On the Security of an Unlinkable Off-line E-Cash System

Yosuke Tomii * Yoshifumi Manabe Tatsuaki Okamoto †

Abstract — This paper presents an off-line anonymous e-cash schemes, that is secure under the strong RSA assumption and the strong Diffie-Hellman (SDH) assumption. A user can withdraw a wallet containing \(2^k\) coins, each of which she can spend unlinkably. The complexity of the withdrawal operation is \(O(k^4)\), the spend operation is \(O(k^3)\), where \(k\) is security parameter. The user’s wallet can be stored using \(O(k)\) bits. Our scheme also offers exculpability of users, that is, the bank can prove to third parties that a user has double-spent. Our scheme is secure in the random oracle model.

Keywords: e-cash

1 Introduction

1.1 Background

Electronic cash was proposed by Chaum [2][3], and has been extensively studied [4][5][6][7][8][9][10][12][13].

As a coin is represented by digital data, and it is easy to duplicate data, an electronic cash scheme requires a mechanism that prevents a user from spending the same coin twice (double-spending). There are two scenarios. In the on-line scenario, the bank is on-line in each transaction to ensure that no coin is spent twice, and each merchant must consult the bank before accepting a payment. In the off-line scenario, the merchant accepts a payment autonomously, and later submits the payment to the bank; the merchant is guaranteed that such a payment will be either honored by the bank, or will lead to the identification (and therefore punishment) of the double-spender.

In this paper, we give an off-line \(2^k\)-spendable unlinkable electronic cash scheme. Our paper’s framework is based on [15] by Camenisch.

1.2 Our Result

This paper propose a new efficient unlinkable off-line electronic cash scheme secure in the random oracle model. The security proof of our scheme depends on the RSA assumption and the SDH assumption.

2 Preliminaries

2.1 Definition of Off-line E-Cash System

Our electronic cash scenario consists of three usual players: user \(U\), bank \(B\), and merchant \(M\); together with the algorithms: \text{BKeygen}, \text{UKeygen}, \text{MKeygen}, \text{Withdraw}, \text{Spend}, \text{Deposit}, \text{Identify}, \text{Trace} and \text{VerifyOwnership}.

- \text{BKeygen} is a key generation algorithm for bank \(B\). It takes as input \(k\) bit security parameter, and outputs the key pair, \((pk_B, sk_B)\).
- \text{UKeygen} is a key generation algorithm for
user $U$. It takes as input $k$ bit security parameter, and outputs the key pair, $(pk_U, sk_U)$.

- **Withdraw** is a protocol between $U$ and $B$. $U$ withdraws a $2^l$ unit wallet $W$ with serial number $S$. $U$ sends signature $Q$ to $B$. $B$ records $Q$ in database $D$ to trace user’s double spending some coin. $U$ receives $B$’s signature.

- **Spend** is a protocol between $U$ and $M$. $U$ sends zero-knowledge proof of knowledge of $W \Phi$ to $M$.

- **Deposit** is a protocol between $M$ and $B$. $M$ sends $\Phi$ to $B$. $B$ verifies $\Phi$. If the coin has been received already, $B$ rejects $\Phi$. Otherwise, $B$ accepts it.

- **Identify** is an algorithm to find double-spend $U'$ from double spent coin $\Phi_1, \Phi_2$.

- **Trace** is an algorithm to output evidence $\Pi$, which $B$ computes from $\Phi_1, \Phi_2$ and $D$, to be used in the VerifyOwnership step.

- **VerifyOwnership** is an algorithm to confirm that $U'$ certainly spent coin $\Phi_1, \Phi_2$. Anyone can verify double spent coin via serial number $S$ and $\Pi$.

### 2.2 Definition of Security

#### 2.2.1 Unforgeability

Adversary $A$ is given the bank’s public key $pk_B$. First, $A$ interacts with $B$ $K$ times in the Withdraw protocol. $B$ issues $2^l$ coins in each withdrawal. Second, $A$ executes the Spend protocol with $M$. $B$ accepts $2^l-K+1$ coins and $M$ cannot output $A$’s secret key. We define $\text{Adv}_A^{\text{unforge}}$ to be the probability that $A$ wins the above game, taken over the coin tosses made by $A$ and $M$.

#### 2.2.2 Identification of double-spenders

Adversary $A$ is given the bank’s public key $pk_B$. First, $A$ interacts $K$ times with $B$ in the Withdraw protocol. $B$ issues $2^l$ coins in each withdrawal. Last, $A$ issues $2^l$ coins in each withdrawal. $A$ executes the Spend protocol with $M$. $A$ wins the game if $M$ accepts $2^l-K+1$ coins and $M$ cannot output $A$’s secret key. We define $\text{Adv}_A^{\text{identify}}$ to be the probability that $A$ wins the above game, taken over the coin tosses made by $A$ and $M$.

### 2.2.3 Tracing double-spenders

Adversary $A$ is given the bank’s public key $pk_B$. First, $A$ interacts $K$ times with $B$ in the Withdraw protocol. $B$ issues $2^l$ coins in each withdrawal. Second, $A$ executes the Spend protocol with $M$. Last, $M$ executes the Deposit protocol. $B$ accepts $2^l-K+1$ coins and outputs evidence $\Pi$. $A$ wins the game if $B$ cannot output valid verify-ownership data. We define $\text{Adv}_A^{\text{trace}}$ to be the probability that $A$ wins the above game, taken over the coin tosses made by $A$ and $M$.

### 2.2.4 Anonymity of users

Adversary $A$ sets a bank’s secret and public key $pk_B, sk_B$. Honest users $U_0$, $U_1$ execute the withdraw protocol, and get wallet $W_0$, $W_1$, respectively.

One of $U_0$ and $U_1$ is now selected randomly, say $U_0$. $U_0$ executes the spend protocol. $A$ outputs $b' = 0$ or 1.

$$\text{Adv}_A^{\text{anonymity}} := 2 \Pr[b = b'] - 1$$

#### 2.2.5 Exculpability

Adversary $A$ sets $pk_B, sk_B$. An honest $U$ executes withdraw and spend protocols as many times as $A$ wishes.

$A$ wins the game if $A$ outputs $(S, \Pi)$ of user $U$ such that $\text{VerifyOwnership}(S, \Pi)$ returns accept.

### 2.3 Verifiable Encryption

In Section 4.2, we apply a technique by Camenisch and Damgard [14] for turning any semantically secure encryption scheme into a verifiable encryption scheme. A verifiable encryption scheme is a two-party protocol between a
prover and encryptor \( \mathcal{U} \) and a verifier and receiver \( \mathcal{B} \).

In the following, the verifiable encryption of a committed value is shown, in which ElGamal encryption is applied to the keys using bilinear maps.

- **Encryption and Decryption**
  
  \[ \tilde{g} \in \mathbb{G}_1 \text{ and } g, f, h \in \mathbb{G}_2 \] are public data. \( \mathcal{U} \) randomly chooses \( u \in \mathbb{Z}_p^\ast \) and computes \( e(\tilde{g}, g^u) = e(\tilde{g}, g)^u \). Let \( m \) be the plaintext and \( c \) the cyphertext.

  **Encrypt**: \( \mathcal{U} \) randomly chooses \( k \in \mathbb{Z}_p^\ast \),
  \[ c := (c_1, c_2) = (\tilde{g}^k, p_k^k m). \]

  **Decrypt**: \( m = \frac{c_2}{e(\tilde{g}_1, g^r)} \)

- **A Verifiable Encryption Scheme**
  
  \[ A := \tilde{g}^w f^h s \] is a commitment to \( s \). \( E(s) := (\tilde{g}^k, p_k^k s) \) is an encryption of \( s \). \( \mathcal{U} \) randomly chooses \( r_1, r_2, r_3, r_4, k_1, k_2 \in \mathbb{Z}_p^\ast \). \( \mathcal{U} \) computes

  \[ X = \tilde{g}^{r_1} \tilde{g}^{r_2} \tilde{h}^{r_3} \]
  \[ c_{11} = r_1 + u \mod p \]
  \[ c_{12} = r_2 + t \mod p \]
  \[ c_{13} = r_3 + s \mod p \]
  \[ c_{21} = r_1 + 2u \mod p \]
  \[ c_{22} = r_2 + 2t \mod p \]
  \[ c_{23} = r_3 + 2s \mod p \]
  \[ e_1 = (\tilde{g}^{k_1}, p_k^{k_1}(c_{11}||c_{12}||c_{13})) \]
  \[ e_2 = (\tilde{g}^{k_2}, p_k^{k_2}(c_{21}||c_{22}||c_{23})) \]

  and sends \( \{X, e_1, e_2\} \) to \( \mathcal{B} \).

  \( \mathcal{B} \) returns to \( a = \{1 \text{ or } 2\} \) randomly.

  \( \mathcal{U} \) sends \( \{c_{a1}, c_{a2}, c_{a3}, k_a\} \) to \( \mathcal{V} \). Let \( \tilde{a} = \{1 \text{ if } a = 2, 2 \text{ if } a = 1\} \).

  \( \mathcal{B} \) verifies

  \[ e_a = (\tilde{g}^{k_a}, p_k^{k_a}(c_{a1}||c_{a2}||c_{a3})) \]
  \[ \tilde{f}^{c_{a1}} \tilde{g}^{c_{a2}} \tilde{h}^{c_{a3}} = X^a \]
  \[ E(s) = (e_{\tilde{a}}, c_{a1}||c_{a2}||c_{a3}) \]

  By decrypting \( e_{\tilde{a}} \), \( \mathcal{B} \) obtains \( c_{a1}, c_{a2} \) and \( c_{a3} \), and calculates \( s \) by \( s := c_{23} - c_{13} \mod p \).

  This protocol is repeated \( k \) times, and \( \mathcal{U} \) succeeds in cheating \( \mathcal{B} \) with probability \( \frac{1}{2^k} \).

2.4 **Committed Number Lies in an Interval**

In Section 4.3, we apply a technique proposed by Boudot [1] to prove Committed Number \( J \) belongs to \([0, 2^k)\). This requires the strong RSA assumption.

2.5 **Signature Scheme**

In Sections 4.2, 4.3, 4.7, we apply a signature scheme proposed by Okamoto [16] to achieve anonymity and traceability. The signature scheme is existentially unforgeable against adaptive chosen message attacks.

- **Key generation**

  Randomly select generators \( g_2, u_2, v_2 \in \mathbb{G}_2 \) and set \( g_1 \leftarrow \phi(g_2), u_1 \leftarrow \phi(u_2) \) and \( v_1 \leftarrow \phi(v_2) \).

  Randomly select \( x \in \mathbb{Z}_p^\ast \) and compute \( w_2 \leftarrow g_2^x \in \mathbb{G}_2 \). The public and secret keys are:

  **Public key**: \( g_1, g_2, u_2, v_2, v_2 \),

  **Secret key**: \( x \)

- **Signature generation**

  Let \( m \in \mathbb{Z}_p^\ast \) be the message to be signed. Signer \( \mathcal{S} \) randomly selects \( r \) and \( s \) from \( \mathbb{Z}_p^\ast \) and computes

  \[ \sigma \leftarrow (g_1^m u_1 v_1^s)^{1 \over x+r} \]

  \( (\sigma, r, s) \) is the signature of \( m \).

- **Signature verification**

  Given public-key \((g_1, g_2, u_2, v_2)\), message \( m \), and signature \((\sigma, r, s)\), check that \( m, r, s \in \mathbb{Z}_p^\ast \), \( \sigma \in \mathbb{G}_1 \), \( \sigma \neq 1 \), and

  \[ e(\sigma, w_2 g^r) = e(g_1, g_2^m u_2 v_2^s). \]

  If they hold, the verification result is valid; otherwise the result is invalid.

- **Security of signature**

  The signature scheme is secure against adaptive chosen message attacks in the standard model under the SDH-assumption.[16]

3 Assumptions

3.1 **Strong RSA Assumption**:

Given RSA module \( n \) and random element \( g \in \mathbb{Z}_n^\ast \), it is hard to compute \( h \in \mathbb{Z}_n^\ast \) and
integer $e > 1$ such that $h^e \equiv g \mod n$. Module $n$ has special form $pq$, where $p = 2p' + 1$ and $q = 2q' + 1$ are safe primes. The S-RSA problem is defined as follows: given $n$ as input, output pair $(a, b)$

\[
\text{Adv}_{S\text{-RSA}} := \Pr[n = ab]
\]

Adversary $A(t, e)$-breaks S-RSA problem if $A$ runs in time at most $t$ and $\text{Adv}_{S\text{-RSA}}$ is at least $e$. The $(t, e)$-S-RSA assumption holds if no adversary $A(t, e)$-breaks the S-RSA problem.

### 3.2 Discrete Logarithm (DL) Assumption:

Given a large prime $p$ and random elements $f, g, h \in \mathcal{G}$ order $p$. It is hard to find $x, y, z$ that $f^x = g^y h^z$.

\[
\text{Adv}_{DL} := \Pr[A(f, g, h) = (x, y, z : f^x = g^y h^z)]
\]

Adversary $A(t, e)$-breaks DL problem if $A$ runs in time at most $t$ and $\text{Adv}_{DL}$ is at least $e$. The $(t, e)$-DL assumption holds if no adversary $A(t, e)$-breaks the DL problem.

### 3.3 Strong Diffie-Hellman (SDH) Assumption:

Let $(\mathcal{G}_1, \mathcal{G}_2)$ be bilinear groups. The $q$-SDH problem in $(\mathcal{G}_1, \mathcal{G}_2)$ is defined as follows: given the $(q + 2)$-tuple $(g_1, g_2, g_2^x, \ldots, g_2^{x^n})$ as input, output pair $(g_1^{x+z}, c)$ where $c \in \mathbb{Z}_p^*$. Algorithm $A$ has advantage, $\text{Adv}_{SDH}(q)$, in solving q-SDH in $(g_1^{x+z}, c)$ if

\[
\text{Adv}_{SDH}(q) \leftarrow \Pr[A(g_1, g_2, g_2^x, \ldots, g_2^{x^n}) = (g_1^{x+z}, c)]
\]

Adversary $A(t, e)$-breaks the q-SDH problem if $A$ runs in time at most $t$ and $\text{Adv}_{SDH}(q)$ is at least $e$. The $(q, t, e)$-SDH assumption holds if no adversary $A(t, e)$-breaks the q-SDH problem.

### 3.4 External Diffie-Hellman Assumption (XDH):

Suppose a bilinear mapping $e: \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}$. The XDH assumption states that the Decisional Diffie-Hellman (DDH) problem is hard in $\mathcal{G}_1$. This implies that there does not exist an efficiently computable isomorphism $\psi: \mathcal{G}_1 \to \mathcal{G}_2$.

\[
\text{Adv}_{XDH} := \Pr[A(g_1 \in \mathcal{G}_1, g_2 \in \mathcal{G}_2, g \in \mathcal{G}) = (g_1^u : g = e(g_1, g_2)^u)]
\]

Adversary $A(t, e)$-breaks XDH problem if $A$ runs in time at most $t$ and $\text{Adv}_{XDH}$ is at least $e$. The $(t, e)$-XDH assumption holds if no adversary $A(t, e)$-breaks the XDH problem.

### 4 Proposed E-cash System

#### 4.1 Key Generation

**Bank:** Upon input of security parameter, $B$ randomly generates

\[
\{g, f, h, v_b, w_b\} \in \mathbb{G}_2 \text{ and sets } \tilde{g} \leftarrow \psi(g), \tilde{f} \leftarrow \psi(f), \tilde{h} \leftarrow \psi(h), \tilde{v}_b \leftarrow \psi(v_b), \tilde{w}_b \leftarrow \psi(w_b)
\]

$B$ randomly selects $b \in \mathbb{Z}_p^*$ and computes $x_b \leftarrow g^b, y_b \leftarrow f^b, z_b \leftarrow h_b$. $B$'s public key $pk_B$ and secrets key $sk_B$ are:

\[
\begin{align*}
pk_B &= \{g, f, h, v_b, w_b, x_b, y_b, z_b\}, \quad sk_B = \{b\}.
\end{align*}
\]

**User:** $U$ randomly selects $\{v_u, w_u\} \in \mathbb{G}_2, u \in \mathbb{Z}_p^*$ and computes $x_u \leftarrow h^u, \tilde{v}_u \leftarrow v^u$ and $\tilde{w}_u \leftarrow w^u, \lambda = g^u$. $U$'s public key $pk_U$ and secret key $sk_U$ are:

\[
\begin{align*}
pk_U &= \{g, f, h, v_u, w_u, x_u, e(\tilde{g}, g)^u, \lambda\}, \quad sk_U = \{u, g^u, \mu\}.
\end{align*}
\]

### 4.2 Withdraw

1. $U$ identifies himself to bank $B$ by proving knowledge of $u. PK[u; x_u = h^u]$

2. $U$ randomly selects $v, s', t \in \mathbb{Z}_p^*$. $U$ sends $A' = \tilde{f} \tilde{v}^t \tilde{h}^{s'}$ to $B$. $B$ randomly selects $r' \in \mathbb{Z}_p^*$, and sends it to $U$. $U$ sets $s = r' + s$. $U$ and $B$ locally compute $A = \tilde{f}^u \tilde{g}^{r't} = A'h^{r'}$.

3. $U$ and $B$ execute the verifiable encryption protocol $k$ times. $U$ randomly selects $\pi_1, \pi_2 \in \mathbb{Z}_p^*$. $U$ computes signature $\{\sigma_u, \pi_1, \pi_2\}$ for $E(s_i) := g^{k_{ia}}|e(\tilde{g}, g)^k_{ia}s_i$.

\[
\sigma_u := (\tilde{g}^{E(s_i)} v_u w_u^{\pi_1})^{(\pi_2 + u)}
\]

$B$ verifies signature $\pi_1, \pi_2$ by

\[
e(\sigma_u, x_u g^{fu}) = e(\tilde{g}, g^{E(s_i)} v_u w_u^{\pi_1})
\]
$\mathcal{B}$ accepts

$$Q = (Q_1, \ldots, Q_k),$$

$$Q_i = (E(s_i), \pi_i, \rho_i)$$

4. $\mathcal{B}$ randomly selects $r_1, r_2 \in Z_p^*$. $\mathcal{B}$ computes $\sigma_B = (\alpha_B y_B)^{r_1} \mod p$, and sends $\sigma := \{\sigma_B, r_1, r_2\}$ to $\mathcal{U}$. $\mathcal{B}$ records the entry $(p_ku, Q, e(g, g)^{\eta})$ in his database $\mathcal{D}$. $\mathcal{U}$ verifies signature $\sigma$ by

$$e(\sigma_B, x_hy_b z_b (g f h)^{\tau_2}) = e(\tilde{f} \hat{g} \tilde{h}, f^u g^t h^s v_b w_b^{\tau_1}).$$

5. $\mathcal{U}$ saves the wallet $\mathcal{W} = (s, t, \sigma, J)$, where $J$ is an $l$-bit counter initially set to zero.

### 4.3 Spend

1. $\mathcal{U}$ receives spending data $I$ including merchant information. $\mathcal{U}$ computes $R = H(I)$.

2. $\mathcal{U}$ sends $S = g^{\frac{1}{r_1}}, T = g^{u + \frac{R}{r_2}}$ to $\mathcal{M}$.

3. $\mathcal{U}$ chooses $R_1, \ldots, R_{13} \in Z_p^*$ randomly. $\mathcal{U}$ computes

$$\sigma_B' = \sigma_B^{\eta},$$

$$\alpha = \{x_hy_b (f g h)^{\tau_2}\}^{\theta \eta},$$

$$\beta = \{f^u g^t h^s v_b w_b^{\tau_1}\}^{\theta},$$

$$X_u = f^{R_1} z^{R_2},$$

$$X_s = g^{R_3} z^{R_4},$$

$$X_t = h^{R_5} z^{R_6},$$

$$X_J = g^{R_7} z^{R_8},$$

$$X_{\alpha} = x_hy_b R_9 (f g h)^{R_{10}},$$

$$X_{\beta_1} = (f g h)^{R_{11}},$$

$$X_{\beta_2} = g^{R_6} R_{11} f^{-R_3} h^{R_1} R_{11} u_b^{R_{12}} w_b^{R_{13}},$$

$$X_{\beta_3} = g^{R_5} R_{13}^2,$$

$$X_s = g^{R_3} z^{R_7},$$

$$X_{T_1} = T z^{R_8} z^{R_7},$$

$$X_{T_2} = g^{R_5} z^{R_5},$$

$$X_{T_3} = g^{R_7} z^{R_4},$$

$$X_{T_4} = g^{R_5} z^{R_4} z^{R_7}.$$
\[ PK[u, t, J; T = g^{u + \frac{R}{1 + T}}] \]

4. \( \mathcal{M} \) verifies \( \Phi \). \( \mathcal{M} \) accepts the coin \( \{S, T, \Phi, R, I\} \).

5. If \( J > 2^j - 1 \), \( \mathcal{U} \) sets \( J = J + 1 \).

4.4 Deposit

\( \mathcal{M} \) sends the coin \( \{S, T, \Phi, R, I\} \) to \( \mathcal{B} \). \( \mathcal{B} \) verifies \( \Phi \), and accepts the coin if the \( (S, R) \) pair hasn’t been spent.

4.5 Identify

From the two coins that have the same \( S \) and different \( R \), \( \mathcal{B} \) computes \( s_{\mathcal{B}U} \):

\[
\left( \frac{R_1 - R_2}{T_1 - T_2} \right)^{R_1 - R_2} = g^u.
\]

4.6 Trace

\( \mathcal{B} \) finds \( e(g, g)^u = e(g^u, g) \). \( \mathcal{B} \) searches for \( e(g, g)^u \) in \( \mathcal{D} \), \( \mathcal{B} \) discovers double-spender’s \( p_{\mathcal{B}U} \).

\( \mathcal{B} \) recovers double spent coin \( s, J, S_j = \tilde{g}^s \).

from \( \mathcal{D} \). \( \mathcal{B} \) outputs \( \Pi := (s, J, g^u, p_{\mathcal{B}U}, \theta_1) \).

4.7 Verify Ownership

Bank opens \( \{S, J, s, g^u, E(s)i, \sigma_u, \pi_i, p_i, \tilde{g}^k\} \) where \( \{x_u, v_u, w_u, e(\tilde{g}, g)^u\} \) is user’s public key data. Thus anyone can check that the user with \( p_{\mathcal{B}U} \) is the owner of the coin with serial number \( s \) by \( S = g^{\frac{1}{T_2}}, e(\tilde{g}, g)^u = e(g^u, g)^u \), \( E(s)i = (\tilde{g}^k||e(\tilde{g}, g)^u s), e(\sigma_u, x_u g^{p_u}) = e(\tilde{g}, g^{E(s)i}, v_u w_u ^\pi) \).

5 Proof of Security

5.1 Unforgeability

Assume Adversary \( A \) is a \( t \)-time adversary whose \( \text{Adv}_{A}^{\text{unforge}} \) is at least \( \epsilon \). We will construct algorithm \( \mathcal{D} \) that breaks the unforgeability of the underlying signature scheme running in time at most \( t' \) with probability \( \epsilon' \).

1. (Input:)

\( \mathcal{D} \)’s input \( \{g_1, g_2, w_2, u_2, v_2\} \) is the public key of the signature. \( \mathcal{D} \) selects \( g, f, h \) such that \( g f h = g_1 \) and \( \mathcal{D} \) sets \( \{w_b = u_2, v_b = v_2, x_b y_b z_b = w_2(\text{random } x_b, y_b, z_b)\} \) as the public key of signature.

\( \mathcal{D} \) sets this input data as a bank’s public key.

2. (Withdraw)

\( \mathcal{D} \) must simulate a e-cash system when using \( A \). However, because \( \mathcal{D} \) cannot issue signatures, \( \mathcal{D} \) asks a signing oracle to get bank’s signature. While \( \mathcal{A} \) executes the withdraw protocol \( K \) times, \( \mathcal{A} \) demands \( K \) signatures for \( \mathcal{D} \). Whenever \( \mathcal{A} \) demands a signature, \( \mathcal{D} \) asks the signing oracle. Let \( \sigma_1, \sigma_2, \ldots, \sigma_K \) be the \( K \) signatures \( \mathcal{D} \) obtains from the signing oracle.

3. (Spend)

We show that if \( \mathcal{A} \) completes one spend protocol, \( \mathcal{D} \) extracts one signature.

\( \mathcal{A} \) sends spending data information \( I \) to the hash oracle to obtain a hash value. \( \mathcal{D} \) simulates random oracle \( H \). \( \mathcal{D} \) sends random number \( \gamma \) to \( \mathcal{A} \).

Once \( \mathcal{D} \) gets a valid coin data, \( \mathcal{D} \) resets \( \mathcal{A} \) and changes its response to the hash oracle to \( \gamma' \). Let \( \{\gamma', C_{\gamma'}, C_{\gamma'1}, \ldots\} \) be the spend coin data when \( \mathcal{A} \) completes the spend protocol for the first time. Those values are defined in Chapter 4.3. Let \( \{\gamma', C_{\gamma'}, C_{\gamma'1}, \ldots\} \) be the spend coin data when \( \mathcal{A} \) completes the spend protocol the second time.

\( \mathcal{A} \) randomizes the signature with \( \eta, \theta \) to use the signature in the spend protocol. Let \( \sigma_B \) be the original signature from the signing oracle and \( \sigma'_B \) be a randomized signature by \( \mathcal{A} \). \( \mathcal{D} \) computes

\[
C_\eta - C_{\eta'} = \frac{\theta}{\eta} (\gamma' - \gamma')
\]

\[
C_{\eta1} - C_{\eta'1} = \theta (\gamma - \gamma'),
\]

obtains \( \eta \), and calculates \( \sigma_B \) because \( \gamma' - \gamma' \neq 0 \). Consequently, \( \sigma_B = (\sigma'_B)^{\frac{1}{\eta}} \).
4. (Output)
\(\mathcal{A}\) completes the deposit protocol \(2^L K + 1\) times using \(2^L K\) coins, thus \(\mathcal{D}\) extracts \(2^L K + 1\) signatures.

In contrast, \(\mathcal{D}\) only received \(K\) signatures from the signing oracle. \(\mathcal{A}\) can use signature \(2^L\) times per serial number \(s\). \(\mathcal{A}\) must use serial number \(s\), which is unsigned from the bank, in at least one execution of the spend protocol. Thus \(\mathcal{D}\) gets at least one signature that the signing oracle did not issue.

\(\mathcal{D}\) selects random \(i\) and \(\mathcal{D}\) extracts only the \(i\)-th signature. We pay attention to the one spend protocol, \(\mathcal{A}\) must use \(s\) that is unsigned from the bank with probability greater than \(\frac{1}{2^t} K + 1\).

From the assumption, \(\mathcal{A}\) will complete the deposit protocol with probability \(\epsilon\) at this time. However, \(\mathcal{A}\) is not always completes the deposit protocol, probability of \(\epsilon\) again when \(\mathcal{D}\) resets \(\mathcal{A}\). Thus we consider the probability that \(\mathcal{A}\) in completing the deposit protocol with different \(\gamma\).

**Heavy Low Lemma**

Assume \(\mathcal{A}\) wins the unforgeability game with probability \(\epsilon\). Let \(\gamma\) be the hash value in the spend protocol. If \(\mathcal{D}\) resets \(\mathcal{A}\) and gives another hash value \(\gamma\), \(\mathcal{A}\) in completing the same spend protocol with probability of at least \(\frac{\epsilon}{2}\).

**Proof**

\(\mathcal{A}\) in completing the deposit protocol with probability \(\epsilon\) with random \(\gamma\) and other random parameters. We show a random \(\gamma\) and other random parameters, which \(\mathcal{A}\) succeeds the deposit protocol probability \(\frac{\epsilon}{2}\) as an different \(\gamma\) and the same other parameters, are selected probability more than \(\frac{1}{2}\). We define the heavy row that \(\mathcal{A}\) in completing the deposit protocol with probability of more than \(\frac{\epsilon}{2}\) using other fixed parameters. No heavy rows exist with probability of more than \(\frac{\epsilon}{2}\). Therefore, heavy rows exist with probability of more than \(\frac{\epsilon}{2}\). In other words, the random \(\gamma\) and other random parameters that allow \(\mathcal{A}\) to complete in the deposit protocol belong to a heavy row with probability of more than \(\frac{\epsilon}{2}\). Namely \(\mathcal{A}\) succeeds the deposit protocol probability \(\epsilon\), it belongs to heavy row probability more than \(\frac{1}{2}\) and \(\mathcal{A}\) succeeds the deposit protocol using different \(\gamma\) probability more than \(\frac{\epsilon}{2}\).

\[t' = 2t, \epsilon' = \frac{\epsilon^2}{4(2^t K + 1)}\]

\(2^L\) is coin size and a constant value. \(K\) is a polynomial value.

### 5.2 Identification of double-spenders

Assume Adversary \(\mathcal{A}\) is a \(t\)-time adversary whose \(\text{Adv}_A^{\text{idemifly}}\) is at least \(\epsilon\). We will construct algorithm \(\mathcal{D}\) that breaks the underlying signature scheme running in time at most \(t\) with probability \(\epsilon\).

1. (\(A\) breaks unforgeability)

   From definition of identification, \(\mathcal{A}\) withdraws a \(2^t\) coin \(K\) times and \(\mathcal{A}\) spends \(2^t K + 1\) coins.

   If two spending datas \(\{T = g^{u+\frac{K^t}{\gamma}}}, R\}, \{T' = g^{u'+\frac{K^t}{\gamma}}}, R\}'\) exist \(u = u' \wedge t = t' \wedge s = s'\), \(\mathcal{A}\) can compute user’s public key \(g^u\).

   \[
   \left(\frac{T^{u R}}{T'^{u R}}\right)^{\frac{1}{2^t}} = g^u.
   \]

   Thus two spending datas \(\{T = g^{u+\frac{R}{\gamma}}}, R\}, \{T' = g^{u'+\frac{R}{\gamma}}}, R\}'\) exist \(\overline{(u = u' \wedge t = t' \wedge s = s')}\). It is show that \(\mathcal{A}\) breaks unforgeability.

2. (\(A\) cheats verifiable encryption protocol)

   No more than \(\frac{1}{2^t}\) probability \(\mathcal{A}\) cheats the verifiable encryption protocol.

Because \(\mathcal{A}\) can break unforgeability with probability \(\epsilon - \frac{1}{2^t}\),

\[t' = 2t, \epsilon' = \frac{(\epsilon - \frac{1}{2^t})^2}{4(2^t K + 1)}\]

\(2^L\) is coin size and a constant value. \(K\) is a polynomial value.
5.3 Tracing double-spenders

Assume Adversary \( \mathcal{A} \) is a \( t \)-time adversary whose \( \text{Adv}_{\mathcal{A}}^\text{race} \) is at least \( \epsilon \). We will then construct algorithm \( \mathcal{D} \) that breaks the DL assumption with \( (t', c') \).

An informal outline of our proof is as follows: If \( \mathcal{A} \) completes the verifiable encryption, \( \mathcal{D} \) gets a DL-relation by resetting \( \mathcal{A} \). If \( \mathcal{A} \) completes the spend protocol, \( \mathcal{D} \) gets another DL-relation by resetting \( \mathcal{A} \). We show that if \( \mathcal{A} \) breaks the trace security, \( \mathcal{D} \) can obtain two different DL-relations.

Algorithm \( \mathcal{D} \) is constructed as follows:

1. (Input:)
   \( \mathcal{D} \)'s input \((f, g, h)\) is 3 elements of DL assumption.

2. (Bank's key generation:)
   \( \mathcal{D} \) sets \( \hat{f} = \psi(f), \hat{g} = \psi(g), \hat{h} = \psi(h) \). \( \mathcal{D} \) randomly selects generators \( v_b \) and \( w_b \) and sets \( \hat{v}_b = \psi(v_b), \hat{w}_b = \psi(w_b) \). \( \mathcal{D} \) randomly selects \( b \in \mathbb{Z}_p^* \) and computes \( x_b = g^b, y_b = f^b, z_b = h^b \).

3. (Share \( A \) in simulation withdraw protocol)
   \( \mathcal{D} \) receives \( A' \) from \( \mathcal{A} \). \( \mathcal{D} \) randomly selects \( r' \in \mathbb{Z}_p^* \) and sends it to \( \mathcal{A} \). Let \( A = A'h^{r'} \).

4. (Verifiable encryption)
   First, \( \mathcal{A} \) and \( \mathcal{D} \) execute verifiable encryption \( m \) times. At the \( m \)-th verifiable encryption cycle, \( \mathcal{D} \) receives \( X_m, e_{m1}, e_{m2} \). \( \mathcal{D} \) sends bit \( b_m = \{1 \text{ or } 2\} \) to \( \mathcal{A} \). \( \mathcal{D} \) receives \( m \)-th verifiable encryption data:
   \[
   c_{(m,b_m,1)}, c_{(m,b_m,2)}, c_{(m,b_m,3)}, \hat{k}_m, \tau_m, s_{m1}, s_{m2}.
   \]
   Second, \( \mathcal{D} \) resets \( \mathcal{A} \) and repeats verifiable encryption \( m \) times. If \( \mathcal{A} \) is reset, so \( \mathcal{A} \) returns the same \( X_m, e_{m1}, e_{m2} \). \( \mathcal{D} \) sends \( \bar{b}_m = \{1 \text{ or } 2\} \) that \( \bar{b}_m = 3 - b_m \). Finally, \( \mathcal{D} \) gets \( c_{(m,1,1)}, c_{(m,1,2)}, c_{(m,1,3)}, c_{(m,2,1)}, c_{(m,2,2)} \) and \( c_{(m,3,3)} \).
   \( \mathcal{D} \) computes \( \bar{u}_m = c_{(m,2,1)} - c_{(m,1,1)}, \bar{t}_m = c_{(m,2,2)} - c_{(m,1,2)} \) and \( s_m = c_{(m,2,3)} - c_{(m,1,3)} \).

If verifiable encryption is successful, \( f^{\bar{u}_m}g^{\bar{t}_m}h^{s_m} = A \) with probability \((1 - \frac{1}{\text{DL}})\) because
   \[
   \exists m, f^{c_{(m,1,1)}}g^{c_{(m,1,2)}}h^{c_{(m,1,3)}} = XA \land f^{c_{(m,2,1)}}g^{c_{(m,2,2)}}h^{c_{(m,2,3)}} = XA^2.
   \]
   Let \( \bar{u}, \bar{t} \) and \( \bar{s} \) denote \( \exists m \) above \( \bar{u}_m, \bar{t}_m \) and \( \bar{s}_m \).

5. (Spend)
   First, \( \mathcal{A} \) sends spending data information \( I \) to the hash oracle. \( \mathcal{D} \) simulates random oracle \( H \). \( \mathcal{D} \) sends random number \( \gamma \) as \( H(I||\ldots) \) to \( \mathcal{A} \).
   \( \mathcal{D} \) receives \( \{\gamma, C_u, C_t, C_s, \ldots\} \).
   Second, \( \mathcal{D} \) resets \( \mathcal{A} \) and repeats the spend protocol again. \( \mathcal{A} \) is reset, so \( \mathcal{A} \) returns the same value \( \{I, \ldots\} \). \( \mathcal{D} \) sends another random number \( \gamma' \) as \( H(I||\ldots) \) to \( \mathcal{A} \).
   \( \mathcal{D} \) receives \( \{\gamma', C'_{u}, C'_{t}, C'_{s}, \ldots\} \).
   \( \mathcal{D} \) computes
   \[
   u = \frac{C_u - C'_u}{\gamma - \gamma'}, \quad t = \frac{C_t - C'_t}{\gamma - \gamma'}, \quad s = \frac{C_s - C'_s}{\gamma - \gamma'}.
   \]
   Since \( \mathcal{A} \) succeeds in spending, \( (u, t, s) \) satisfies \( A = f^{u}g^{t}h^{s} \) with probability \( \epsilon \).

6. (Trace)
   If \( (\bar{u}, \bar{t}, \bar{s}) = (u, t, s) \), \( \mathcal{D} \) gets correct value \( g^{t} \) in the identify phase. Thus the bank can find the withdraw data to search the database. \( s \) is also correct, so \( Q \) is an evidence of double-spending.

7. (Output)
   \( \mathcal{D} \) finds discrete logarithm relation, \( f^{u}g^{t}h^{s} = f^{u}g^{t'}h^{s'} \).
   \[
   h = f^{\frac{\bar{u} - u}{\bar{t} - t}}g^{\frac{\bar{t} - t}{\bar{t} - t}}.
   \]
\[ D \text{ outputs} \]
\[ x = \frac{\bar{u} - u}{s - \bar{s}} \]
\[ y = 1 \]
\[ z = \frac{\bar{t} - t}{s - \bar{s}} . \]

\[ D \] gets the DL relation \( t' = 3t + cmT, \epsilon' = \epsilon(1 - \frac{1}{2^m}) \). \( c \) is a constant value, where \( T \) is the time to calculate an exponential.

5.4 Anonymity of users

Let \( A_1 = d^{y_1}f^{u_1}g^{t_1}s_1 \) be a share between \( A \) and \( U_1 \) and \( A_2 = d^{y_2}f^{u_2}g^{t_2}s_2 \) be a share between \( A \) and \( U_2 \).

\( d^{y_1} \) and \( d^{y_2} \) are randomize numbers, \( \forall y_1, u_1, t_1, s_1, y_2, u_2, t_2, s_2 \). In other words, no one distinguish \( A_1 \) of random select \( y_1 \) from \( A_2 \) of random select \( y_2 \). Thus this protocol is information-theoretically secure.

5.5 Exculpability

Assume Adversary \( A \) is a \( t \)-time adversary whose \( \text{Adv}'_{A}^{\text{exculpability}} \) is at least \( \epsilon \). We will then construct algorithm \( D \) that breaks the XDH assumption with \( (t', \epsilon') \).

\( A \) has to output \( g^u \) from \( e(g, \tilde{g})^u \). If such \( A \) exists, it breaks the XDH assumption. Thus \( D \) gives \( A e(g, \tilde{g})^u \) and outputs \( g^u \).

\[ t' = t, \epsilon' = \epsilon \]

6 Conclusion

This paper proposes two off-line anonymous e-cash schemes. One is an efficient off-line e-cash schemes that is secure under the strong RSA assumption, the strong Diffie-Hellman (SDH) assumption and External Diffie-Hellman (XDH) assumption with the random oracle model. This scheme is more efficient than the “Compact E-Cash” proposed by Jan Camenisch, Susan Hohenberger and Anna Lysyanskaya, and is removed non-standard assumption such as Sum-Free Decisional Diffie-Hellman assumption using “Efficient Blind and Partially Blind Signature Without Random Oracles” proposed by Tatsuaki Okamoto.

References


