Design and Implementation of an Efficient Fair Off-line E-Cash System based on Elliptic Curve Discrete Logarithm Problem

Manho Lee, Gookwhan Ahn, Jinho Kim, Jaegwan Park, Byoungcheon Lee, Kwangjo Kim, and Hyuckjae Lee

Abstract: In this paper, we design and implement an efficient fair off-line electronic cash system based on Elliptic Curve Discrete Logarithm Problem (ECDLP), in which the anonymity of coins is revocable by a trustee in case of dispute. To achieve this, we employ the Petersen and Poupard’s electronic cash system [1] and extend it by using an elliptic curve over the finite field $GF(2^m)$. This naturally reduces message size by 85% compared with the original scheme and makes a smart card to store coins easily. Furthermore, we use the Baek et al.’s provably secure public key encryption scheme [2] to improve the security of electronic cash system. As an extension, we propose a method to add atomicity into new electronic cash system. To the best of our knowledge, this is the first result to implement a fair off-line electronic cash system based on ECDLP with provable security.

Index Terms: Electronic cash, anonymity revocation, atomicity, elliptic curve discrete logarithm problem.

I. INTRODUCTION

Recently, a variety of on-line businesses are rapidly emerging over the Internet, which is believed to be one of the most efficient and convenient ways to provide all electronic services. An efficient and secure electronic cash (“e-cash” in short) system plays an important role to support these businesses safely as a trustful payment over the Internet. Real money (or paper money) using traditional means of payment has potential security problems such as counterfeiting and forgeability. E-cash also exhibits similar drawbacks, but properly-designed e-cash system can provide more secure and flexible service for non face-to-face exchange of digital goods than real money.

After Chaum [3] introduced the anonymity of an e-cash using blind signature, numerous researches have been done in the field of e-cash system. Whether the bank is required to be on-line or not in the processing of an electronic transaction, Chaum [4] suggested an anonymous on-line e-cash system and Chaum et al. [5] proposed an anonymous off-line e-cash system, which satisfies double-spending prevention with the cut-and-choose method. The e-cash systems [6], [7] by Okamoto and Ohta satisfy the divisibility and transferability in addition. Their schemes overcome some limitations of previous e-cash systems and provide more efficient features than real money. Brands [8] proposed an efficient e-cash system with single-term method which is more efficient compared with the cut-and-choose method. Brands’ scheme has been used as a basic model by other researchers.

However, Solms and Naccache [9] raised the issue of perfect crime by abusing the anonymity of the e-cash system. Other malicious actions such as blackmailing and money laundering were also presented in [1], [9], [10], and [11]. The revocable e-cash system [10], [12] (or fair payment system) in which anonymity can be revoked when needed, becomes one of the active research areas of preventing such misuses. In the revocable e-cash scheme, the identification of an illegal user can be traced by the cooperation of a trustee and a bank.

Along with countermeasures [10], [13] against the blackmailing and money laundering, many schemes in [1], [11], [12], [14], and [15] have been proposed to resist against the abuse of anonymity. Two schemes suggested by Camenish et al. [16] and Frankel et al. [17] require the trustee to take part in the initialization phase but do not provide a prevention against extortion and blindfolding attacks. Some schemes were suggested to prevent these attacks. Among them, Fujisaki and Okamoto’s scheme [14] and Jakobsson and Moti’s scheme [11] are said to be not efficient in a sense that users need to communicate with a trustee in every payment phase. On the other hand, Petersen and Poupard’s scheme [1] (called as “PePo97”) is considered to be an efficient off-line electronic payment system in which users need to communicate with the trustee in the registration phase only when registering a pseudonymous public key. This scheme is found to be secure against many kinds of attacks such as secret key extortion, framing against the user, trustee’s blind-folding, blackmailing, and money laundering.

In addition to these security requirements, atomicity is also of another importance in serving a complete payment system. If the network crashes in the middle of the withdrawal or payment protocol, some disputes such as losing e-cash, repudiation, and no procurement of goods even after a full payment will happen. Under a network model where customers purchase electronic goods and receive its service, Tygar [18] introduced the problem of atomicity in electronic transactions and defined three classes of atomicity: money atomic, goods atomic, and certified delivery. Camp et al. [19] proposed a method to support atomicity using on-line TTP (Trusted Third Party). Moreover, Sirbu and Tygar [20] and Bellare et al. [21] suggested on-line account-
based atomicity and token-based atomicity, respectively. Xu et al. [22] addressed money conservation via atomicity in fair offline e-cash.

A. Security Requirements

In general, the security requirements of e-cash system can be classified into two parts [1], [11]. Hereafter, we use “coin” and “e-cash” interchangeably for the sake of simplicity.

Basic Requirements

- Unforgeability: Only authorized bank can issue coins.
- Double-spending prevention: A coin cannot be used more than once.
- Anonymity: A bank cannot link a coin to the honest owner of the coin without trustee’s help.
- Untraceability: A bank cannot trace the relationship between a coin and the user of the coin its owner without a legal order.
- Efficiency: The system must be efficient in terms of storage, communication and computation.

Additional Requirements

- Revocability: Any coin which is legally obtained can be revealed.
- User-tracing: A bank can legally trace the user of a paid coin with trustee’s help.
- Extortion-tracing: A bank can legally trace with the matching information of a paid or deposited coin with trustee’s help.
- Blindfolded-freeness: No one can obtain a blinded coin without bank’s knowledge that this particular coin has been blinded.
- Coin-tracing: A bank can legally link the information of a coin to be used with the information of deposited coin with trustee’s help.
- Unlinkability: It is not possible to find any relation between different coins used by the same user.
- Refundability: It is possible to refund a coin if a legally withdrawn coin cannot be accepted by a bank or a trustee.
- Fairness: User’s anonymity with respect to the bank and the trustee must be guaranteed, but the anonymity must be revocable with the cooperation of the bank and the trustee.
- Framing-freeness: Any user or shop cannot be falsely incriminated by a bank or a trustee.
- Overspent-tracing: Any user or shop cannot be falsely incriminated by a bank or a trustee.
- Transferability: Once withdrawn coins can be transferred to any other user.
- Divisibility: The denomination of a coin can be divided into lower unit.

B. Petersen and Poupard’s Scheme

PePo97 which consists of initialization, account opening, registration, withdrawal, payment, deposit, and revocation protocols, is shown to be secure against the attacks of secret key extortion, coin extortion, blackmailing, framing, and blindfolding. It also satisfies the requirements of double-spending prevention, k-spendability, divisibility, anonymity revocation, coin-tracing, and refundability. These requirements were satisfied if existential forgery of a signature with respect to an adaptively chosen message attack is equivalent to a known computational hard problem (e.g., factorization or discrete log.). Off-line property in PePo97 can be realized by the registration of user’s pseudonymous public key to the trustee. Moreover, the prevention against extortion attack can be implemented by using the secure revocation lists and database. PePo97 has two types of protocols, i.e., the Internet payment and electronic purse protocols, which are quite different in terms of security and efficiency.

C. Our Goals and Approach

Our work aims at implementing an efficient Internet payment protocol with a smart card. This new e-cash system based on ECDLP (called as “Cashpia-v2”) has targeted a small-amount-of-money electronic transaction. The anonymity of transaction is of prime concern because it deals with a small-amount-of-money transaction. In general, when an honest user deals with a large-amount-of-money transaction (e.g., purchasing a house, a car, etc.), it is usually not allowed to make a payment anonymously due to lawful taxation and registration. On the other hand, in a small-amount-of-money transaction (e.g., purchasing private goods, etc.), there are many situations in which we want the purchasing details to be confidential. The tradeoff must be considered to design Cashpia-v2 because the complete anonymity will induce crimes when abused. Thus, anonymity must be guaranteed, but it can be revoked under a lawful order. In addition, Cashpia-v2 is also designed to provide many preventive functions against money laundering, blackmailling, and double spending. However Cashpia-v2 doesn’t provide divisibility, transferability and overspent-tracing because these requirements make all protocols hard to implement. To achieve these goals, we choose PePo97 as an underlying frame since PePo97 is found to be a strong off-line e-cash scheme resistant against various attacks with good efficiency, and also has most features which we try to achieve. But PePo97 requires high cost in storage and communication due to processing lots of databases. Thus it is hard to use PePo97 in the storage-limited platform such as smart card. To overcome this problem we extend PePo97 into ECDLP-based scheme.

In order to enforce the high efficiency and provable security together, Cashpia-v2 employs Baek et al.’s public key encryption scheme [2]. This scheme called as PSLC-2 (Provably Secure Length-saving public-key encryption scheme based on Computational Diffie-Hellman Assumption) is optimally designed on the elliptic curve over the finite field and is length-efficient in a sense that the length of ciphertext is shorter than that of Poincheval’s scheme [23]. Under the CDH-A (Computational Diffie-Hellman Assumption), the security of PSLC-2 is suggested to be provably secure against an adaptive chosen ciphertext attack.

We implement Cashpia-v2 using the cryptographic library ICUCLIB-v2 (ICU Cryptographic Library-version 2) developed and maintained by ICU. This library provides various cryptographic primitives to implement Cashpia-v2.

It is very hard to define how much the small-amount-of-money absolutely is. It may depend on the local customs.
Organization:
In Section II, we overview cryptographic primitives such as PSLC-2 in brief. In Sections III and IV, we describe the details of functional design and the implementation of Cashpia-v2, respectively. In Section V, we suggest an intuitive method to provide atomicity in Cashpia-v2. In Section VI, we analyze Cashpia-v2 in terms of its security and efficiency. Finally, we summarize our results and suggest future work in Section VII.

II. CRYPTOGRAPHIC PRIMITIVES
A key exchange protocol is used for two entities to share a session key secretly. The shared session key is used to exchange authenticated messages between them. In Cashpia-v2 all transmitted messages are encrypted with the shared session key by using Diffie-Hellman key agreement protocol [24].

We also adapt PSLC-2 to encrypt coin information with trustee’s public key. This operation is required to protect a withdrawn coin signed by the bank when the key is extorted. The following is a brief summary of PSLC-2 working on the elliptic curve over the finite field.

- Key generator \( \mathcal{K} \):
  - Choose a non-supersingular elliptic curve \( E(GF(p)) \) defined over Galois field \( GF(p) \) and calculate the order of elliptic curve \( #E(GF(p)) \). Let \( q \) be a large prime number dividing \( #E(GF(p)) \) and let \( P \) be a point of order \( q \) on \( E(GF(p)) \).
  - For a given random number \( u \in R \, GF(q) \), compute \( W = uP \). Then the public key is \( pk = (E, P, q, W) \) and the corresponding private key is \( sk = (E, P, q, u) \).

- Hash Function (two random oracles):
  - Choose \( H : \{0,1\}^k \rightarrow GF(q) \), and \( G : GF(p) \rightarrow \{0,1\}^k \).

- Encryption \( \mathcal{E} \):
  - Compute \( R = tP \) and \( S = tW \) where \( t = H(m||s) \), message \( m \in \{0,1\}^{k_0} \), and \( s \leftrightarrow R \, \{0,1\}^{k_1} \).
  - \( \mathcal{E}_{sk}(m, s) = (A, B) = (R, \langle G(x_S) \oplus (m||s) \rangle) \) where \( x_S \) is the \( x \)-coordinate of \( S \).

- Decryption \( \mathcal{D} \):
  - Compute \( S' = uA \) and \( t' = H(B \oplus G(x_{S'})) \).
  - If \( A = tP \), output \( D_{sk}(A, B) = [B \oplus G(x_{S'})]^{k_0} \). Otherwise, output “null”. Here, \( x_{S'} \) denotes the \( x \)-coordinate of \( S' \) and \( [B \oplus G(x_{S'})]^{k_0} \) denotes the first \( k_0 \) bits of \( [B \oplus G(x_{S'})] \).

**Theorem 1:** PSLC-2 is a provably secure public key encryption scheme.

**Proof:** See [2].

III. SYSTEM DESIGN
In this section, we describe the overall system architecture and each protocol step of Cashpia-v2 assuming that the bank and trustee may not falsely conspire together to frame an honest user, and the trustee is to be highly trustful.

A. System Architecture

Fig. 1 shows the overall configuration of Cashpia-v2. In general, an e-cash system consists of three basic entities: user, bank, and shop. To support anonymity revocation, the trustee must participate in the e-cash system. Thus, Cashpia-v2 consists of four entities: user, bank, shop, and trustee. Bank can play the role of trustee to make Cashpia-v2 simple. As shown in Fig. 1, Cashpia-v2 performs five payment protocols which consist of opening an account, registration, withdrawal, payment, and deposit protocols. Before beginning these protocols, the initialization protocol must be executed. The anonymity revocation protocol may be performed only when it is required. The major functions of each step are as follows:

- **Initialization:** System parameters and all participants’ key pairs are generated.
- **Opening an account:** The bank opens user’s account and registers user’s private information.
- **Registration:** The user generates and registers a pseudonymous public key to the trustee.
- **Withdrawal:** The user withdraws a coin from his own account into his device (PC or a smart card).
- **Payment:** The user pays the withdrawn coin to the shop in return for goods.
- **Deposit:** The shop transfers a coin to the bank and the bank deposits the corresponding coin to the shop’s account.
- **Anonymity revocation:** The trustee reveals the owner of a coin from the transcripts of the withdrawal phase or user’s identification from the transcripts of the payment phase.

**Notation:**
Throughout this paper, we use the following notation to describe Cashpia-v2.

- \( U, S, B, T \): Identity of user, shop, bank, and trustee, respectively
- \( Z \): an entity, \( Z \in \{U, S, B, T\} \)
- \( c \): coin (randomly chosen 24-bit value)
Cashpia-v2 uses databases and revocation lists to store and manage the data, to cope with extortion attack, and to revoke the anonymity efficiently. In the following, classification, element items, access authority, and descriptions of databases and revocation lists are explained in the form of “classification: {element items} [access authority]; Description.”.

\begin{itemize}
  \item **Coin-DB**: \{c, Ind(Q_p), σ_B\}, [User]: Store coin and signature obtained from the bank in the withdrawal phase.
  
  \item **User-DB**: \{id_u, Name, acc_u, Address, e-mail\}, ["Bank"] ; Store user-related information. It is set up in account opening phase. Here, acc_u includes acc_{CU} and acc_{SS}.
  
  \item **PsdPub-DB**: \{id_u, Q_p, σ_u, Ind(x_T)<\}, ["Trustee"] ; Store registered user’s pseudonymous public key and other transcripts of the registration phase.
  
  \item **Pay-DB**: \{c, Q_p, msg, σ_B, σ_T, σ_c\}, [Shop]: Store coin information obtained from a user in the payment phase.
  
  \item **PsdPrv-DB**: \{x_p, Q_p, R_T, σ_T\}, [User]: Store pseudonymous key pairs and other transcripts of the registration phase.
  
  \item **With-DB**: \{id_u, Ind(x_B), e', E_T(p_k(m, s))\}, [Bank]: Store transcripts of the withdrawal phase.
  
  \item **Dep-DB**: \{c, Q_p, id_u, σ_B, σ_T, σ_c\}, [Bank]: Store coin information obtained from the shop in the deposit phase.
  
  \item **User-RL**: \{Q_p\}, [Trustee, Shop]: List of user’s pseudonymous public key corresponding to the extorted pseudonymous private key.
  
  \item **Bank-RL**: \{Q_p\}, [Trustee, Shop]: List of bank’s public key corresponding to the extorted private key.
  
  \item **Trust-RL**: \{Q_T\}, [Trustee, Shop]: List of trustee’s public key corresponding to the extorted private key.
  
  \item **Coin-RL**: \{Q_p, c\}, [Trustee, Shop]: List of withdrawn but unused legal coins signed by bank’s extorted private key.
  
  \item **PsdPub-RL**: \{Q_p\}, [Trustee, Shop]: List of honest user’s pseudonymous public keys signed by trustee’s extorted private key.
  
  \item **Customer-RL**: \{id_u\}, [Bank]: List of the identifications of double-spending users.
\end{itemize}

\section{B. Databases and Revocation Lists}

Cashpia-v2 uses databases and revocation lists to store and manage the data, to cope with extortion attack, and to revoke the anonymity efficiently. In the following, classification, element items, access authority, and descriptions of databases and revocation lists are explained in the form of “classification: {element items} [access authority]; Description.”.

\begin{itemize}
  \item **Coin-DB**: \{c, Ind(Q_p), σ_B\}, [User]: Store coin and signature obtained from the bank in the withdrawal phase.
  
  \item **User-DB**: \{id_u, Name, acc_u, Address, e-mail\}, ["Bank"] ; Store user-related information. It is set up in account opening phase. Here, acc_u includes acc_{CU} and acc_{SS}.
  
  \item **PsdPub-DB**: \{id_u, Q_p, σ_u, Ind(x_T)<\}, ["Trustee"] ; Store registered user’s pseudonymous public key and other transcripts of the registration phase.
  
  \item **Pay-DB**: \{c, Q_p, msg, σ_B, σ_T, σ_c\}, [Shop]: Store coin information obtained from a user in the payment phase.
  
  \item **PsdPrv-DB**: \{x_p, Q_p, R_T, σ_T\}, [User]: Store pseudonymous key pairs and other transcripts of the registration phase.
  
  \item **With-DB**: \{id_u, Ind(x_B), e', E_T(p_k(m, s))\}, [Bank]: Store transcripts of the withdrawal phase.
  
  \item **Dep-DB**: \{c, Q_p, id_u, σ_B, σ_T, σ_c\}, [Bank]: Store coin information obtained from the shop in the deposit phase.
  
  \item **User-RL**: \{Q_p\}, [Trustee, Shop]: List of user’s pseudonymous public key corresponding to the extorted pseudonymous private key.
  
  \item **Bank-RL**: \{Q_p\}, [Trustee, Shop]: List of bank’s public key corresponding to the extorted private key.
  
  \item **Trust-RL**: \{Q_T\}, [Trustee, Shop]: List of trustee’s public key corresponding to the extorted private key.
  
  \item **Coin-RL**: \{Q_p, c\}, [Trustee, Shop]: List of withdrawn but unused legal coins signed by bank’s extorted private key.
  
  \item **PsdPub-RL**: \{Q_p\}, [Trustee, Shop]: List of honest user’s pseudonymous public keys signed by trustee’s extorted private key.
  
  \item **Customer-RL**: \{id_u\}, [Bank]: List of the identifications of double-spending users.
\end{itemize}

\section{C. Sub-protocols}

\subsection{Initialization}

\begin{itemize}
  \item 1. Each entity generates ephemeral key pairs under the assumption that each entity has its certificate.
  
  \item 2. B opens user’s account acc_u and sends it to U.
\end{itemize}

\subsection{Registration}

\begin{itemize}
  \item 1. U and T perform Diffie-Hellman key agreement protocol to share a session key K_{U,T}. All transmitted messages in this phase are encrypted with K_{U,T}.
  
  \item 2. U randomly generates a pseudonymous private key x_p in \([2, q-1]\). Corresponding pseudonymous public key Q_p is obtained with scalar multiplication x_p P. U also randomly generates an ephemeral private key k_{U_2} in \([2, q-1]\) and computes its corresponding public key R_{U_2}. U generates σ_T as concatenation of σ_T and R_{U_2}, encrypts it with K_{U,T} and sends it to T.
  
  \begin{equation}
  \begin{aligned}
  Q_p &= x_p P \\
  R_{U_2} &= k_{U_2} P \\
  σ_T &= x_{U_2} h(R_{U_2}) | id_u | Q_p + k_{U_2}
  \end{aligned}
  \end{equation}

  \item 3. T verifies signature σ_T as shown in (2). T generates signature σ_T for Q_p and generates σ_T by concatenating σ_T and R_{T_2} as shown in (3). T sends σ_T to U and stores id_u, Q_p, σ_u, and Ind(x_T<) in PsdPub-DB.
  
  \begin{equation}
  \begin{aligned}
  \text{if } (σ_T < = h(R_{T_2} | [id_u | Q_p]) Q_T_2 + R_{T_2}), \text{ then accept.}
  \end{aligned}
  \end{equation}

  \item 4. Choose k_{T_2} \in \{2, q-1\}.
  
  \begin{equation}
  \begin{aligned}
  R_{T_2} &= k_{T_2} P \\
  σ_T &= x_{T_2} h(R_{T_2}) | Q_p + k_{T_2}
  \end{aligned}
  \end{equation}

  \item 5. U verifies signature σ_T as (4). Then, U stores Q_p, x_p, R_{T_2}, and σ_T in PsdPrv-DB.
  
  \begin{equation}
  \begin{aligned}
  \text{if } (σ_T < = h(R_{T_2} | [id_u | Q_p]) Q_T_2 + R_{T_2}), \text{ then accept.}
  \end{aligned}
  \end{equation}

\subsection{Withdrawal}

\begin{itemize}
  \item 1. U and B perform Diffie-Hellman key agreement protocol and share the session key K_{U,B}. All transmitted messages in this phase are encrypted with K_{U,B}.
  
  \item 2. U randomly generates a coin c with 24 bits. U generates encryption E_{T_{p_k}}(m, s) for c and Q_p using the trustee’s
public key $Q_{T_2}$. (5) is the outcome of its encryption with PSLC-2. When extortion attack occurs, $E_{TrPr}(m, s)$ is transmitted to the trustee. $U$ asks a withdrawal of coin to $B$.

$$m = \left[ h\left(Q_{P_{\text{ps}}}ight)^b \right]^{k_1}$$

$$t = H\left(\left| m \right|\right)s$$

$$R_{U_1} = tP$$

$$S = tQ_{Tr}$$

$$E_{TrPr}(m, s) = (A, B) = (R_{U_2}, G(x_s) \oplus \left(\left| m \right|\right)s)$$

3. $B$ chooses a random number $k_{B_1} \in \mathbb{Z}_q \setminus \{2, q - 1\}$, computes $R_{B_1} = k_{B_1}P$, and sends $R_{B_1}$ to $U$.

4. After receiving $R_{B_1}$ from $B$, $U$ computes blinded value $e'$ for $c$, $Q_{ps}$, and $R_{B_1}$ using secret random values $u$ and $v$, and sends $e'$ to $B$.

$$s'_{B_1} = x_Be' + k_{B_1}$$

6. $U$ computes $B$’s signature $s_B$ by unblinding $s'_{B_1}$ using $u$ and $v$. $U$ generates $s_B$ by concatenating $s_B$ and $R_{B_1}$ as (8). $U$ verifies $\sigma_B$ by checking (9) and stores $\sigma_B$, $c$, and $Ind\left(Q_{ps}\right)$ in Coin-DB.

$$s_B = s'_{B_1} + v$$

$$\sigma_B = (R_{B_1}, \parallel s_B)$$

if $(s_B \parallel P = h(\langle R_{U_2}, \parallel \left| Q_{ps}\right| Q_{B_1} + R_{B_1} \rangle)$, then accept.

7. $B$ stores $id_U, Ind\left(x_{B_1}\right), e'$, and $E_{TrPr}(m, s)$ in With-DB. And $B$ withdraws the coin value from $accU$ of User-DB.

### Payment:

1. $S$ sends challenge $msg$ to $U$ as (10).

$$msg = h(id_{S})\left(time\right)$$

2. $U$ generates signature $\sigma_c$ as (11). $U$ sends $c, Q_{ps}, \sigma_T, \sigma_B, \sigma_c$, and $\sigma_T$ to $S$. At this time, $\sigma_c$ may be encrypted with $S$’s public key to prevent framing attack by $S$.

Choose $k_{U_3} \in \mathbb{Z}_q \setminus \{2, q - 1\}$.

$$R_{U_3} = k_{U_3}P$$

$$s_c = x_{ps}h(R_{U_3} \parallel e'[ids]\parallel msg\parallel Q_{ps}) + k_{U_3}$$

$$\sigma_c = (R_{U_3}\parallel s_c)$$

3. $S$ verifies signatures $\sigma_T, \sigma_B, \sigma_c$, and $\sigma_T$ as (12) and stores transcripts in Pay-DB. However, if extortion attack was reported, $S$ must examine black/white lists. If $U$’s private key $x_{ps}$ was extorted, $S$ must check that its corresponding public key $Q_{ps}$ is in User-BL. If $B$’s $x_B$ was extorted, $S$ must check that $Q_{B_1}$ is in Bank-BL and $c$ signed by $x_B$ is in Coin-DB. $S$ must check that $Q_{T_2}$ is in Trust-DB and $Q_{ps}$ signed by $x_{T_2}$ is in Pay-DB. After checking, $S$ stores transcripts $c, Q_{ps}, msg, \sigma_T, \sigma_B$, and $\sigma_c$ in Pay-DB.

Obtain $c, Q_{ps}, \sigma_T, \sigma_B, \sigma_c$, and $\sigma_T$ from the user if $(s_B \parallel P = h(\langle R_{T_2}, Q_{ps}\rangle + R_{T_2}) \& \& (s_{B_1} \parallel P = h(R_{B_1} \parallel Q_{ps} + R_{B_1}) \& \& (s_c \parallel P = h(R_{U_3} \parallel [ids \parallel msg\parallel Q_{ps} + R_{U_3}])$ (12) then accept.

### Deposit:

1. $S$ sends $c, Q_{ps}, \sigma_B$, and $\sigma_T$ obtained from $U$ to $B$. $B$ verifies the signatures as given in (12).

2. After the verification succeeds, $B$ checks if $c$ and $Q_{ps}$ obtained from $S$ exist in Dep-DB. If they do, $B$ finds $\sigma'$ for the deposited coin in Dep-DB and sends it to $S$ (detection of double-deposit or double-spending).

3. If $S$ receives $\sigma'$ from $B$, $S$ checks whether it is equal to $\sigma_c$. If $\sigma'$ and $\sigma_c$ are the same, $S$ rejects performing protocol (double-deposit). Otherwise, $S$ sends $\sigma_c, ids_S, \text{accs}_S$, and $msg$ to $B$. If $S$ does not receive any signature from $B$, $S$ sends $\sigma_c, ids_S, \text{accs}_S$, and $msg$ to Dep-DB.

4. If $B$ receives $\sigma_c, ids_S, \text{accs}_S$, and $msg$ from $S$, $B$ verifies signature $\sigma_c$. After the verification, $B$ deposits the value of coin into the account of $S$. $B$ stores $c, Q_{ps}, ids_T, \sigma_T, \sigma_c$, and $\sigma_B$ in Dep-DB.

5. If the same coin was already deposited, double spending is found. $B$ performs user tracing protocol with user’s $Q_{ps}$ and $c$ and detects the identity of the double spender.

### User Tracing:

1. $B$ cooperates with $T$ to detect the identity of the double spender. $B$ finds double-spent information $(c, Q_{ps}, \sigma_B, \sigma_T, \sigma_c)$ and $(c, Q_{ps}, \sigma_B, \sigma_T, \sigma_c)$ from Dep-DB and sends them to $T$.

2. $T$ verifies all signatures by (12). After detecting double spender’s $Q_{ps}$, $id_T$, and $\sigma_T$, $T$ sends them to $B$.


### Extortion Tracing:

The process of this phase is similar to that of PePo97. If the extortion of the secret key occurs, $T$ recognizes this and records the corresponding public key in the revocation list. $T$ distributes the modified revocation list to all participating shops. All shops use it to check the legality of coins in the payment phase.

Since key agreement protocol is performed after identifying $U$ and $B$, transparent blindfolding coins under $B$’s secret key $x_B$ is impossible. Also transparent blindfolding coins under trustee’s secret key $x_{T_2}$ is impossible since we use Schnorr signature scheme which is provably secure against chosen message attack [30].

### IV. SYSTEM IMPLEMENTATION

#### A. Tools

To implement Cash-opt-v2, we use the cryptographic library IUCLIB-v2 in which various cryptographic primitives are provided. The primitive algorithms included in IUCLIB-v2 are SEED [25], SHA-1, ElGamal public key encryption scheme [26], etc. It also includes cryptographic operations such as scalar multiplication on an elliptic curve, addition and doubling of points (Schroeppele et al.’s algorithm [27]) in optimal normal basis, and multiplication and multiplicative inverse operation over the finite field [28].

E-cash can be stored either in smart card or in hard disk. Smart card has limitation on memory capacity and processing
speed. On the other hand, an elliptic curve cryptographic algorithm can efficiently reduce data size and accelerate the computation. Cashpia-v2 can efficiently use smart card to store the coins.

**B. Implementation Overview**

We choose the elliptic curve over $GF(2^{131})$ as $y^2 + xy = x^3 + 1$ whose order is $4*6056473384187729693232012949340985129$ [26]. The simulation prototype was implemented by using C++ on Windows NT 4.0 and Windows 95/98. Fig. 2 shows an example of GUI (Graphical User Interface) of the system at the withdrawal phase. Access DB is used for the database management and cashpia.mdb is the database table used in servers and clients. The system is assumed to work on the client/server architecture. A client contacts a server in advance to make connections. The client side has the IP address and port configuration of the server. The login process requires client’s ID and password. We assume that all the pertinent clients are registered in the server. In the server side, all the transactions with clients are monitored through a log screen. Participants’ certificates are distributed each other in advance. Now, we describe the data structure and functions used to process the protocols at each phase.

**Data Structure:**

<table>
<thead>
<tr>
<th>Structure Name</th>
<th>MEMBER VARIABLE TYPE</th>
<th>member variable name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECI_CURVE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDEX</td>
<td>form</td>
<td>type of the elliptic curve</td>
</tr>
<tr>
<td></td>
<td>GF2N</td>
<td>pm_order</td>
<td>order of the base point</td>
</tr>
<tr>
<td></td>
<td>GF2N</td>
<td>cofactor</td>
<td>cofactor = $E/pn_{word7}$</td>
</tr>
<tr>
<td></td>
<td>GF2N</td>
<td>a_2</td>
<td>$y^2 + xy = x^3 + a_2 * x + a_3$</td>
</tr>
<tr>
<td></td>
<td>GF2N</td>
<td>a_3</td>
<td>$a_3, a_3$ are constants over the elliptic curve</td>
</tr>
<tr>
<td>ECI_POINT</td>
<td>GF2N</td>
<td>x</td>
<td>x coordinate value of the point</td>
</tr>
<tr>
<td></td>
<td>GF2N</td>
<td>y</td>
<td>y coordinate value of the point</td>
</tr>
<tr>
<td>EC_PARAMETER</td>
<td>CURVE</td>
<td>crv</td>
<td>elliptic curve</td>
</tr>
<tr>
<td></td>
<td>POINT</td>
<td>pt</td>
<td>base point of the elliptic curve</td>
</tr>
<tr>
<td>EC_KEYPAIR</td>
<td>GF2N</td>
<td>prvPsk_ec</td>
<td>private key</td>
</tr>
<tr>
<td></td>
<td>POINT</td>
<td>pubPsk_ec</td>
<td>public key</td>
</tr>
</tbody>
</table>

**Opening an Account:**

The information on transaction-enabled users and shops such as name, address, and phone number, etc. has been set to the database of the bank. Then bank issues account numbers to users and shops.

**Registration:**

In the registration phase, we apply DiffieHellman_KeyAgreement function of trustee and DH_Key_Exchange_step1, DH_Key_Exchange_step2, and DH_Key_Exchange_step3 functions of user to produce the session key. User generates a pseudonymous public key by executing the function PskKey_Creat_Reg_Step1 and PskKey_Creat_Reg_Step2. User encrypts by using the session key and passes to trustee. Trustee generates a signature on user’s pseudonymous public key and passes it to user in Registering function. And trustee’s Storing function stores the transcripts of the registration phase in PskPwReg_tbl table. User calls CODECset function and stores the transcripts in PskPwReg_tbl table.

**Withdrawal:**

User produces a coin as a 24-bit random number by using Generate_Regist_Coin_Step1 function configuring first 8-bit as the denomination of the coin and blinds the coin through Generate_Regist_Coin_Step2 function. Bank signs the blinded coin by using Withdrawal_Blind_Sign function and sends it to user. User who has received the signature unblinds it by Generate_Regist_Coin_Step3 function. The transcripts of this phase are stored in each WithBank-tbl table and Coin-tbl table. If a smart card is used, the coin information to be stored in the Coin-tbl table will be saved in the card.

**Payment:**

ReceiveData functions of shop and user are the main function of this phase. When user is connected, shop produces challenge information through Make_Mess function, and sends it to user. The user generates signature after receiving challenge information through ReceiveData function and sends the signature to shop with the coin information withdrawn from PskPwReg-tbl table using UserPskPubVfDB function. The shop verifies the signature received from ReceiveData function. After then, shop reads table TrustBL, UserBL, and BankBL from CTrustBL, CUserBL, and CBankBL functions, respectively. If the public key for each DB is not found and the coin and user’s pseudonymous public key are recorded in CoinWL table and PskPubWL tables, respectively, shop stores all transcripts received from CPay function in PayShop-tbl table.

**Deposit:**

Shop reads the currency information from Pay-tbl table of Pay-tbl function in the payment phase and sends the data to the bank. The bank verifies the signature by analyzing the data received from ReceiveData function. Then, the bank calls CDepBank_DB function and checks if it is already recorded in DepBank-tbl table. If there exists the coin with the same ID, the bank must extract user’s signature for the coin. If the signature to be sent by shop is not equal to the signature received from the bank or if the shop does not receive the signature from the bank, the shop passes the rest of the coin information to the bank through the ReceiveData function. If the signatures
have the same value, shop must stop the process since it represents double-deposit. If the same coin already exists in DB but the signatures are not identical, the bank verifies all the signatures in the ReceiveData function and calls the revocation protocol since it represents double-spending.

**Anonymity Revocation:**

This phase traces the identification of double-spending user. The deposit phase can call this phase. Bank searches for double-spending information in DepBank-tbl table of CReadDepBkTbl function and passes the information to trustee. Then trustee verifies the received signatures in SigVerify function and searches the user’s identification from the PsdPub-tbl table of SearchDB function, and passes the identification to the bank.

**V. ATOMICITY**

Xu et al. [22] addressed money conservation via atomicity in fair off-line e-cash. They assumed some possible scenarios in each phase of e-cash protocol, withdrawal, payment, and deposit protocol. For example, in the withdrawal protocol, the network can crash while a user is waiting for receiving the signature of e-cash. In this case, the e-cash that the user was supposed to receive would be lost because the user didn’t receive the e-cash although the bank had already withdrawn the corresponding money. They solved such problems via withdrawal atomicity, payment-delivery atomicity, and deposit atomicity in fair off-line e-cash schemes [17], [11] by using recovery, resolve, dispute handling protocols, and refund function.

Cashpia-v2 can be extended to guarantee atomicity for purchasing electronic goods as follows:

1. After initialization, $U$ and $S$ negotiate through some interaction to agree on the price, goods, and time-out value, which means the predetermined time limit from receiving coin of $S$ to deposit it in its own account.
2. $S$ sends an encrypted goods $E_k(\text{goods})$ and a transaction data $\text{msg}$ to $U$.
3. When $U$ claims that he has received goods different from what he had ordered, he computes $h(\text{good})$ and transmits it to $T$ in order to get the validity of goods.
4. $U$ sends $\sigma_c = (R_{\sigma_c}, [[s]], \text{time-out})$ to $S$ during the withdrawal phase, where $\sigma_c$ includes the time-out value with the signature of coin.
5. After $S$ verifies the signature of coin, $S$ sends the encryption key $k$ to $U$.

**Withdrawal Atomicity:**

If the network crashes during the withdrawal protocol, then recovery is processed as the following steps:

1. $U$ sends $(R_{\sigma_c}, e')$ to $B$, where $R_{\sigma_c}$ is ephemeral public key of $B$ and $e'$ is blinded value computed by $U$.
2. If a withdrawal phase started before the network crashed, and hasn’t completed yet, $B$ sends the signature $S_{\sigma_c}'$ to $U$.
3. Otherwise, this protocol is cancelled.
4. To prevent a double withdrawal, $B$ has to send a new $R_{\sigma_c}$ to $U$ and previously signed coin should be blacklisted.

**Payment and Delivery Atomicity:**

1. $U$ sends the transcript of the broken payment phase transaction to $S$.
2. $S$ verifies the transcript, in particular time-out value involved in the signature of coin. If a payment phase started before the network crashed, and still bounded within time-out, $S$ sends the encryption key, $k$, for goods to $U$.
3. If the payment phase has been exceeded the time-out, $U$ sends the transcript of the broken transaction to $T$.
4. $T$ asks $B$ whether this coin has ever been deposited.
5. If the deposit has been done, $T$ asks $S$ to send encryption key $k$ to $U$. Otherwise, this payment is cancelled and the coin will be stored at the blacklist.

**Deposit Atomicity:**

1. $S$ sends the transcript of the broken deposit phase transaction to $B$.
2. After $B$ verifies the received transcripts, $B$ processes the deposit phase.

**VI. SECURITY AND EFFICIENCY**

We will show that Cashpia-v2 satisfies most security requirements of e-cash system such as unforgeability, double-spending prevention, anonymity, untraceability, refundability, fairness, framing-freeness, coin-tracing, and blindfolding freeness.

**Theorem 2:** Assuming $(S_B, V_B)$ is a computationally secure elliptic curve blind signature scheme and the function $h$ is a collision free hash function, the system supports the security against forging of coin.

**Proof:** Since the blind signature is secure against existential forger, this allows only the legal bank to generate the signature for coin. As the hash function has the feature of collision free, the user cannot find a value $c' \neq c$ or $Q_{Ps} \neq Q_p$ with $h(c' || Q_{Ps}) = h(c || Q_p)$. Thus, the system satisfies unforgeability of coins.

**Theorem 3:** Assuming $(S_B, V_B)$ is a computationally secure elliptic curve blind signature scheme, and $(E_{TrK}, V_{TrK})$ is a strong probabilistic encryption scheme, the system supports the security against tracing an honest user by the bank.

**Proof:** Since the blind signature cannot give any information for the coin, the bank cannot link the blind coin with an encrypted coin by the probabilistic encryption scheme.

**Theorem 4:** Assuming $(S_C, V_C)$ is a computationally secure elliptic curve signature scheme, the system supports the security against impersonation, framing attack by a bank, and tracing an honest user with the trustee’s help.

**Proof:** Since anyone (even the bank or the trustee) who doesn’t know user’s pseudonymous secret key cannot generate a signature on the coin, the impersonation by the bank or trustee is impossible. If the bank wants to frame a user, the bank must present at least two different signatures $\sigma_c$ and $\sigma_c'$ to claim double spending. Since $(S_C, V_C)$ is a computationally secure elliptic curve signature scheme, the bank cannot forge any signature. Therefore, the framing attack is prevented. Given a coin, the
be secure against secret key extortion, coin extortion, forgery, framing, and blindfolding attacks. Cashpia-v2 is designed under the limited-storage environment such as smart card. To achieve these goals, we choose PePo97 scheme and extend to improve efficiency in terms of message size. The message size is reduced by 6.4 times compared to the original PePo97. Because of the shortened message size, handling and managing database is easy even under the smart card environment. Furthermore, we can achieve more secure and efficient electronic cash system by using PSLC-2. As an extension, we propose an intuitive method to add atomicity into an electronic cash system.

However, Cashpia-v2 doesn’t satisfy divisibility and transferability which are required to build a versatile e-cash system. As a further work, we suggest to research an efficient versatile e-cash system on the elliptic curve and to apply it to the real implementation where the computing power is rather limited.

**ACKNOWLEDGMENT**

The authors are very grateful to the anonymous reviewers for their valuable suggestion to improve our manuscript. Special thanks go to Mrs Heesun Kim for her powerful initiating this work when she was at ICU.

**REFERENCES**


