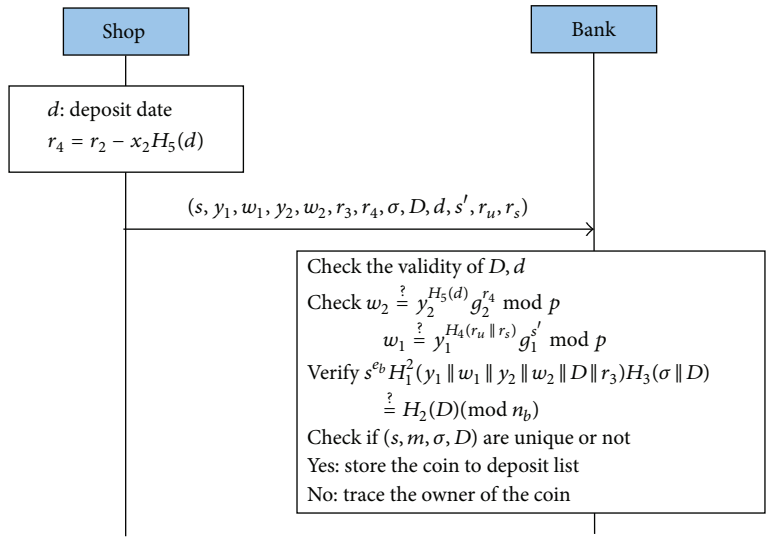

Figure 3: Deposit protocol.

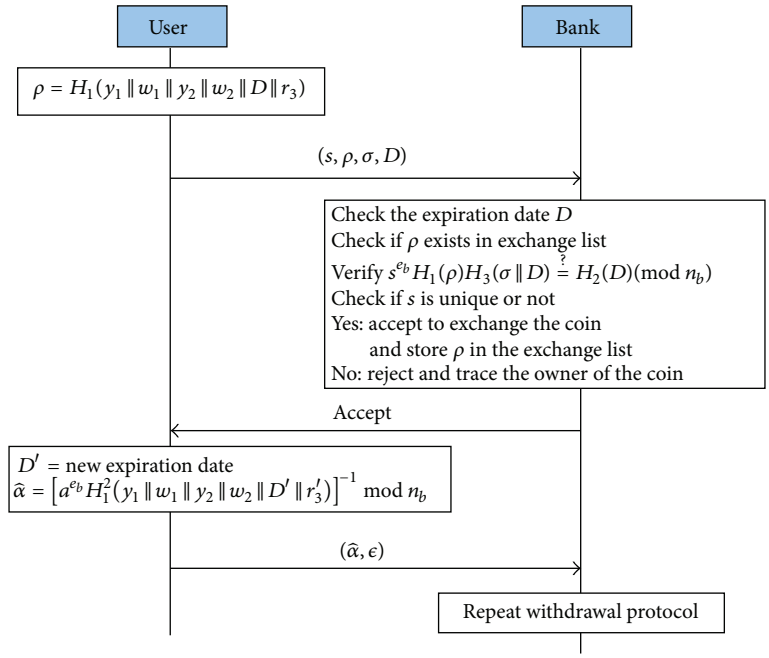

Figure 4: E-Cash renewal protocol.

exchange the e-cash. It will send a new expiration date D' and store ρ in the *exchange list*. Otherwise, it will reject the exchange request.

(3) User $\rightarrow$ Bank: $(\widehat{\alpha}, \epsilon)$

The user computes

$$\widehat{\alpha} = \left[a^{e_b} H_1^2(m' \parallel D')\right]^{-1} \bmod n_b, \quad (9)$$

where $m' = (y_1, w_1, y_2, w_2, x_2, r_3')$, r_3' is a random, and D' is the new expiration date issued by the bank. The user sends $(\widehat{\alpha}, \epsilon, \mathrm{ID}_c)$ to the bank. Then the bank repeats the withdrawal protocol in Section 3.2.2 from Step 2 with the user.

3.2.6. Double-Spending Checking and Anonymity Control. In our scheme, the identity of the users is anonymous in general except when the users violate any security rules and, therefore, their identities will be revealed.

(1) Double-Spending Checking

When an e-cash is being doubly spent, there must be two e-cash(s) with the same record prefixed by $(s, y_1, w_1, y_2, w_2, r_3, \sigma, D)$ stored in the database of the

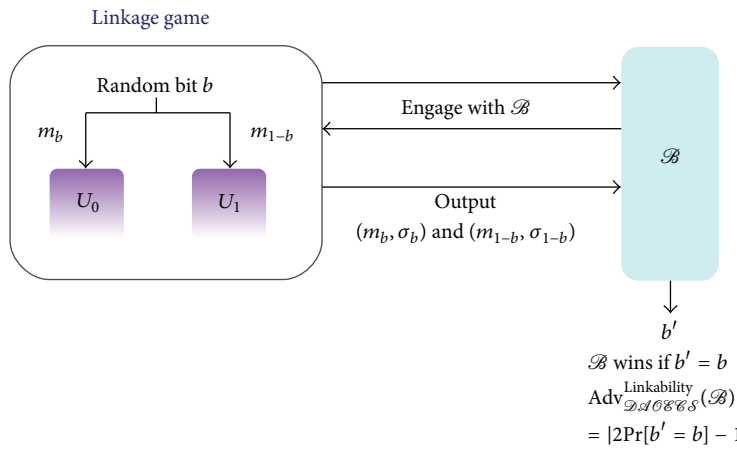

Figure 5: The game environment of linkage game.

bank. Therefore, the bank is able to detect any double-spent e-cash easily by checking the above parameters. For instance, the bank has received two e-cash(s),

$$\begin{aligned}&\left(s, y_1, w_1, y_2, w_2, x_2, r_3, r_4, \sigma, D, d, s', r_u, r_s\right),\\&\left(s, y_1, w_1, y_2, w_2, x_2, r_3, \widehat{r_4}, \sigma, D, \widehat{d}, \widehat{s'}, \widehat{r}_u, \widehat{r}_s\right).\end{aligned} \tag{10}$$

Thus, the bank can obtain two equations as follows:

$$\begin{aligned}s' &\equiv r_1 - H_4\left(r_u \parallel r_s\right) x_1 \left(\mathrm{mod}\, q\right),\\ \widehat{s'} &\equiv r_1 - H_4\left(\widehat{r}_u \parallel \widehat{r}_s\right) x_1 \left(\mathrm{mod}\, q\right).\end{aligned} \tag{11}$$

The bank can derive (x_1, r_1) from the above equations and send $(s, y_1, w_1, y_2, w_2, x_2, r_3, \sigma, D)$ and (x_1, r_1) to the judge to trace the owner of the e-cash.

(2) Revocation

The judge can trace any user who doubly spends e-cash(s) or violates any transaction regulations. When the judge receives $(s, y_1, w_1, y_2, w_2, x_2, r_3, \sigma, D)$ and (x_1, r_1) from the bank, it checks the following equations:

$$\begin{aligned}s^{e_b} H_1^2\left(m \parallel D\right) H_3\left(\sigma \parallel D\right) &\stackrel{?}{\equiv} H_2\left(D\right) \left(\mathrm{mod}\, n_b\right),\\ y_1 &\stackrel{?}{\equiv} g_1^{x_1} \left(\mathrm{mod}\, p\right),\\ w_1 &\stackrel{?}{\equiv} g_1^{r_1} \left(\mathrm{mod}\, p\right).\end{aligned} \tag{12}$$

If all of the above equalities are true, the judge will decrypt σ and return the extracted ID_c to the bank.

4. Security Proofs

In this section, we provide security definitions and formal proofs of the following security features: unlinkability, unforgeability, traceability, and no-swindling for our proposed date attachable offline electronic cash scheme ($\mathscr{DAOECS}$).

4.1. E-Cash Unlinkability. Based on the definition of unlinkability introduced by Abe and Okamoto [35] and Juels et al. [36], we formally define the unlinkability property of $\mathscr{DAOECS}$.

Definition 1 (The Linkage Game). Let U_0, U_1, and $\mathscr{J}$ be two honest users and the judge that follows $\mathscr{DAOECS}$, respectively. Let $\mathscr{B}$ be the bank that participates the following game with U_0, U_1, and $\mathscr{J}$. The game environment is shown in Figure 5.

Step 1. According to $\mathscr{DAOECS}$, $\mathscr{B}$ generates the bank's public key (e_b, n_b), the bank's private key (d_b, p_b, q_b), system parameters (p, q, g_1, g_2), the expiration date D, and the five public one-way hash functions H_1, H_2, H_3, H_4, and H_5. $\mathscr{J}$ generates the judge's public-private key pair (pk_j, sk_j).

Step 2. $\mathscr{B}$ generates $x_{1i}, x_{2i}, r_{1i}, r_{2i}, r_{3i}$ in random, where x_1, x_2, r_1, r_2, $r_3 \in_R \{0, 1, \ldots, q-1\}$, and computes (y_{ki}, w_{ki}) for $k = \{1, 2\}$ and $i = \{0, 1\}$, where $y_{ki} = g_k^{x_k} \bmod p$ and $w_{ki} = g_k^{r_k} \bmod p$.

Step 3. We choose a bit $\widehat{b} \in \{0, 1\}$ randomly and place $(y_{1\widehat{b}}, w_{1\widehat{b}}, y_{2\widehat{b}}, w_{2\widehat{b}})$ and $(y_{1_{1-\widehat{b}}}, w_{1_{1-\widehat{b}}}, y_{2_{1-\widehat{b}}}, w_{2_{1-\widehat{b}}})$ on the private input tapes of U_0 and U_1, respectively, where $\widehat{b}$ is not disclosed to $\mathscr{B}$.

Step 4. $\mathscr{B}$ performs the withdrawal protocol of $\mathscr{DAOECS}$ with U_0 and U_1, respectively.

Step 5. If U_0 and U_1 output two e-cash(s) $(s_{\widetilde{b}}, m_{\widetilde{b}}, \sigma_{\widetilde{b}}, D_{\widetilde{b}})$ and $(s_{1-\widetilde{b}}, m_{1-\widetilde{b}}, \sigma_{1-\widetilde{b}}, D_{1-\widetilde{b}})$, where $m_i = (y_{1_i}, w_{1_i}, y_{2_i}, w_{2_i}, r_{3_i})$, on their private tapes, respectively, we give the two e-cash(s) in a random order to $\mathscr{B}$; otherwise, $\perp$ is given to $\mathscr{B}$.

Experiment $\mathbf{Exp}_{\mathscr{A}}^{\text{FG-1}}(l_k)$
$\left(pk_j, sk_j, g_1, g_2, e_b, d_b, p_b, q_b, n_b, H_1, H_2, H_3, H_4, H_5\right) \leftarrow \text{Setup}(l_k)$
$\{(s_1, m_1, \sigma_1, D_1), \ldots, (s_{\ell+1}, m_{\ell+1}, \sigma_{\ell+1}, D_{\ell+1})\} \leftarrow \mathscr{A}^{\mathscr{O}_S}\left(pk_j, g_1, g_2, e_b, n_b, H_1, H_2, H_3, H_4, H_5\right)$
if the following checks are true, **return 1;**
(i) $s_i^{e_b} H_1^2(m_i) H_3(\sigma_i \parallel D_i) \equiv H_2(D_i) \pmod{n_b}$, $\forall i \in \{1, \ldots, \ell+1\}$;
(ii) $m_1, \ldots, m_{\ell+1}$ are all distinct
else **return 0;**

Algorithm 1: Experiment FG-1.

Step 6. $\mathscr{B}$ outputs $\widehat{b}' \in \{0,1\}$ as the guess of $\widehat{b}$. The bank $\mathscr{B}$ wins the game if $\widehat{b}' = \widehat{b}$ and $\mathscr{J}$ has not revoked the anonymity of $(s_{\widehat{b}}, m_{\widehat{b}}, \sigma_{\widehat{b}}, D_{\widehat{b}})$ and $(s_{1-\widehat{b}}, m_{1-\widehat{b}}, \sigma_{1-\widehat{b}}, D_{1-\widehat{b}})$ to $\mathscr{B}$. We define the advantage of $\mathscr{B}$ as

$$\text{Adv}_{\mathscr{DAOECS}}^{\text{Linkability}}(\mathscr{B}) = \left|2\Pr\left[\widehat{b}' = \widehat{b}\right] - 1\right|, \tag{13}$$

where $\Pr[\widehat{b}' = \widehat{b}]$ denotes the probability of $\widehat{b}' = \widehat{b}$.

Definition 2 (Unlinkability). A $\mathscr{DAOECS}$ satisfies the unlinkability property if and only if the advantage $\text{Adv}_{\mathscr{DAOECS}}^{\text{Linkability}}(\mathscr{B})$ defined in Definition 1 is negligible.

Theorem 3. *A $\mathscr{DAOECS}$ satisfies the unlinkability property of Definition 2 if the adopted cryptosystems are semantically secure.*

Proof. If $\mathscr{B}$ is given $\perp$ in the Step 5 of the game, it will determine $\widehat{b}$ with probability 1/2, which is exactly the same as a random guess of $\widehat{b}$.

Here, we assume that $\mathscr{B}$ gets two e-cash $(s_0, m_0, \sigma_0, D_0)$ and $(s_1, m_1, \sigma_1, D_1)$. Let $(\alpha_i, \beta_i, t_i, \epsilon_i, \widetilde{E}_{k_i}(b_i, \sigma_i, r_{j_i}))$, $i \in \{0,1\}$, be the view of data exchanged between U_i and $\mathscr{B}$ in the withdrawal protocol (Section 3.2.2) and let $(x_{2_i}, r_{2_i}, r_{4_i}, r_{u_i}, r_{s_i}, s_i', d_i)$ be the view of data exchanged when $\mathscr{B}$ performs the payment protocol (Section 3.2.3) and the deposit protocol (Section 3.2.4) by using $(s_i, m_i, \sigma_i, D_i)$, where $i \in \{0,1\}$.

For $(s, m, \sigma, D, x_2, r_2, r_4, r_u, r_s, s', d) \in$

$$\begin{aligned}&\left\{\left(s_0, m_0, \sigma_0, D_0, x_{2_0}, r_{2_0}, r_{4_0}, r_{u_0}, r_{s_0}, s_0', d_0\right),\right.\\&\left.\left(s_1, m_1, \sigma_1, D_1, x_{2_1}, r_{2_1}, r_{4_1}, r_{u_1}, r_{s_1}, s_1', d_1\right)\right\}\end{aligned} \tag{14}$$

and $(\alpha_i, \beta_i, t_i, \epsilon_i, \widetilde{E}_{k_i}(b_i, \sigma_i, r_{j_i}))$, $i \in \{0,1\}$, there always exists a pair (a_i', b_i') such that

$$\begin{aligned}a_i' &= \left[\alpha_i H_1^2(m \parallel D)\right]^{-d_b} \bmod n_b \ \ (\text{via (1)}),\\ b_i' &= \left[\beta_i H_3(\sigma \parallel D)\right]^{-d_b} \bmod n_b \ \ (\text{via (2)}).\end{aligned} \tag{15}$$

And from (3), $t_i \equiv (\alpha_i \beta_i H_2(D))^{d_b} \pmod{n_b}$, (4) always holds as

$$\begin{aligned}s &\equiv \left(a_i' b_i' t_i\right)\\ &\equiv \left[\left(H_1^2(m \parallel D) H_3(\sigma \parallel D)\right)^{-1} H_2(D)\right]^{d_b} \left(\text{mod } n_b\right).\end{aligned} \tag{16}$$

Besides, $\widehat{E}_{pk_j}$ and $\widetilde{E}_{k_i}$ are semantically secure encryption functions. $\mathscr{B}$ cannot learn any information from ϵ_i and $\widetilde{E}_{k_i}(b_i, \sigma_i, r_{j_i})$.

From the above, given any $(s, m, \sigma, D) \in \{(s_0, m_0, \sigma_0, D_0), (s_1, m_1, \sigma_1, D_1)\}$ and (α_i, β_i, t_i), where $i \in \{0,1\}$, there always exists a corresponding pair (a_i', b_i') such that (1), (2), (3), and (4) are satisfied.

Thus, go back to Step 6 of the game, the bank $\mathscr{B}$ succeeds in determining $\widehat{b}$ with probability $(1/2) + \varepsilon$, where ε is negligible since $\widehat{E}$ and $\widetilde{E}$ are semantically secure. Therefore, we have $\text{Adv}_{\mathscr{DAOECS}}^{\text{Linkability}}(\mathscr{B}) = 2\varepsilon$, which is negligible, so that $\mathscr{DAOECS}$ satisfies the unlinkability property. □

4.2. E-Cash Unforgeability. In this section, we will formally prove that the proposed date attachable offline electronic cash scheme ($\mathscr{DAOECS}$) is secure against forgery attack. The forgery attack can be roughly divided into two types, one is the typical one-more forgery type (i.e., $(\ell, \ell+1)$-forgery) [37] and the other is the forgery on some specific expiration date of an e-cash after sufficient communications with the signing oracle (i.e., bank). The details of definitions and our formal proofs will be described as follows.

Definition 4 (Forgery Game 1 in $\mathscr{DAOECS}$ (FG-1)). Let $l_k \in \mathbb{N}$ be a security parameter and $\mathscr{A}$ be an adversary in $\mathscr{DAOECS}$. $\mathscr{O}_S$ is an oracle which plays the role of the bank in $\mathscr{DAOECS}$ to be responsible for issuing e-cash(s) (i.e., (s, m, σ, D), where $m = (w_1, y_1, w_2, y_2, r_3, D)$) according to the queries from $\mathscr{A}$. $\mathscr{A}$ is allowed to query $\mathscr{O}_S$ for ℓ times; consider the experiment $\text{Exp}_{\mathscr{A}}^{\text{FG-1}}(l_k)$ shown in Algorithm 1. $\mathscr{A}$ wins the forgery game FG-1 if the probability $\Pr[\text{Exp}_{\mathscr{A}}^{\text{FG-1}}(l_k) = 1]$ of $\mathscr{A}$ is nonnegligible.

Definition 5 (Forgery Game 2 in $\mathscr{DAOECS}$ (FG-2)). Let $l_k \in \mathbb{N}$ be a security parameter and $\mathscr{A}$ be an adversary in $\mathscr{DAOECS}$. $\mathscr{O}_S$ is an oracle which plays the role of the bank in $\mathscr{DAOECS}$ to take charge of the following two events:

(i) issue e-cash(s) (i.e., (s, m, σ, D), where $m = (w_1, y_1, w_2, y_2, r_3, D)$) according to the queries from $\mathscr{A}$,

(ii) record the total number ℓ_{D_i} of each distinct expiration date D_i.

$\mathscr{A}$ is allowed to query $\mathscr{O}_S$ for ℓ times; consider the experiment $\text{Exp}_{\mathscr{A}}^{\text{FG-2}}(l_k)$ shown in Algorithm 2. $\mathscr{A}$ wins the forgery game

Experiment $\mathbf{Exp}_{\mathscr{A}}^{\text{FG-2}}(l_k)$
$\left(pk_j, sk_j, g_1, g_2, e_b, d_b, p_b, q_b, n_b, H_1, H_2, H_3, H_4, H_5\right) \leftarrow \text{Setup}(l_k)$
$\{(s_i, m_i, \sigma_i, D^*), 1 \le i \le \ell_{D^*} + 1\} \leftarrow \mathscr{A}^{\mathscr{O}_S}\left(pk_j, g_1, g_2, e_b, n_b, H_1, H_2, H_3, H_4, H_5\right)$
if the following checks are true, **return 1;**
(i) $s_i^{e_b} H_1^2(m_i) H_3(\sigma_i \parallel D^*) \equiv H_2(D^*) \pmod{n_b}, \forall i \in \{1, \ldots, \ell_{D^*} + 1\}$;
(ii) $m_1, \ldots, m_{\ell_{D^*}+1}$ are all distinct;
else **return 0;**

Algorithm 2: Experiment FG-2.

Experiment $\mathbf{Exp}_{\mathscr{A}}^{\text{RSA-ACTI}}(k)$
$(N, e, d) \xleftarrow{R} KeyGen(k)$.
$(y_1, \ldots, y_m) \leftarrow \mathscr{O}_t(N, e, k)$
$\{\pi, (x_1, y_1), \ldots, (x_n, y_n)\} \leftarrow \mathscr{A}^{\mathscr{O}_{\text{inv}}, \mathscr{O}_t}(N, e, k)$
if the following checks are true, **return 1;**
(i) $\pi : \{1, \ldots, n\} \to \{1, \ldots, m\}$ is injective
(ii) $x_i^e \equiv y_i \pmod{N}, \forall i \in \{1, ..., n\}$
(iii) $n > q_h$
else **return 0;**

Algorithm 3

FG-2 if the probability $\Pr[\text{Exp}_{\mathscr{A}}^{\text{FG-2}}(k) = 1]$ of $\mathscr{A}$ is nonnegligible.

Here we introduce the hard problems used in our proof models.

Definition 6 (Alternative Formulation of RSA Chosen-Target Inversion Problem (RSA-ACTI)). Let $k \in \mathbb{N}$ be a security parameter and $\mathscr{A}$ be an adversary who is allowed to access the RSA-inversion oracle $\mathscr{O}_{\text{inv}}$ and the target oracle $\mathscr{O}_t$. $\mathscr{A}$ is allowed to query $\mathscr{O}_t$ and $\mathscr{O}_{\text{inv}}$ for m and q_h times, respectively. Consider Algorithm 3.

We say $\mathscr{A}$ breaks the RSA-ACTI problem if the probability $\Pr[\text{Exp}_{\mathscr{A}}^{\text{RSA-ACTI}}(k) = 1]$ of $\mathscr{A}$ is nonnegligible.

Definition 7 (The RSA Inversion Problem). Given (e, n), where n is the product of two distinct large primes p and q with roughly the same length and e is a positive integer relatively-prime to $(p-1)(q-1)$, and a randomly-chosen positive integer y less than n, find an integer x such that $x^e \equiv y \pmod{n}$.

Definition 8 (E-Cash Unforgeability). If there exists no probabilistic polynomial-time adversary who can win FG-1 or FG-2, then $\mathscr{DAOECCS}$ is secure against forgery attacks.

Theorem 9. *For a polynomial-time adversary $\mathscr{A}$ who can win FG-1 or FG-2 with nonnegligible probability, there exists another adversary $\mathscr{S}$ who can break the RSA-ACTI problem or RSA inversion problem with nonnegligible probability.*

Proof. $\mathscr{S}$ simulates the environment and controls three hash oracles, $\mathscr{O}_{H_1}$, $\mathscr{O}_{H_2}$, $\mathscr{O}_{H_3}$ and an e-cash producing oracle $\mathscr{O}_S$ of $\mathscr{DAOECCS}$ scheme to respond to different queries from $\mathscr{A}$ in the random oracle model and takes advantage of $\mathscr{A}$ to solve RSA-ACTI problem or RSA inversion problem, simultaneously. Then, for consistency, $\mathscr{S}$ maintains three lists $\mathscr{L}_{H_1}$, $\mathscr{L}_{H_2}$, and $\mathscr{L}_{H_3}$ to record every response of $\mathscr{O}_{H_1}$, $\mathscr{O}_{H_2}$, and $\mathscr{O}_{H_3}$, respectively.

Here we will start to do the simulation for the two games (i.e., FG-1 and FG-2) to prove $\mathscr{DAOECCS}$ is secure against forgery attacks. The details of simulation are set below and illustrated in Figures 6 and 7, respectively.

Simulation in FG-1. In this proof model, $\mathscr{S}$ is allowed to query the oracles $\mathscr{O}_{\text{inv}}$ (i.e., $(\cdot)^d$) and $\mathscr{O}_t$ of RSA-ACTI problem defined in Definition 6 for helping $\mathscr{S}$ to produce e-cash(s) and the corresponding verifying key is (e, n).

(i) H_1 Query of $\mathscr{O}_{H_1}$

Initially, every blank record in $\mathscr{L}_{H_1}$ can be represented as $(\perp, \perp, \perp)$. When $\mathscr{A}$ sends m for querying the hash value $H_1(m)$, $\mathscr{S}$ will check the list $\mathscr{L}_{H_1}$:

(a) if $m = m_i$ for some i, then $\mathscr{S}$ retrieves the corresponding $H_1(m_i)$ and returns it to $\mathscr{A}$;
(b) else if $m = H_1(m_i)$ and $H_1^2(m_i) \neq \perp$ for some i, then $\mathscr{S}$ retrieves the corresponding $H_1^2(m_i)$ and returns it to $\mathscr{A}$;
(c) else if $m = H_1(m_i)$ and $H_1^2(m_i) = \perp$ for some i, then $\mathscr{S}$ queries $\mathscr{O}_t$ to get an instance y and returns it to $\mathscr{A}$, then fills the record $(m_i, H_1(m_i), \perp)$ as $(m_i, H_1(m_i), y)$ in $\mathscr{L}_{H_1}$;
(d) otherwise, $\mathscr{S}$ selects a random $\rho \in \mathbb{Z}_n$, records $(m, \rho, \perp)$ in $\mathscr{L}_{H_1}$, and returns ρ to $\mathscr{A}$.

(ii) H_2 Query of $\mathscr{O}_{H_2}$

When $\mathscr{A}$ asks for H_2 query by sending D to $\mathscr{S}$, $\mathscr{S}$ will look up the list $\mathscr{L}_{H_2}$:

(a) if $D = D_i$ for some i, the corresponding τ will be retrieved and $\mathscr{S}$ will send $(\tau^e \bmod n)$ back to $\mathscr{A}$;
(b) otherwise, $\mathscr{S}$ will select a random $\tau \in \mathbb{Z}_n$, record (D, τ) in $\mathscr{L}_{H_2}$, and return $(\tau^e \bmod n)$ back to $\mathscr{A}$.

(iii) H_3 Query of $\mathscr{O}_{H_3}$

While $\mathscr{A}$ sends (σ, D) to $\mathscr{S}$ for $H_3(\sigma \parallel D)$, $\mathscr{S}$ will look up the list $\mathscr{L}_{H_3}$:

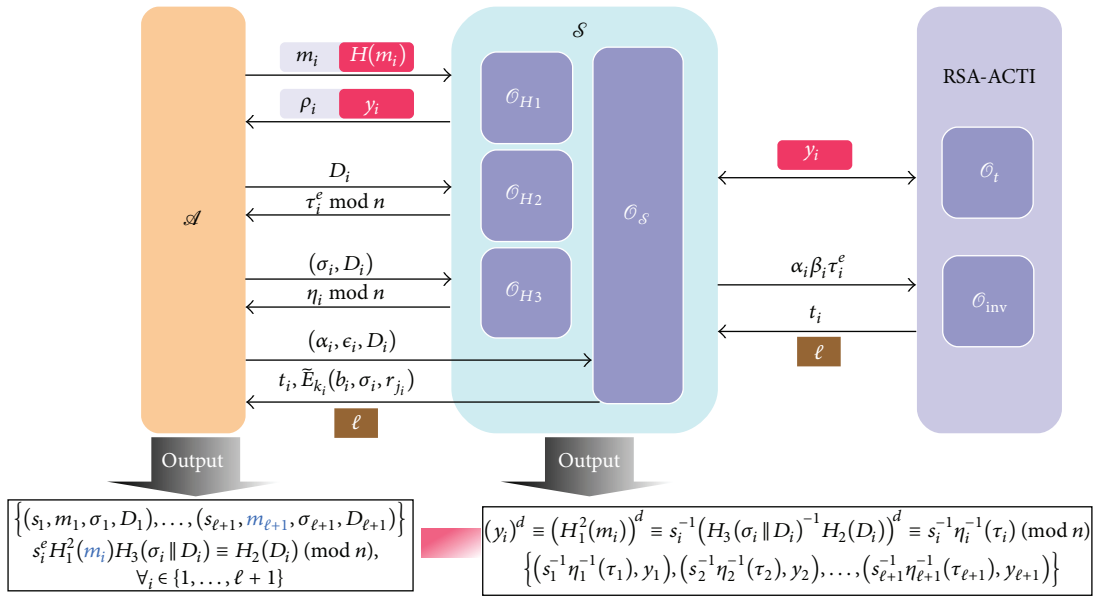

Figure 6: The proof model of FG-1.

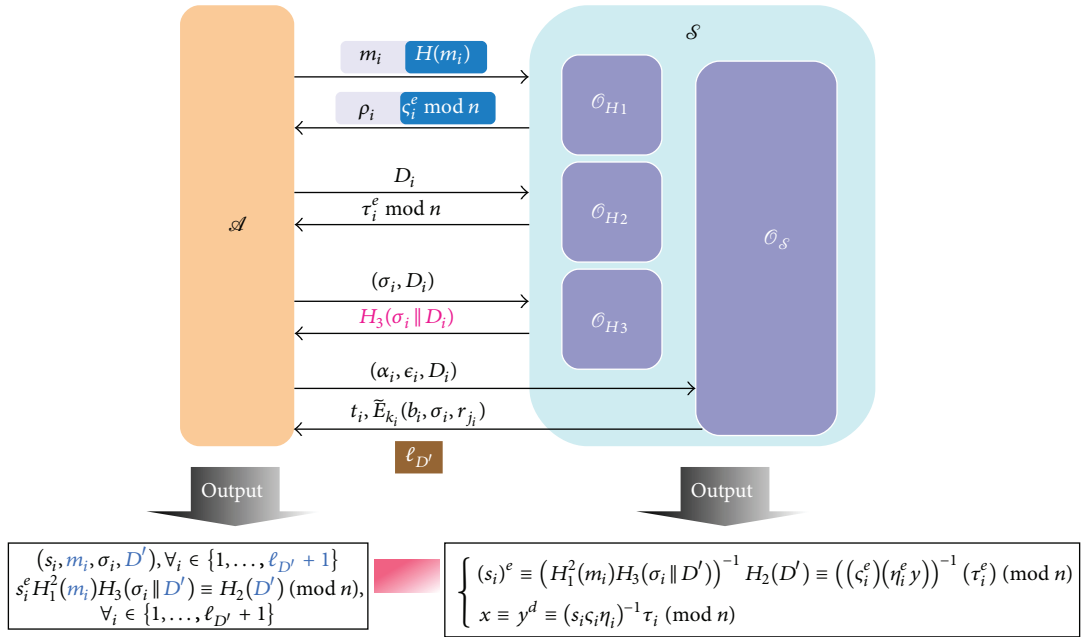

Figure 7: The proof model of FG-2.

(a) if $(\sigma, D) = (\sigma_i, D_i)$ for some i, the corresponding η will be retrieved and $(\eta^e \bmod n)$ will be returned to $\mathcal{A}$;

(b) otherwise, $\mathcal{S}$ will select a random $\eta \in \mathbb{Z}_n$, record $((\sigma, D), \eta)$ in $\mathcal{L}_{H_3}$, and return $(\eta^e \bmod n)$ back to $\mathcal{A}$.

(iv) E-Cash Producing Query of $\mathcal{O}_{\mathcal{S}}$

When $\mathcal{A}$ sends (α, ϵ, D) to $\mathcal{S}$, $\mathcal{S}$ will do the following steps:

(1) decrypt ϵ, obtain (k, ID);

(2) randomly select r_j and prepare $\sigma = \hat{E}_{pk_j}(\text{ID} \| r_j)$;

(3) choose $\eta \in_R \mathbb{Z}_n$, set $H_3(\sigma \| D) = (\eta^e \bmod n)$, and store $((\sigma, D), \eta)$ in $\mathcal{L}_{H_3}$;

(4) select $b \in_R \mathbb{Z}_n^*$ and compute $\beta = (b^e \eta^e)^{-1} \bmod n$;

(5) retrieve or assign τ such that $H_2(D) = (\tau^e)$ as the $\mathcal{O}_{H_2}$ query described above;

(6) send $(\alpha\beta\tau^e)$ to oracle $\mathcal{O}_{\text{inv}}$ to get $t = (\alpha\beta\tau^e)^d \bmod n$;

(7) return $(t, \tilde{E}_k(b, \sigma, r_j))$ back to $\mathcal{A}$.

Eventually, assume that $\mathcal{A}$ can successfully output $\ell+1$ e-cash tuples

$$\{(s_1, m_1, \sigma_1, D_1) \cdots (s_{\ell+1}, m_{\ell+1}, \sigma_{\ell+1}, D_{\ell+1})\}, \quad (17)$$

where m_i are all distinct, $\forall i, 1 \le i \le \ell+1$, such that $s_i^e H_1^2(m) H_3(\sigma_i \parallel D_i) = H_2(D_i) \pmod n$ after ℓ times to query $\mathcal{O}_{\mathcal{S}}$ with nonnegligible probability $\epsilon_{\mathcal{A}}$.

According to $\mathcal{L}_{H_1}$, $\mathcal{L}_{H_2}$, and $\mathcal{L}_{H_3}$, $\mathcal{S}$ can compute and retrieve RSA-inversion instances ($\forall i, 1 \le i \le \ell+1$)

$$\begin{aligned}(y_i)^d &\equiv \left(H_1^2(m_i)\right)^d \equiv s_i^{-1}\left(H_3(\sigma_i \parallel D_i)^{-1} H_2(D_i)\right)^d \\ &\equiv s_i^{-1}\eta_i^{-1}(\tau_i) \pmod n.\end{aligned} \quad (18)$$

Via $\mathcal{A}$ querying the signing oracle $\mathcal{O}_S$ for ℓ times (i.e., query $\mathcal{O}_{\text{inv}}$ for ℓ times by $\mathcal{S}$), $\mathcal{S}$ can output $\ell+1$ RSA-inversion instances

$$\begin{aligned}&\left\{\left(s_1^{-1}\eta_1^{-1}(\tau_1), y_1\right), \left(s_2^{-1}\eta_2^{-1}(\tau_2), y_2\right), \ldots,\right. \\ &\left.\left(s_{\ell+1}^{-1}\eta_{\ell+1}^{-1}(\tau_{\ell+1}), y_{\ell+1}\right)\right\}\end{aligned} \quad (19)$$

and break the RSA-ACTI problem with nonnegligible probability at least $\epsilon_{\mathcal{A}}$.

Simulation in FG-2. Initially, $\mathcal{S}$ is given an instance (y, e, n) of RSA inversion problem defined in Definition 7 and simulates the environment as follows.

(i) H_1 Query of $\mathcal{O}_{H_1}$

Initially, every blank record in $\mathcal{L}_{H_1}$ can be represented as $(\perp, \perp, \perp)$. When $\mathcal{A}$ sends m for querying the hash value $H_1(m)$, $\mathcal{S}$ will check the list $\mathcal{L}_{H_1}$:

(a) if $m = m_i$ for some i, then $\mathcal{S}$ retrieves the corresponding ρ_i and returns it to $\mathcal{A}$;

(b) else if $m = H_1(m_i)$ and $H_1^2(m_i) \neq \perp$ for some i, then $\mathcal{S}$ retrieves the corresponding ς and returns $(\varsigma^e \bmod n)$ to $\mathcal{A}$;

(c) else if $m = H_1(m_i)$ and $H_1^2(m_i) = \perp$ for some i, then $\mathcal{S}$ selects a random $\varsigma \in \mathbb{Z}_n$, returns $(\varsigma^e \bmod n)$ to $\mathcal{A}$, and then fills the record $(m_i, H_1(m_i), \perp)$ as $(m_i, H_1(m_i), \varsigma)$ in $\mathcal{L}_{H_1}$;

(d) otherwise, $\mathcal{S}$ selects a random $\rho \in \mathbb{Z}_n$, records $(m, \rho, \perp)$ in $\mathcal{L}_{H_1}$, and returns ρ to $\mathcal{A}$.

(ii) H_2 Query of $\mathcal{O}_{H_2}$

When $\mathcal{A}$ asks for H_2 query by sending D to $\mathcal{S}$, $\mathcal{S}$ will look up the list $\mathcal{L}_{H_2}$:

(a) if $D = D_i$ for some i, the corresponding τ will be retrieved and $\mathcal{S}$ will send $(\tau^e \bmod n)$ back to $\mathcal{A}$;

(b) otherwise, $\mathcal{S}$ will select a random $\tau \in \mathbb{Z}_n$, record (D, τ) in $\mathcal{L}_{H_2}$, and return $(\tau^e \bmod n)$ back to $\mathcal{A}$.

(iii) H_3 Query of $\mathcal{O}_{H_3}$

While $\mathcal{A}$ sends (σ, D) to $\mathcal{S}$ for $H_3(\sigma \parallel D)$, $\mathcal{S}$ will look up the list $\mathcal{L}_{H_3}$:

(a) if $(\sigma, D) = (\sigma_i, D_i)$ for some i, the corresponding $H_3(\sigma_i \parallel D_i)$ will be retrieved and returned to $\mathcal{A}$;

(b) otherwise, $\mathcal{S}$ will select a random $\eta \in \mathbb{Z}_n$, set $H_3(\sigma \parallel D) = (\eta^e y \bmod n)$, record $((\sigma, D), \eta, H_3(\sigma \parallel D))$ in $\mathcal{L}_{H_3}$, and return $H_3(\sigma \parallel D)$ back to $\mathcal{A}$.

(iv) E-Cash Producing Query of $\mathcal{O}_{\mathcal{S}}$

Let ℓ_{D_i} be a counter to record the number of queries on each expiration date D_i, which is initialized by 0. When $\mathcal{A}$ sends (α, ϵ, D) to $\mathcal{S}$, $\mathcal{S}$ will do the following steps:

(1) decrypt ϵ, obtain (k, ID);

(2) randomly select r_j and prepare $\sigma = \widehat{E}_{pk_j}(\text{ID} \parallel r_j)$;

(3) choose $\eta \in_R \mathbb{Z}_n$, set $H_3(\sigma \parallel D) = (\alpha\eta^e \bmod n)$, and store $((\sigma, D), \perp, (\alpha\eta^e \bmod n))$ and (σ, D) in $\mathcal{L}_{H_3}$ and $\mathcal{L}_x$, respectively;

(4) select $b \in_R \mathbb{Z}_n^*$ and compute $\beta = (b^e \alpha \eta^e)^{-1} \bmod n$;

(5) retrieve or assign τ such that $H_2(D) = (\tau^e)$ as the $\mathcal{O}_{H_2}$ query described above;

(6) compute $t \equiv (\alpha\beta\tau^e)^d \equiv ((b\eta)^{-1}\tau) \pmod n$;

(7) set $\ell_D = \ell_D + 1$ and return $(t, \widetilde{E}_k(b, \sigma, r_j))$ back to $\mathcal{A}$.

Eventually, assume that $\mathcal{A}$ can successfully output $\ell_{D'} + 1$ e-cash tuples for some expiration date D'

$$\left\{\left(s_1, m_1, \sigma_1, D'\right) \cdots \left(s_{\ell_{D'}+1}, m_{\ell_{D'}+1}, \sigma_{\ell_{D'}+1}, D'\right)\right\} \quad (20)$$

such that $s_i^e H_1^2(m_i) H_3(\sigma_i \parallel D') = H_2(D') \pmod n$, $\forall i$, $1 \le i \le \ell_{D'} + 1$, after $\ell_{D'}$ times to query $\mathcal{O}_{\mathcal{S}}$ on D', with nonnegligible probability $\epsilon_{\mathcal{A}}$.

Assume some (σ_i, D'), $1 \le i \le \ell_{D'} + 1$, is not recorded in $\mathcal{L}_x$; then by the $\mathcal{L}_{H_1}$, $\mathcal{L}_{H_2}$, and $\mathcal{L}_{H_3}$, $\mathcal{S}$ can compute and retrieve

$$\begin{aligned}(s_i)^e &\equiv \left(H_1^2(m_i) H_3\left(\sigma_i \parallel D'\right)\right)^{-1} H_2\left(D'\right) \\ &\equiv \left(\left(\varsigma_i^e\right)\left(\eta_i^e y\right)\right)^{-1}\left(\tau_i^e\right) \pmod n, \\ x &\equiv y^d \equiv \left(s_i \varsigma_i \eta_i\right)^{-1} \tau_i \pmod n\end{aligned} \quad (21)$$

and solve the RSA inversion problem with nonnegligible probability at least $\epsilon_{\mathcal{A}}$. □

4.3. E-Cash Conditional-Traceability. In this section, we will prove that the ID information embedded in e-cash(s) cannot be replaced or moved out by any user against being traced after some misbehavior or criminals. The details of our proof model are illustrated in Figure 8.

Definition 10 (Tampering Game (TG)). Let $l_k \in \mathbb{N}$ be a security parameter and $\mathcal{A}$ be an adversary in $\mathcal{DAOECCS}$. $\mathcal{O}_{\mathcal{S}}$ is an oracle which plays the role of bank in $\mathcal{DAOECCS}$

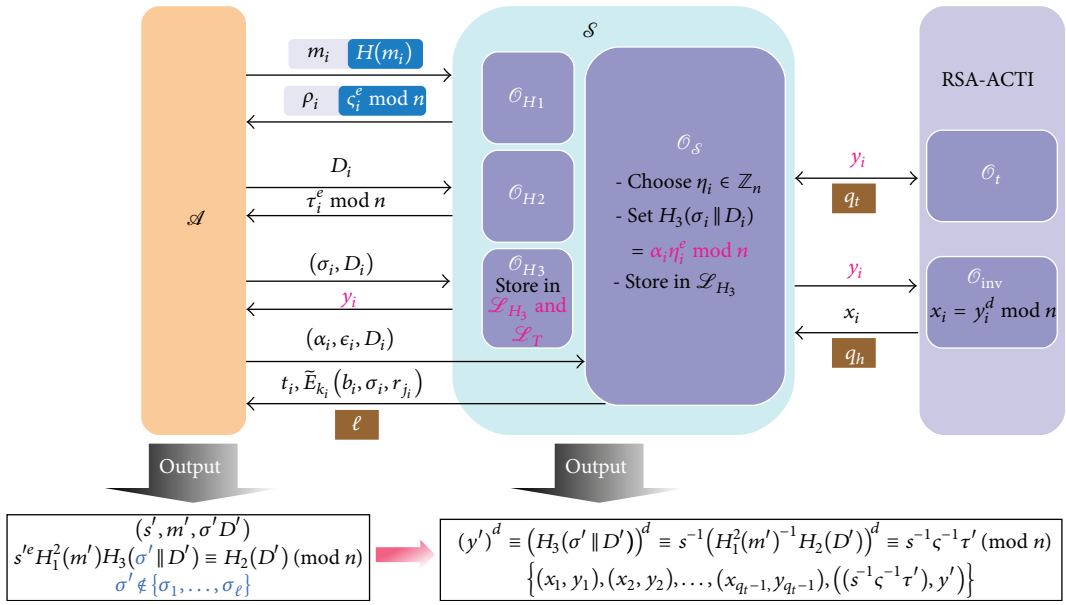

Figure 8: The proof model of TG.

Experiment $\text{Exp}_{\mathcal{A}}^{\text{TG}}(l_k)$
$(pk_j, sk_j, g_1, g_2, e_b, d_b, p_b, q_b, n_b, H_1, H_2, H_3, H_4, H_5) \leftarrow \text{Setup}(l_k)$
$(s', m', \sigma', D') \leftarrow \mathcal{A}^{\mathcal{O}_{\mathcal{S}}}(pk_{TA}, e_R, n_R, H_1, H_2)$
$\{\sigma_1, \ldots, \sigma_\ell\} \leftarrow \mathcal{O}_{\mathcal{S}}$
if the following two checks are true, **return 1;**
(i) $\sigma' \notin \{\sigma_1, \ldots, \sigma_\ell\}$
(ii) $s'^e H_1^2(m') H_3(\sigma' \| D') = H_2(D') \bmod n$
else **return 0;**

Algorithm 4

to record parameters from the queries of $\mathcal{A}$ and issue e-cash(s) (i.e., (s, m, σ, D), where $m = (w_1, y_1, w_2, y_2, r_3, D)$) accordingly. $\mathcal{A}$ is allowed to query $\mathcal{O}_{\mathcal{S}}$ for ℓ times; consider Algorithm 4.

$\mathcal{A}$ wins the game if the probability $\Pr[\text{Exp}_{\mathcal{A}}^{\text{TG}}(k) = 1]$ of $\mathcal{A}$ is nonnegligible.

Definition 11 (E-Cash Traceability). If there exists no probabilistic polynomial-time adversary who can win the tracing game TG, then $\mathcal{DAOECCS}$ satisfies the E-Cash Traceability.

Definition 12 (Alternative Formulation of RSA Known-Target Inversion Problem (RSA-AKTI)). Let $k \in \mathbb{N}$ be a security parameter and $\mathcal{A}$ be an adversary who is allowed to access the RSA-inversion oracle $\mathcal{O}_{\text{inv}}$ and the target oracle $\mathcal{O}_t$. $\mathcal{A}$ is allowed to query $\mathcal{O}_t$ and $\mathcal{O}_{\text{inv}}$ for q_t and q_h times ($q_h < q_t$), respectively. Consider Algorithm 5.

We say $\mathcal{A}$ breaks the RSA-AKTI problem if the probability $\Pr[\text{Exp}_{\mathcal{A}}^{\text{RSA-AKTI}}(k) = 1]$ of $\mathcal{A}$ is nonnegligible.

Theorem 13. *For a polynomial-time adversary $\mathcal{A}$ who can win the tracing game TG with nonnegligible probability, there exists another adversary $\mathcal{S}$ who can break the RSA-AKTI problem with nonnegligible probability.*

Experiment $\text{Exp}_{\mathcal{A}}^{\text{RSA-AKTI}}(k)$
$(N, e, d) \xleftarrow{R} KeyGen(k)$.
$(y_1, \ldots, y_{q_t}) \leftarrow \mathcal{O}_t(N, e, k)$
$\{(x_1, y_1), \ldots, (x_{q_t}, y_{q_t})\} \leftarrow \mathcal{A}^{\mathcal{O}_{\text{inv}}, \mathcal{O}_t}(N, e, k)$
if $x_i^e \equiv y_i \pmod N, \forall i \in \{1, \ldots, q_t\}$, **return 1;**
else **return 0;**

Algorithm 5

Proof. $\mathcal{S}$ simulates the environment of $\mathcal{DAOECCS}$ by controlling three hash oracles, $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$, $\mathcal{O}_{H_3}$, to respond hash queries and an e-cash producing oracle $\mathcal{O}_S$ of $\mathcal{DAOECCS}$ to respond e-cash producing queries from $\mathcal{A}$, respectively, in the random oracle model. Eventually, $\mathcal{S}$ will take advantage of $\mathcal{A}$'s capability to solve RSA-AKTI problem. Then, for consistency, $\mathcal{S}$ maintains three lists $\mathcal{L}_{H_1}$, $\mathcal{L}_{H_2}$, and $\mathcal{L}_{H_3}$ to record every response of $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$, and $\mathcal{O}_{H_3}$, respectively.

Besides, in the proof model, $\mathcal{S}$ is allowed to query the oracles $\mathcal{O}_{\text{inv}}$ (i.e., $(\cdot)^d$) and $\mathcal{O}_t$ of the RSA-AKTI problem defined in Definition 12 for helping $\mathcal{S}$ produce valid e-cash(s) and the corresponding verifying key is (e, n).

Here we will do the simulation for game TG to prove that $\mathcal{DAOECCS}$ satisfies the e-cash traceability. Details are described as follows.

(i) H_1 Query of $\mathcal{O}_{H_1}$

Initially, every blank record in $\mathcal{L}_{H_1}$ can be represented as $(\perp, \perp, \perp)$. When $\mathcal{A}$ sends m for querying the hash value $H_1(m)$, $\mathcal{S}$ will check the list $\mathcal{L}_{H_1}$:

(a) if $m = m_i$ for some i, then $\mathcal{S}$ retrieves the corresponding $H_1(m_i)$ and return it to $\mathcal{A}$;

(b) else if $m = H_1(m_i)$ and $H_1^2(m_i) \neq \perp$ for some i, then $\mathcal{S}$ retrieves the corresponding ς_i and returns $(\varsigma_i^e \bmod n)$ to $\mathcal{A}$;

(c) else if $m = H_1(m_i)$ and $H_1^2(m_i) = \perp$ for some i, then $\mathcal{S}$ chooses $\varsigma \in_R \mathbb{Z}_n$, sets $H_1^2(m_i) = (\varsigma^e \bmod n)$, and returns $H_1^2(m_i)$ to $\mathcal{A}$ then fills the original record $(m_i, H_1(m_i), \perp)$ as $(m_i, H_1(m_i), \varsigma)$ in $\mathcal{L}_{H_1}$;

(d) otherwise, $\mathcal{S}$ selects a random $\rho \in \mathbb{Z}_n$, sets $H_1(m_i) = \rho$, records $(m, H_1(m_i), \perp)$ in $\mathcal{L}_{H_1}$, and returns ρ to $\mathcal{A}$.

(ii) H_2 Query of $\mathcal{O}_{H_2}$

When $\mathcal{A}$ asks for H_2 query by sending D to $\mathcal{S}$, $\mathcal{S}$ will look up the list $\mathcal{L}_{H_2}$:

(a) if $D = D_i$ for some i, the corresponding τ will be retrieved and $\mathcal{S}$ will send $(\tau^e \bmod n)$ back to $\mathcal{A}$;

(b) otherwise, $\mathcal{S}$ will select a random $\tau \in \mathbb{Z}_n$, record (D, τ) in $\mathcal{L}_{H_2}$, and return $(\tau^e \bmod n)$ back to $\mathcal{A}$.

(iii) H_3 Query of $\mathcal{O}_{H_3}$

While $\mathcal{A}$ sends (σ, D) to $\mathcal{S}$ for $H_3(\sigma)$, $\mathcal{S}$ will look up the list $\mathcal{L}_{H_3}$:

(a) if $(\sigma, D) = (\sigma_i, D_i)$ for some i, the corresponding y_i will be retrieved and returned to $\mathcal{A}$;

(b) otherwise, $\mathcal{S}$ will query $\mathcal{O}_t$ to get an instance y; record y and $((\sigma, D), y)$ in $\mathcal{L}_T$ and $\mathcal{L}_{H_3}$, respectively;

(c) return y back to $\mathcal{A}$.

(iv) E-Cash Producing Query of $\mathcal{O}_{\mathcal{S}}$

While $\mathcal{A}$ sends (α, ϵ, D) to $\mathcal{S}$, $\mathcal{S}$ will do the following steps:

(1) decrypt ϵ, obtain (k, ID);

(2) randomly select r_j and prepare $\sigma = \widehat{E}_{pk_j}(\text{ID} \parallel r_j)$;

(3) choose $\eta \in_R \mathbb{Z}_n$, set $H_3(\sigma \parallel D) = (\alpha\eta^e \bmod n)$, and store $((\sigma, D), H_3(\sigma \parallel D))$ in $\mathcal{L}_{H_3}$;

(4) select $b \in_R \mathbb{Z}_n^*$ and compute $\beta = (b^e \alpha \eta^e)^{-1} \bmod n$;

(5) retrieve or assign τ such that $H_2(D) = (\tau^e)$ as the $\mathcal{O}_{H_2}$ query described above;

(6) compute $t \equiv (\alpha\beta\tau^e)^d \equiv ((b\eta)^{-1}\tau) \pmod n$;

(7) return $(t, \widetilde{E}_k(b, \sigma, r_j))$ back to $\mathcal{A}$.

Assume that $\mathcal{A}$ can successfully output an e-cash tuples (s', m', σ', D'), where σ' never appeals as a part for some $\mathcal{O}_{\mathcal{S}}$ query such that $s'^e H_1^2(m') H_3(\sigma' \parallel D') \equiv H_2(D') \pmod n$; then by $\mathcal{L}_{H_1}$, $\mathcal{L}_{H_2}$, and $\mathcal{L}_{H_3}$, $\mathcal{S}$ can derive

$$\begin{aligned}\left(y'\right)^d &\equiv \left(H_3\left(\sigma' \parallel D'\right)\right)^d \equiv s'^{-1}\left(H_1^2\left(m'\right)^{-1} H_2\left(D'\right)\right)^d \\ &\equiv s'^{-1}\varsigma'^{-1}\tau' \pmod n.\end{aligned} \tag{22}$$

Let $|\mathcal{L}_T| = q_t$ and $\mathcal{L}_T = \{y_1, \ldots, y_{q_t}\}$. $\mathcal{S}$ sends $y_i \in (\mathcal{L}_T - \{y'\})$, $1 \leq i \leq (q_t - 1)$, to $\mathcal{O}_{\text{inv}}$ and obtains $q_t - 1$ x_i such that $x_i = y_i^d \bmod n$.

Eventually $\mathcal{S}$ can output q_t RSA-inversion instances

$$\left\{\left(x_1, y_1\right), \left(x_2, y_2\right), \ldots, \left(x_{q_t-1}, y_{q_t-1}\right), \left(\left(s'^{-1}\varsigma'^{-1}\tau'\right), y'\right)\right\} \tag{23}$$

after querying $\mathcal{O}_{\text{inv}}$ for q_h times, where $q_h = q_t - 1 < q_t$ and thus, it breaks the RSA-AKTI problem with nonnegligible probability at least $\epsilon_{\mathcal{A}}$. □

4.4. E-Cash No-Swindling. In typical online e-cash transactions, when an e-cash has been spent in previous transactions, another spending will be detected immediately owing to the double-spending check procedure. However, in an offline e-cash model, the merchant may accept a transaction involving a double-spent e-cash first and then do the double-spending check later. In this case, the original owner of the e-cash may suffer from loss. Therefore, a secure offline e-cash scheme should guarantee the following two events.

(i) No one, except the real owner, can spend a fresh and valid offline e-cash successfully.

(ii) No one can double spend an e-cash successfully.

Roughly, it can be referred to as *e-cash no-swindling* property. In this section, we will define the no-swindling property and formally prove that our scheme is secure against swindling attacks.

Definition 14 (Swindling Game in $\mathcal{DAOECCS}$). Let $l_k \in \mathbb{N}$ be a security parameter and $\mathcal{A}$ be an adversary in $\mathcal{DAOECCS}$. $\mathcal{O}_B$ is an oracle issuing generic e-cash(s) (i.e., $(s, y_1, w_1, x_2, r_2, r_3, \sigma, D)$) of $\mathcal{DAOECCS}$ to $\mathcal{A}$. $\mathcal{O}_{\text{off}}$ is an oracle to show the expanding form $(s, y_1, w_1, x_2, r_2, r_3, \sigma, D, r_s, s')$ for the payment according to the input (s, m, σ, D). Consider the two experiments SWG-1 and SWG-2 shown in Algorithms 6 and 7, respectively.

$\mathcal{A}$ wins the game if the probability $\Pr[\text{Exp}_{\mathcal{A}}^{\text{SWG-1}}(l_k) = 1]$ or $\Pr[\text{Exp}_{\mathcal{A}}^{\text{SWG-2}}(l_k) = 1]$ of $\mathcal{A}$ is nonnegligible.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{SWG-1}}(l_k)$
$\left(pk_j, sk_j, g_1, g_2, e_b, d_b, p_b, q_b, n_b, p, q, H_1, H_2, H_3, H_4, H_5\right) \leftarrow \text{Setup}\,(l_k)$
$\{(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_u, r_s, s')\} \leftarrow \mathcal{A}^{\mathcal{O}_B, \mathcal{O}_{\text{off}}}\left(pk_j, g_1, g_2, e_b, n_b, p, q, H_1, H_2, H_3, H_4, H_5\right)$
if the following checks are true, **return 1;**
(i) $s^{e_b} H_1^2(y^{H_4(r_u \| r_s)} g^{s'} \bmod p \parallel y_1 \parallel w_2 \parallel y_2 \parallel D \parallel r_3) H_3(\sigma \parallel D) = H_2(D) \bmod n_b$;
(ii) $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D)$ never be a query to $\mathcal{O}_{\text{off}}$
else **return 0;**

Algorithm 6: Experiment SWG-1.

Experiment $\mathbf{Exp}_{\mathcal{A}}^{\text{SWG-2}}(l_k)$
$(pk_j, sk_j, g_1, g_2, e_b, d_b, p_b, q_b, n_b, p, q, H_1, H_2, H_3, H_4, H_5) \leftarrow \text{Setup}(l_k)$
$\{(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_u, r_s, s')\} \leftarrow \mathcal{A}^{\mathcal{O}_B, \mathcal{O}_{\text{off}}}\left(pk_j, g_1, g_2, e_b, n_b, p, q, H_1, H_2, H_3, H_4, H_5\right)$
if the following checks are true, **return 1;**
(i) $s^{e_b} H_1^2(y^{H_4(r_u \| r_s)} g^{s'} \bmod p \parallel y_1 \parallel w_2 \parallel y_2 \parallel D \parallel r_3) H_3(\sigma \parallel D) = H_2(D) \bmod n_b$;
(ii) $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D)$ is allowed to be queried to $\mathcal{O}_{\text{off}}$ for once;
(iii) $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_s, s')$ is not obtained from $\mathcal{O}_{\text{off}}$
else **return 0;**

Algorithm 7: Experiment SWG-2.

Definition 15 (E-Cash No-Swindling). If there exists no probabilistic polynomial-time adversary who can win the swindling game defined in Definition 14, then $\mathcal{DAOECS}$ satisfies e-cash no-swindling.

Theorem 16. *For a polynomial-time adversary $\mathcal{A}$ who can win the swindling game SWG with nonnegligible probability, there exists another adversary $\mathcal{S}$ who can solve the discrete logarithm problem with nonnegligible probability.*

Proof. Consider the swindling game defined in Definition 14. $\mathcal{S}$ simulates the environment by controlling the hash oracles, $\mathcal{O}_{H_4}$, to respond hash queries on H_4 of $\mathcal{DAOECS}$ in the random oracle model. Eventually, $\mathcal{S}$ will take advantage of $\mathcal{A}$'s capability to solve the discrete logarithm problem. Then, for consistency, $\mathcal{S}$ maintains a list $\mathcal{L}_{H_4}$ to record every response of $\mathcal{O}_{H_4}$. $\mathcal{S}$ is given all parameters $(pk_j, sk_j, g_1, g_2, e_b, d_b, p_b, q_b, n_b, p, q, H_1, H_2, H_3, H_4, H_5)$ of $\mathcal{DAOECS}$ and an instance y^* of discrete logarithm problem (i.e., $y^* = g^{x^*} \bmod p$). Here we will describe the simulations for the two experiments $\text{Exp}_{\mathcal{A}}^{\text{SWG-1}}$ and $\text{Exp}_{\mathcal{A}}^{\text{SWG-2}}$, individually.

The simulation for $\text{Exp}_{\mathcal{A}}^{\text{SWG-1}}$ is illustrated in Figure 9 and each oracle is constructed as follows.

(i) Oracle $\mathcal{O}_B$

Initially, $\mathcal{S}$ guesses that the generic e-cash produced from νth query will be the attack target. When $\mathcal{A}$ sends ith query to $\mathcal{O}_B$ for an e-cash, $\mathcal{O}_B$ will do the following:

(a) select $r_1, x_1, r_3 \in_R \mathbb{Z}_q$ and $y_2, w_2 \in_R \mathbb{Z}_p$;
(b) if $i = \nu$,
(1) compute $(w_1 = (y^*)^{r_1} \bmod p)$ and $(y_1 = g^{x_1} \bmod p)$;
(c) if $i \neq \nu$,
(1) compute $(w_1 = g^{r_1} \bmod p)$ and $(y_1 = g^{x_1} \bmod p)$;
(d) prepare $s = ((H_1^2(m) H_3(\sigma \parallel D))^{-1} H_2(D))^{d_b} \bmod n_b$, where $m = (w_1, y_1, w_2, y_2, r_3, D)$;
(e) record $(i, (s, m, \sigma, D), (r_1, x_1)))$ in list $\mathcal{L}_B$ and return (s, m, σ, D) to $\mathcal{A}$.

(ii) Oracle $\mathcal{O}_{\text{off}}$

When $\mathcal{A}$ sends a valid e-cash tuple $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_s)$ to $\mathcal{O}_{\text{off}}$, it will look up the list $\mathcal{L}_B$:

(a) if $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D)$ exists with prefix index ν, then abort;
(b) otherwise, $\mathcal{O}_{\text{off}}$ will retrieve the corresponding (r_1, x_1); choose a random r_u, compute $u = H_4(r_u \parallel r_s)$ and $(s' = r_1 - u x_1 \bmod q)$, and send $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_u, r_s, s')$ back to $\mathcal{A}$.

Assume that $\mathcal{A}$ can successfully output a valid offline e-cash expansion tuple $(s^*, w_1^*, y_1^*, w_2^*, y_2^*, r_3^*, \sigma^*, D^*, r_u^*, r_s^*, s'^*)$, where $(s^*, w_1^*, y_1^*, w_2^*, y_2^*, r_3^*, \sigma^*, D^*)$ is prefixed with ν and postfixed with (r_1^*, x_1^*) in $\mathcal{L}_B$. Then, since $w_1^* = y_1^{*H_4(r_u^* \| r_s^*)} g^{s'^*} \bmod p$ and $w_1^* = (y^*)^{r_1^*}$, $\mathcal{S}$ can derive

$$x^* = (r_1^*)^{-1}\left(x_1^* H_4\left(r_u^* \parallel r_s^*\right) + s'^*\right) \bmod q \tag{24}$$

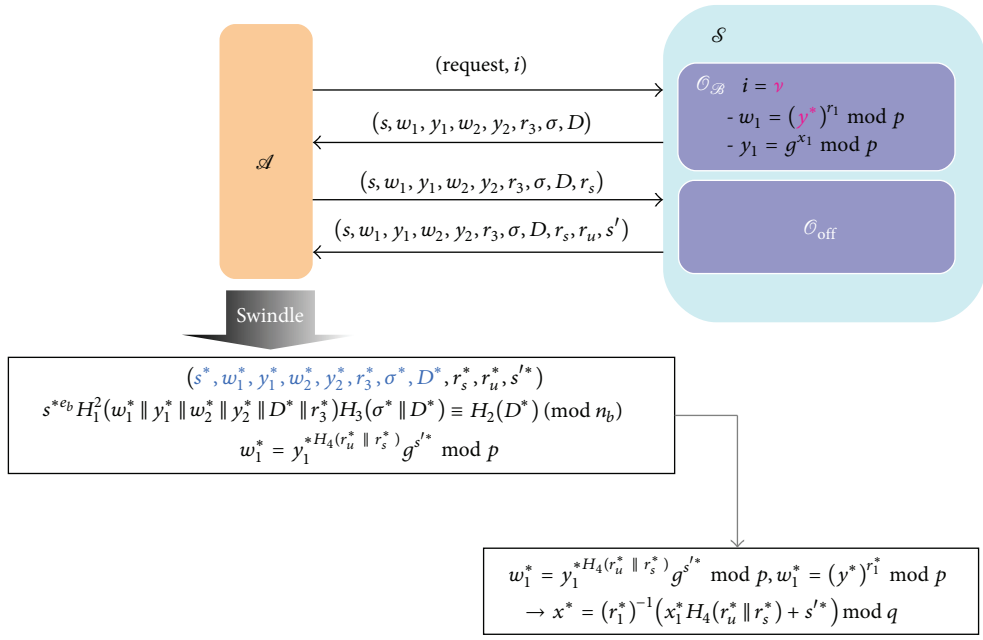

Figure 9: The proof model of SWG-1.

and solve the discrete logarithm problem with nonnegligible probability at least $(1/q_{\mathcal{O}_B})\epsilon_{\mathcal{A}}$, where $q_{\mathcal{O}_B}$ is the total number of $\mathcal{O}_B$ query.

The simulation for $\text{Exp}_{\mathcal{A}}^{\text{SWG-2}}$ is illustrated in Figure 10 and each oracle is constructed as follows.

(i) Oracle $\mathcal{O}_B$

Initially, $\mathcal{S}$ guesses that the generic e-cash produced from νth query will be the attack target. When $\mathcal{A}$ sends ith query to $\mathcal{O}_B$ for an e-cash, $\mathcal{O}_B$ will do the followings.

(a) if $i = \nu$:

(1) select $s', u, x_1, r_3 \in_R \mathbb{Z}_q$ and $y_2, w_2 \in_R \mathbb{Z}_p$;
(2) compute $(y_1 = (y^*)^{x_1} \bmod p)$ and $(w_1 = y_1^u g^{s'} \bmod p)$;
(3) prepare $s = ((H_1^2(m)H_3(\sigma \| D))^{-1} H_2(D))^{d_b} \bmod n_b$, where $m = (w_1, y_1, w_2, y_2, r_3, D)$;
(4) record $(i, (s, m, \sigma, D), (u, s'))$ in list $\mathcal{L}_{\mathcal{B}}$;

(b) if $i \neq \nu$:

(1) select $r_1, x_1, r_3 \in_R \mathbb{Z}_q$ and $y_2, w_2 \in_R \mathbb{Z}_p$;
(2) compute $(w_1 = g^{r_1} \bmod p)$ and $(y_1 = g^{x_1} \bmod p)$;
(3) prepare $s = ((H_1^2(m)H_3(\sigma \| D))^{-1} H_2(D))^{d_b} \bmod n_b$, where $m = (w_1, y_1, w_2, y_2, r_3, D)$;
(4) record $(i, (s, m, \sigma, D), (r_1, x_1))$ in list $\mathcal{L}_{\mathcal{B}}$;

(c) return (s, m, σ, D) to $\mathcal{A}$.

(ii) Oracle $\mathcal{O}_{\text{off}}$

A status parameter sta is initialized by 0. When $\mathcal{A}$ sends a valid e-cash tuple $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_s)$ to $\mathcal{O}_{\text{off}}$, it will look up the list $\mathcal{L}_{\mathcal{B}}$:

(a) if $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D)$ exists with prefix index ν and sta $= 0$, $\mathcal{O}_{\text{off}}$ will perform the following procedures:

(1) set sta $= 1$
(2) retrieve the corresponding (u, s') from $\mathcal{L}_{\mathcal{B}}$ and choose a random r_u;
(3) set $H_4(r_u \| r_s) = u$ and record $((r_u \| r_s), u)$ in $\mathcal{L}_H$;
(4) record $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_u, r_s, s')$ in list $\mathcal{L}_{\text{off}}$;
(5) send $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_u, r_s, s')$ back to $\mathcal{A}$;

(b) if $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D)$ exists with prefix index $\neq \nu$, $\mathcal{O}_{\text{off}}$ will retrieve the corresponding (r_1, x_1), choose random r_u and u, set $H_4(r_u \| r_s) = u$, record $((r_u \| r_s), u)$ in $\mathcal{L}_H$, compute $(s' = r_1 - ux_1 \bmod q)$, and send $(s, w_1, y_1, w_2, y_2, r_3, \sigma, D, r_u, r_s, s')$ back to $\mathcal{A}$.

(c) Otherwise, abort.

(iii) Oracle $\mathcal{O}_{H_4}$

While $\mathcal{A}$ sends $(r_u \| r_s)$ to query for $H_4(r_u \| r_s)$, $\mathcal{O}_{H_4}$ will check the list $\mathcal{L}_H$:

(a) if $(r_u \| r_s)$ exists as the prefix of some record, $\mathcal{O}_{H_4}$ will retrieve the corresponding u and return it to $\mathcal{A}$;

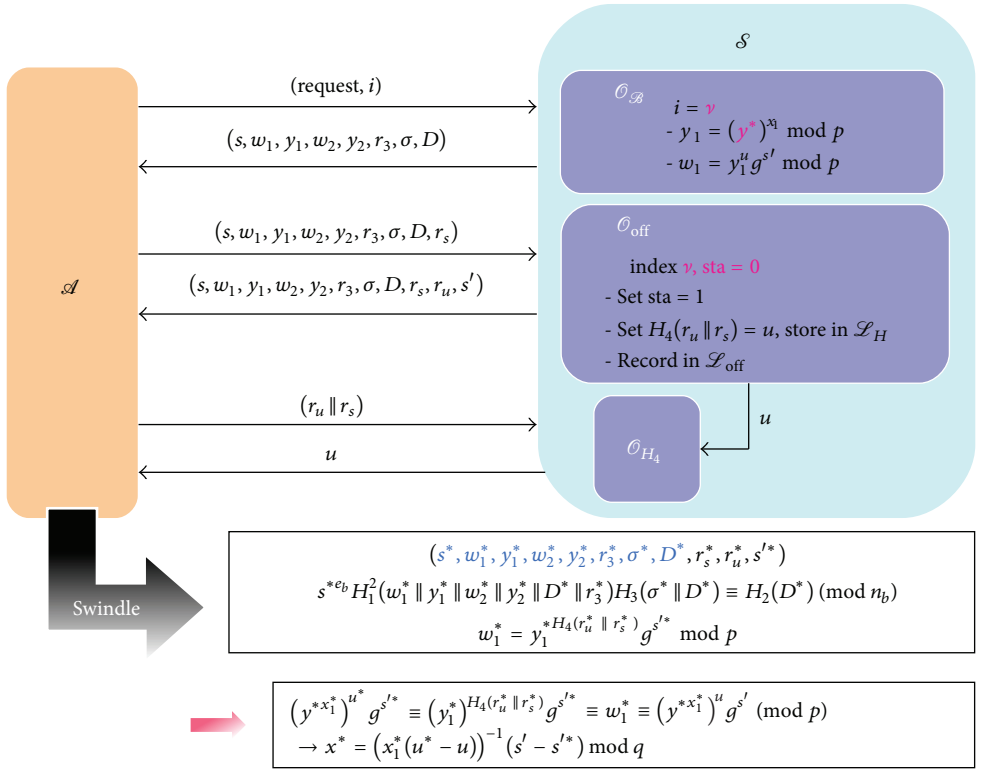

FIGURE 10: The proof model of SWG-2.

(b) otherwise, $\mathcal{O}_{H_4}$ will choose a random u, record $((r_u \| r_s), u)$ in $\mathcal{L}_H$, and return u to $\mathcal{A}$.

Assume that $\mathcal{A}$ can successfully output a valid offline e-cash expansion tuple $(s^*, w_1^*, y_1^*, w_2^*, y_2^*, r_3^*, \sigma^*, D^*, r_u^*, r_s^*, s'^*)$, where $(s^*, w_1^*, y_1^*, w_2^*, y_2^*, r_3^*, \sigma^*, D^*)$ is prefixed with ν and postfixed with (u, s') in $\mathcal{L}_{\mathcal{B}}$ and $H_4(r_u^* \| r_s^*) \neq u$.

Then, via $\mathcal{L}_{\mathcal{H}}$, since

$$\begin{aligned}\left(y^{*x_1^*}\right)^{u^*} g^{s'^*} &\equiv \left(y_1^*\right)^{H_4(r_u^* \| r_s^*)} g^{s'^*} \equiv w_1^* \\ &\equiv \left(y^{*x_1^*}\right)^u g^{s'} \pmod p,\end{aligned} \tag{25}$$

$\mathcal{S}$ can derive

$$x^* = \left(x_1^* \left(u^* - u\right)\right)^{-1} \left(s' - s'^*\right) \bmod q \tag{26}$$

and solve the discrete logarithm problem with nonnegligible probability at least $(1/q_{\mathcal{O}_B})\epsilon_{\mathcal{A}}$, where $q_{\mathcal{O}_B}$ is the total number of $\mathcal{O}_B$ query. □

Summarize the proof models for the two experiments shown above, if there exists a polynomial-time adversary who can win the swindling game with nonnegligible probability, then there exists another one who can solve the discrete logarithm problem with nonnegligible probability. It implies that there exists no p.p.t. adversary who can win the swindling game, and our proposed offline e-cash scheme $\mathcal{DAOECS}$ satisfies no-swindling property.

5. E-Cash Advanced Features and Performance Comparisons

In this section, we compare the e-cash features and performance of our proposed scheme with other schemes given in [9, 13–15, 21, 22, 27, 38–40]. We analyze the features and performance of the aforementioned schemes and form a table (Table 1) for the summary.

5.1. Features Comparisons. All the schemes mentioned above fulfill the basic security requirements stated in Section 1, which are anonymity, unlinkability, unforgeability, and no double-spending. Besides these features, there can be other advanced features on an e-cash system discussed in the literatures. We focus on three other advanced features, which are traceability, date attachability, and no-swindling, and we compare the proposed scheme with the aforementioned schemes.

We also propose an e-cash renewal protocol for users to exchange a new valid e-cash with their unused but expired e-cash(s); therefore, users do not have to deposit the e-cash before it expires and withdraw a new e-cash again. Our proposed e-cash renewal protocol reduces the computation cost by 49.5% as compared to withdrawal and deposit protocols, which is almost half of the effort of getting a new e-cash, at the user side. It does a great help to the users since their devices usually have a weaker computation capability, such as smart phones.

Table 1: Advanced features and performance comparisons.

	Ours	[38]	[14]	[15]	[9]	[21]	[22]	[39]	[40]	[13]	[27]
						Advanced features					
On/off-line	Off	Off	Off	Off	On	Off	Off	Off	Off	On	Off
Conditional-traceability	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No
Date attachability	Yes	No	No	No	Yes	Yes	No	No	No	No	Yes
No-swindling	Yes	No	No	No	—	No	Yes	No	No	—	No
Renewal protocol	Yes	—	Yes	—	No	Yes	Yes	—	—	—	Yes
Formal proof	Yes	Yes	No	Yes	No	No	Yes	Yes	Yes	Yes	No
						Performance					
Transaction cost*	$5E + 7M$ $+7H+1\text{inv}$ $+1A$ $\approx 1454M$	$14E + 14M$ $+1H + 5A$ $\approx 3375M$	$6E + 8M$ $\approx 1448M$	$23E + 14M$ $+1A$ $\approx 5534M$	$2E + 2M$ $+2H$ $\approx 966M$	$5E + 9M$ $+1H + 1\text{inv}$ $+2A$ $\approx 1450M$	$2E \approx 480M$	$18E + 15M$ $+2H + 8A$ $\approx 4337M$	$31E + 22M$ $+6H + 10A$ $\approx 7468M$	$22E + 11M$ $+4A$ $\approx 5291M$	$6E + 8M$ $+1H$ $\approx 1449M$
Communication cost◇	1092	576	1288	939	769	644	300	828	968	1536	728

According to [41], $H \approx M$, $E \approx \text{inv} \approx 240M$.

E: a modular exponentiation; M: a modular multiplication; H: a hash operation; zkp: a zero-knowledge proof.

A: a modular addition; inv: a modular inversion.

*The computation cost of withdrawal and payment protocols at user side.

◇The communication cost of each transaction at user side in bytes.

5.2. Performance Comparisons. According to [41], we can summarize and induce the computation cost of all operations as follows. The computation cost of a modular exponentiation computation is about 240 times of the computation cost of a modular multiplication computation, while the computation cost of a modular inversion almost equals to that of a modular exponentiation. Also, the computation cost of a hash operation almost equals to that of a modular multiplication.

With the above assumptions, the total computation cost of users during withdrawal and payment phases of our proposed scheme can be induced as 1452 times of a modular multiplication computation, while other works [9, 13–15, 21, 22, 27, 38–40] need 3375, 1448, 5534, 966, 1450, 480, 4337, 7468, 5291, and 1449 times of a modular multiplication computation to finish withdrawal and payment phases at the user ends.

According to [15], we assume the RSA parameters n, p, q are 1024, 512, and 512 bits, respectively. We adopt AES and SHA-1 as the symmetric cryotsystem and one-way hash function used in all protocols, respectively; therefore, the signed message and hash massage are in 128 and 160 bits, respectively. We assume the expiration date is in 32 bits.

With the above assumptions, we compute the communication cost of each offline transaction, withdrawal, and payment, at the user side. Our scheme needs 2048 bits for withdrawing an e-cash and 6688 bits for spending an e-cash, which is 1092 bytes for each transaction.

The details of the comparisons are summarized in Table 1.

6. Conclusion

In this paper, we have presented earlier a provably secure offline electronic cash scheme with an expiration date and a deposit date attached to it. Besides, we have also designed an e-cash renewal protocol, where users can exchange their unused and expired e-cash(s) for new ones more efficiently. Compared with other similar works, our scheme is efficient from the aspect of considering computation cost of the user side and satisfying all security properties, simultaneously. Except for anonymity, unlinkability, unforgeability, and no double-spending, we also formally prove that our scheme achieves conditional-traceability and no-swindling. Not only does our scheme help the bank to manage their huge databases against unlimited growth, but also it strengthens the preservation of users' privacy and rights as well.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was partially supported by the National Science Council of Taiwan under Grants NSC 102-2219-E-110-002, NSYSU-KMU Joint Research Project (NSYSUKMU 2013-I001), and Aim for the Top University Plan of the National Sun Yat-sen University and Ministry of Education, Taiwan.

References

[1] H. Chen, P. P. Y. Lam, H. C. B. Chan, T. S. Dillon, J. Cao, and R. S. T. Lee, "Business-to-consumer mobile agent-based internet commerce system (MAGICS)," *IEEE Transactions on Systems, Man and Cybernetics C: Applications and Reviews*, vol. 37, no. 6, pp. 1174–1189, 2007.

[2] S. C. Fan and Y. L. Lai, "A study on e-commerce applying in Taiwan's restaurant franchise," in *Proceedings of the IET International Conference on Frontier Computing. Theory, Technologies and Applications*, pp. 324–329, August 2010.

[3] D. R. W. Holton, I. Nafea, M. Younas, and I. Awan, "A class-based scheme for E-commerce web servers: formal specification and performance evaluation," *Journal of Network and Computer Applications*, vol. 32, no. 2, pp. 455–460, 2009.

[4] Z. Jie and X. Hong, "E-commerce security policy analysis," in *Proceedings of the International Conference on Electrical and Control Engineering (ICECE '10)*, pp. 2764–2766, June 2010.

[5] D. R. Liuy and T. F. Hwang, "An agent-based approach to flexible commerce in intermediary-Centric electronic markets," *Journal of Network and Computer Applications*, vol. 27, no. 1, pp. 33–48, 2004.

[6] S. J. Lin and D. C. Liu, "An incentive-based electronic payment scheme for digital content transactions over the Internet," *Journal of Network and Computer Applications*, vol. 32, no. 3, pp. 589–598, 2009.

[7] H. Wang, Y. Zhang, J. Cao, and V. Varadharajan, "Achieving Secure and Flexible M-Services through Tickets," *IEEE Transactions on Systems, Man, and Cybernetics A:Systems and Humans*, vol. 33, no. 6, pp. 697–708, 2003.

[8] C. Yue and H. Wang, "Profit-aware overload protection in E-commerce Web sites," *Journal of Network and Computer Applications*, vol. 32, no. 2, pp. 347–356, 2009.

[9] C. C. Chang and Y. P. Lai, "A flexible date-attachment scheme on e-cash," *Computers and Security*, vol. 22, no. 2, pp. 160–166, 2003.

[10] C. L. Chen and J. J. Liao, "A fair online payment system for digital content via subliminal channel," *Electronic Commerce Research and Applications*, vol. 10, no. 3, pp. 279–287, 2011.

[11] C. I. Fan, W. K. Chen, and Y. S. Yeh, "Date attachable electronic cash," *Computer Communications*, vol. 23, no. 4, pp. 425–428, 2000.

[12] C. I. Fan and W. Z. Sun, "Efficient encoding scheme for date attachable electronic cash," in *Proceedings of the 24th Workshop on Combinatorial Mathematics and Computation Theory*, pp. 405–410, 2007.

[13] T. Nakanishi, M. Shiota, and Y. Sugiyama, "An efficient online electronic cash with unlinkable exact payments," *Information Security*, vol. 3225, pp. 367–378, 2004.

[14] Y. Baseri, B. Takhtaei, and J. Mohajeri, "Secure untraceable off-line electronic cash system," *Scientia Iranica*, vol. 20, pp. 637–646, 2012.

[15] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Compact e-cash," in *Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology (EUROCRYPT '05)*, pp. 302–321, May 2005.

[16] J. Camenisch, S. Hohenberger, and A. Lysyanskaya, "Balancing accountability and privacy using E-cash," in *Security and Cryptography for Networks*, vol. 4116 of *Lecture Notes in Computer Science*, pp. 141–155, 2006.

[17] J. Camenisch, A. Lysyanskaya, and M. Meyerovich, "Endorsed e-cash," in *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 101–115, May 2007.

[18] S. Canard, A. Gouget, and J. Traoré, "Improvement of efficiency in (unconditional) anonymous transferable E-cash," in *Financial Cryptography and Data Security*, vol. 5143 of *Lecture Notes in Computer Science*, pp. 202–214, 2008.

[19] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Advances in Cryptology-CRYPTO '88*, vol. 403 of *Lecture Notes in Computer Science*, pp. 319–327, Springer, Berlin, Germany, 1990.

[20] G. Davida, Y. Frankel, Y. Tsiounis, and M. Yung, "Anonymity control in E-cash systems," in *Proceedings of the First International Conference on Financial Cryptography*, pp. 1–16, 1997.

[21] Z. Eslami and M. Talebi, "A new untraceable off-line electronic cash system," *Electronic Commerce Research and Applications*, vol. 10, no. 1, pp. 59–66, 2011.

[22] C. I. Fan, V. S. M. Huang, and Y. C. Yu, "User efficient recoverable off-line e-cash scheme with fast anonymity revoking," *Mathematical and Computer Modelling*, vol. 58, pp. 227–237, 2013.

[23] X. Hou and C. H. Tan, "Fair traceable off-line electronic cash in wallets with observers," in *Proceedings of the 6th International Conference on Advanced Communication Technology*, pp. 595–599, February 2004.

[24] X. Hou and C. H. Tan, "A new electronic cash model," in *Proceedings of the International Conference on Information Technology: Coding and Computing*, pp. 374–379, April 2005.

[25] W. S. Juang, "A practical anonymous off-line multi-authority payment scheme," *Electronic Commerce Research and Applications*, vol. 4, no. 3, pp. 240–249, 2005.

[26] J. K. Liu, V. K. Wei, and S. H. Wong, "Recoverable and untraceable e-cash," in *International Conference on Trends in Communications (EUROCON '01)*, vol. 1, pp. 132–135, 2001.

[27] C. Wang, H. Sun, H. Zhang, and Z. Jin, "An improved off-line electronic cash scheme," in *Proceedings of the 5th International Conference on Computational and Information Sciences (ICCIS '13)*, pp. 438–441, 2013.

[28] W. S. Juang, "D-cash: a flexible pre-paid e-cash scheme for date-attachment," *Electronic Commerce Research and Applications*, vol. 6, no. 1, pp. 74–80, 2007.

[29] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology-CRYPTO '82*, Lecture Notes in Computer Science, pp. 199–203, Springer, Berlin, Germany, 1983.

[30] H. Krawczyk and T. Rabin, "Chameleon signatures," in *Proceedings of the Network and Distributed System Security Symposium (NDSS '00)*, pp. 143–154, 2000.

[31] S. Pearson, *Trusted Computing Platforms: TCPA Technology in Context*, Prentice Hall, New York, NY, USA, 2002.

[32] S. Pearson, "Trusted computing platforms: the next security solution," Tech. Rep. HPL-2002-221, Hewllet-Packard Laboratorie, 2002.

[33] C. I. Fan and V. S. M. Huang, "Provably secure integrated on/off-line electronic cash for flexible and efficient payment," *IEEE Transactions on Systems, Man and Cybernetics C: Applications and Reviews*, vol. 40, no. 5, pp. 567–579, 2010.

[34] S. Bajikar, Trusted platform module (TPM) based security on notebook pcs—white paper, Mobile Platform Group, Intel Corporation, 2002.

[35] M. Abe and T. Okamoto, "Provably secure partially blind signatures," in *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '00)*, pp. 271–286, Springer, 2000.

[36] A. Juels, M. Luby, and R. Ostrovsky, "Security of blind digital signatures," in *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '97)*, pp. 150–164, Springer, 1997.

[37] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, "The one-more-RSA-inversion problems and the security of chaum's blind signature scheme," *Journal of Cryptology*, vol. 16, no. 3, pp. 185–215, 2003.

[38] S. Brands, "Untraceable off-line cash in wallets with observers (extended abstract)," *CRYPTO*, pp. 302–318, 1993.

[39] Y. Hanatani, Y. Komano, K. Ohta, and N. Kunihiro, "Provably secure electronic cash based on blind multisignature schemes," *Financial Cryptography*, vol. 4107, pp. 236–250, 2006.

[40] C. Popescu, "An off-line electronic cash system with revokable anonymity," in *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference*, pp. 763–767, May 2004.

[41] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, NY, USA, 1997.