An efficient certificateless two-party authenticated key agreement protocol

Debiao He [1] Sahadeo Padhye [2], Jianhua Chen [1,] *

[1] School of Mathematics and Statistics, Wuhan University, Wuhan, China

[2] Motilal Nehru NAtional Institute of Technology, Allahabad, India

*Email: chenjianhua.math@gmail.com

Abstract*:* Due to avoiding the key escrow problem in the identity-based cryptosystem, certificateless public key cryptosystem (CLPKC) has received a significant attention. As an important part of the CLPKC, the certificateless authenticated key agreement (CLAKA) protocol also received considerable attention. Most CLAKA protocols are built from bilinear mappings on elliptic curves which need costly operations. To improve the performance, several pairing-free CLAKA protocols have been proposed. In this paper we propose a new pairing-free CLAKA protocol. Compared with the related protocols our protocol has better performance. We also show our protocol is provably secure in a very strong security model, i.e. the extended Canetti-Krawczyk (eCK) model.

Key words*: Certificateless cryptography; Authenticated key agreement; Provable security; Bilinear pairings; Elliptic curve*

Classification Codes: 11T71, 94A60

1. Introduction

To realize information security, the public key cryptography has been widely used in networks communications. In the traditional public key cryptography (PKC), there is a need for certificate to assurance to the user about the relationship between a public key and the identity of the holder of the corresponding private key. So there come the problems of certificate management, including revocation, storage, distribution etc. [1]. To solve the above problem, Shamir introduced the concept of identity-based cryptography (ID-PKC) [2]. In ID-PKC setting, a user's public key can be derived from his identity (e.g., his name or email address) and his secret key is generated by the Key Generation Center (KGC). Then there come the key escrow problem, i.e. the PKG knows all the user's secret keys. In 2003, Al-Riyami et al. [3] proposed the certificateless public key cryptography (CLPKC) to solve the key escrow problem. Since then the CLPKC received a significant attention.

After Al-Riyami et al.'s work [3], numerous certificateless authenticated key agreement (CLAKA) protocols, using bilinear mappings on elliptic curves, have been proposed, e.g., [4–10]. However, the relative computation cost of a pairing is approximately twenty times higher than that of the scalar multiplication over elliptic curve group [11]. Therefore, CLAKA protocols without bilinear pairings would be more appealing in terms of efficiency. Recently, several CLAKA protocols without pairing have been proposed in [12-15]. Yang et al. [14] pointed out that neither Geng et al.'s protocol [14], nor Hou et al.'s protocol [13] is secure. He et al. [15] also proposed a CLAKA protocol without pairing. However, He et al.'s protocol is vulnerable to the type 1 adversary [16]. Although the latest CLAKA protocol [16] is more efficient than other protocols [12-15], it is provably secure under the mBR model [17], which is a very weak model. Yang et al. have shown that their scheme is provably secure in a very strong model-the extended Canetti-Krawczyk (eCK) model [18]. However, the user in Yang et al.' protocol needs nine elliptic curve scalar multiplications to finish the key agreement. Moreover, the user has to verify the validity of public keys. This not only increases the burden of the user, but also reverses the thought of CLPKC. In this paper, we will propose a new pairing-free CLAKA protocol, which is provably secure in the eCK model. Besides, our protocol has better performance than the related protocols.

The remainder of this paper is organized as follows. Section 2 describes some preliminaries. In Section 3, we propose our CLAKA protocol. The security analysis of the proposed protocol is presented in Section 4. In Section 5, performance analysis is presented. Finally, in Section 6 we conclude the paper.

2. Preliminaries

2.2. Notations

In this subsection, we first introduce some notations as follows, which are used in this paper.

- p, n : two large prime numbers;
- F_p : a finite field;
- E / F_p : an elliptic curve defined on F_p ;
- G : the cyclic additive group composed of the points on E / F_p ;

- P: a generator of G;
- $H_1(\cdot)$: a secure one-way hash function, where $H_1 : \{0,1\}^* \times G \to Z_n^*$;
- $H_2(\cdot)$: a secure one-way hash function, where $H_2 : \{0,1\}^* \times \{0,1\}^* \times G \times G \times G \times G \to Z_p^*$;
- ID_i: the identity of user i;
- (x, P_{pub}): the KGC's private/public key pair, where $P_{pub} = xP$;
- (x_i, P_i): the user i's secret value/public key pair, where $P_i = x_i \cdot P$;
- (r_i, R_i): a random point generated by KGC, where $R_i = r_i \cdot P$;
- (s_i, R_i) : the user i's partial private key, where $s_i = r_i + h_i x \bmod n$, $h_i = H_1(ID_i, R_i)$;
- (t_i, T_i): the user i's ephemeral private/public key pair, where $T_i = t_i \cdot P$;

2.1. Background of elliptic curve group

Let the symbol E / F_p denote an elliptic curve E over a prime finite field F_p, defined by an equation

$$y^2 = x^3 + ax + b, \quad a, b \in F_p \tag{1}$$

and with the discriminant

$$\Delta = 4a^3 + 27b^2 \neq 0. \tag{2}$$

The points on E / F_p together with an extra point O called the point at infinity form a group

$$G = \{(x, y) : x, y \in F_p, E(x, y) = 0\} \bigcup \{O\}. \tag{3}$$

G is a cyclic additive group in the point addition "+" defined as follows: Let $P, Q \in G$, l be the line containing P and Q (tangent line to E / F_p if $P = Q$), and R, the third point of intersection of l with E / F_p. Let l' be the line connecting R and O. Then P "+" Q is the point such that l' intersects E / F_p at R and O. Scalar multiplication over E / F_p can be computed as follows:

$$tP = P + P + \cdots + P(t \ times) \tag{4}.$$

Let the order of G be n. The following problems are commonly used in the security analysis of many cryptographic protocols.

Computational Diffie-Hellman (CDH) problem: Given a generator P of G and (aP, bP) for unknown $a, b \in_R Z_n^*$, the task of CDH problem is to compute abP.

For convenience, we define the function cdh as $cdh(aP, bP) = abP$

Decisional Diffie-Hellman (DDH) problem: Given a generator P of G and (aP, bP, cP) for unknown $a, b, c \in_R Z_n^*$, the task of DDH problem is to decide whether the equation $abP = cP$ holds.

Gap Diffie-Hellman (GDH) problem: Given a generator P of G, (aP, bP) for unknown $a, b \in_R Z_n^*$ and an oracle $\mathcal{O}_{ddhp}$, the task of GDH problem is to compute abP, where $\mathcal{O}_{ddhp}$ is a decision oracle that on input (aP, bP, cP), answers 1 if $cdh(aP, bP) = cP$; answers 0, otherwise.

The GDH assumption states that the probability of any polynomial-time algorithm to solve the GDH problem is negligible.

2.2. CLAKA protocol

A CLAKA protocol consists of six polynomial-time algorithms [2, 8]: $Setup$, $Partial\text{-}Private\text{-}Key\text{-}Extract$, $Set-Secret-Value$, $Set\text{-}Private\text{-}Key$, $Set-Public-Key$ and $Key-Agreement$. These algorithms are defined as follows.

$Setup$: This algorithm takes security parameter k as input and returns the system parameters $params$ and master key.

$Partial\text{-}Private\text{-}Key\text{-}Extract$: This algorithm takes $params$, master key, a user's identity ID_i as inputs and returns a partial private key.

$Set-Secret-Value$: This algorithm takes $params$ and a user's identity ID_i as inputs, and generates a secret value.

$Set\text{-}Private\text{-}Key$: This algorithm takes $params$, a user's partial private key and his secret value as inputs, and outputs the full private key.

$Set-Public-Key$: This algorithm takes $params$ and a user's secret value as inputs, and generates a public key for the user.

$Key-Agreement$: This is a probabilistic polynomial-time interactive algorithm which involves two entities A and B, if the protocol does not fail, A and B will obtain a secret session key.

2.3. Security model for CLAKA protocols

In CLAKA scheme, there are two types of adversaries with different capabilities [9, 14]. The type 1 adversary $\mathcal{A}1$ acts as a dishonest user while the type 2 adversary $\mathcal{A}2$ acts as a malicious key generation center (KGC). $\mathcal{A}1$ does not know the master key, but $\mathcal{A}1$ can replace the public keys of any entity with a value of his choice. $\mathcal{A}2$ knows the master key, but he cannot replace any user's public key.

Let $\prod_{i,j}^{s}$ represents the sth session which runs at party i with intended partner party j. A session $\prod_{i,j}^{s}$ enters an *accepted* state when it computes a session key $SK_{i,j}^{s}$. Two sessions $\prod_{i,j}^{s}$ and $\prod_{j,i}^{t}$ are called matching if they have the same session identity.

Lippold et al. [9] transformed original eCK model [18] from the traditional PKI-based setting to the CLPKC setting. The eCK model in the CLPKC setting is defined by the following game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A} \in \{\mathcal{A}1, \mathcal{A}2\}$. The game runs in two phases. During the first phase, the adversary $\mathcal{A}$ is allowed to issue the following queries in any order:

Create(i): On receiving such a query, $\mathcal{C}$ generates the public/private key pair for participant i with identity ID_i.

RevealMasterKey: $\mathcal{C}$ gives the master secret key to $\mathcal{A}$.

RevealSessionKey$(\prod_{i,j}^{s})$: If the session has not been accepted, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise $\mathcal{C}$ reveals the accepted session key to $\mathcal{A}$.

RevealPartialPrivateKey(i): $\mathcal{C}$ returns participant i's partial private key to $\mathcal{A}$.

RevealSecretValue(i): $\mathcal{C}$ returns participant i's secret value to $\mathcal{A}$.

ReplacePublicKey(i, pk): $\mathcal{C}$ replaces participant i's public key with the value chose by $\mathcal{A}$.

RevealEphemeralKey$(\prod_{i,j}^{s})$: $\mathcal{C}$ returns participant i's ephemeral private key to $\mathcal{A}$.

Send$(\prod_{i,j}^{s}, m)$: The adversary sends the message m to the session $\prod_{i,j}^{s}$ and get a response according to the protocol specification.

Once the adversary $\mathcal{A}$ decides that the first phase is over, it starts the second phase by choosing a fresh session $\prod_{i,j}^{s}$ and issuing a $Test(\prod_{i,j}^{s})$ query, where the fresh session and test query are defined later.

The type 1 adversary $\mathcal{A}1$ could get any user's secret value, since he can replace the public key of any entity with a value of his choice. The type 2 adversary $\mathcal{A}2$ could get any user's partial private key since he has access to the master key. Then several cases do not exist in Lippold et al.'s model [9]. To get better performance, we define the definition of freshness for CLAKA scheme against two type of adversary as follows.

Definition 1 (Freshness for CLAKA Scheme against Type 1 Adversary). Let instance $\prod_{i,j}^{s}$ be a completed session, which is executed by an honest party i with another honest party j. We define $\prod_{i,j}^{s}$ to be fresh if none of the following three conditions hold:

- The adversary $\mathcal{A}1$ reveals the session key of $\prod_{i,j}^{s}$ or of its matching session (if the latter exists).
- j is engaged in $\prod_{j,i}^{t}$ the session matching to $\prod_{i,j}^{s}$ and $\mathcal{A}1$ either reveals both of i's partial private key and $\prod_{i,j}^{s}$'s ephemeral private key or both of j's partial private key and $\prod_{j,i}^{t}$'s ephemeral private key.
- No sessions matching to $\prod_{i,j}^{s}$ exist and $\mathcal{A}1$ either reveals both of i's partial private key and $\prod_{i,j}^{s}$'s ephemeral private key or j's partial private key.

Definition 2 (Freshness for CLAKA Scheme against Type 2 Adversary). Let instance $\prod_{i,j}^{s}$ be a completed session, which is executed by an honest party i with another honest party j. We define $\prod_{i,j}^{s}$ to be fresh if none of the following three conditions hold:

- The adversary $\mathcal{A}2$ reveals the session key of $\prod_{i,j}^{s}$ or of its matching session (if the latter exists).
- j is engaged in $\prod_{j,i}^{t}$ the session matching to $\prod_{i,j}^{s}$ and $\mathcal{A}2$ either reveals both of i's secret value and $\prod_{i,j}^{s}$'s ephemeral private key or both of j's secret value and $\prod_{j,i}^{t}$'s ephemeral private key.

- No sessions matching to $\prod_{i,j}^{s}$ exist and $\mathcal{A}2$ either reveals both of i's secret value and $\prod_{i,j}^{s}$'s ephemeral private key or j's partial private key.

Test$(\prod_{i,j}^{s})$: At some point, $\mathcal{A}$ may choose one of the oracles, say $\prod_{i,j}^{s}$, to ask a single *Test* query. This oracle must be fresh. To answer the query, the oracle flips a fair coin $b \in \{0,1\}$, and returns the session key held by $\prod_{i,j}^{s}$ if $b = 0$, or a random sample from the distribution of the session key if $b = 1$.

At the end of the game, $\mathcal{A}$ must output a guess bit b'. $\mathcal{A}$ wins if and only if $b' = b$. $\mathcal{A}$'s advantage to win the above game, denoted by $Adv_{\mathcal{A}}(k)$, is defined as: $Adv_{\mathcal{A}}(k) = \left| \Pr[b' = b] - \frac{1}{2} \right|$, where k is a security parameter.

Definition 3. A CLAKA scheme is said to be secure if:

(1) In the presence of a benign adversary on $\prod_{i,j}^{s}$ and $\prod_{j,i}^{t}$, both oracles always agree on the same session key, and this key is distributed uniformly at random.

(2) For any adversary $\mathcal{A} \in \{\mathcal{A}1, \mathcal{A}2\}$, $Adv_{\mathcal{A}}(k)$ is negligible.

3. Our protocol

In this section, we will propose a new CLAKA protocol based on previous works [9, 14, 16]. Our protocol consists of six polynomial-time algorithms. They are described as follows.

Setup: This algorithm takes a security parameter k as an input, returns system parameters and a master key. Given k, KGC does the following steps.

1) KGC chooses a k-bit prime p and determines the tuple $\{F_p, E/F_p, G, P\}$ as defined in Section 2.1.

2) KGC chooses the master private key $x \in Z_n^*$ and computes the master public key $P_{pub} = xP$.

3) KGC chooses two cryptographic secure hash functions $H_1 : \{0,1\}^* \times G \to Z_n^*$ and $H_2 : \{0,1\}^* \times \{0,1\}^* \times G \times G \times G \times G \times G \to Z_n^*$.

4) KGC publishes $params = \{F_p, E/F_p, G, P, P_{pub}, H_1, H_2\}$ as system parameters and keeps the master key x secretly.

Partial - Private - Key - Extract : This algorithm takes master key, a user's identifier, system parameters as inputs, and returns the user's ID-based private key. KGC works as follows.

1) KGC chooses a random number $r_i \in Z_n^*$, computes $R_i = r_i \cdot P$ and $h_i = H_1(ID_i, R_i)$.

2) KGC computes $s_i = r_i + h_i x \bmod n$ and issues (s_i, R_i) to the users through secret channel.

Set – Secret – Value: The user picks randomly $x_i \in Z_n^*$, computes $P_i = x_i \cdot P$ and sets x_i as his secret value.

Set - Private - Key: The user with identity ID_i takes the pair $sk_i = (x_i, s_i)$ as its private key.

Set – Public – Key : The user with identity ID_i takes $pk_i = (P_i)$ as its public key.

Key – Agreement: Assume that an entity A with identity ID_A has private key $sk_A = (x_A, s_A)$ and public key $pk_A = (P_A)$ and an entity B with identity ID_B has private key $sk_B = (x_B, s_B)$ and public key $pk_B = (P_B)$ want to establish a session key, they can do, as shown in Fig.1, as follows.

1) A chooses a random number $t_A \in Z_n^*$ and computes $T_A = t_A \cdot P$, then A sends $M_1 = \{ID_A, R_A, T_A\}$ to B.

2) After receiving M_1, B chooses a random number $t_B \in Z_n^*$ and computes $T_B = t_B \cdot P$, then B sends $M_2 = \{ID_B, R_B, T_B\}$ to A.

Then both A and B can compute the shared secrets as follows:

A computes

$$K_{AB}^1 = (t_A + s_A)(T_B + R_B + H_1(ID_B, R_B)P_{pub}) \quad (5)$$

$$K_{AB}^2 = (t_A + x_A)(T_B + P_B) \quad (6)$$

and

$$K_{AB}^3 = t_A \cdot T_B \quad (7)$$

B computes

$$K_{BA}^1 = (t_B + s_B)(T_A + R_A + H_1(ID_A, R_A)P_{pub}) \quad (8)$$

$$K_{BA}^2 = (t_B + x_B)(T_A + P_A) \quad (9)$$

and

$$K_{BA}^3 = t_B \cdot T_A \tag{10}$$

Thus the agreed session key for A and B can be computed as:

$$\begin{aligned} sk &= H_2(ID_A \| ID_B \| T_A \| T_B \| K_{AB}^1 \| K_{AB}^2 \| K_{AB}^3) \\ &= H_2(ID_A \| ID_B \| T_A \| T_B \| K_{BA}^1 \| K_{BA}^2 \| K_{BA}^3) \end{aligned} \tag{11}$$

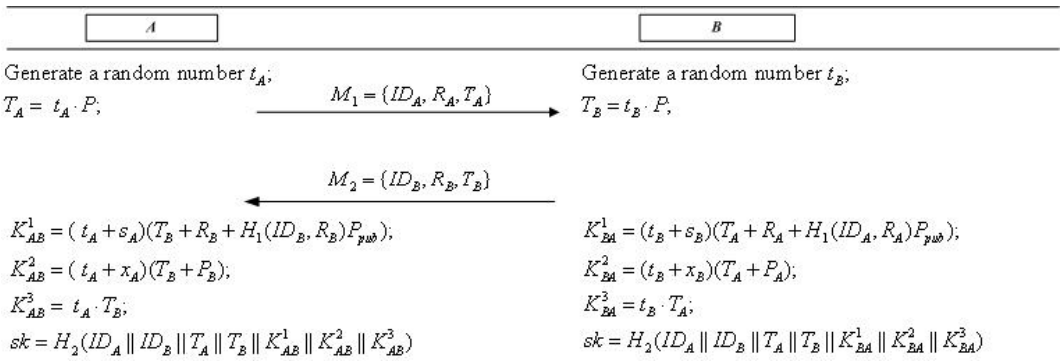

Fig. 1. Key agreement of our protocol

Since $T_A = t_A \cdot P$, $P_A = x_A \cdot P$, $s_A P = R_A + H_1(ID_A, R_A)P_{pub}$, $T_B = t_B \cdot P$, $P_B = x_B \cdot P$ and $s_B P = R_B + H_1(ID_B, R_B)P_{pub}$, then we have

$$\begin{aligned} K_{AB}^1 &= (t_A + s_A)(T_B + R_B + H_1(ID_B, R_B)P_{pub}) \\ &= (t_A + s_A)(t_B + s_B)P = (t_B + s_B)(t_A + s_A)P \\ &= (t_B + s_B)(T_A + R_A + H_1(ID_A, R_A)P_{pub}) = K_{BA}^1 \end{aligned} \tag{12}$$

$$\begin{aligned} K_{AB}^2 &= (t_A + x_A)(T_B + P_B) \\ &= (t_A + x_A)(t_B + x_B)P = (t_B + x_B)(t_A + x_A)P \\ &= (t_B + x_B)(T_A + P_A) = K_{BA}^2 \end{aligned} \tag{13}$$

and

$$K_{AB}^3 = t_A t_B P = t_B t_A P = K_{BA}^3 \tag{14}$$

Thus, the correctness of the protocol is proved.

4. Security Analysis

In this section, we will show our scheme is provably secure in eCK model. We treat H_1 and H_2 as two random oracles [19]. For the security, the following lemmas and theorems are provided.

Lemma 1. If two oracles are matching, both of them will be accepted and will get the same session key which is distributed uniformly at random in the session key sample space.

Proof. From the correction analysis of our protocol in Section 3, we know if two oracles are matching, then both of them are accepted and have the same session key. The session keys are distributed uniformly since t_A and t_B are selected uniformly during the execution.

Lemma 2. Assuming that the GDH problem is intractable, the advantage of a type 1 adversary against our protocol is negligible.

Proof. Suppose that there is a type 1 adversary $\mathcal{A}1$ who can win the game defined in subsection 2.3 with a non-negligible advantage $Adv_{\mathcal{A}1}(k)$ in polynomial-time t. Then, we will show how to use the ability of $\mathcal{A}1$ to construct an algorithm $\mathcal{C}$ to solve the GDH problem.

Let n_0 be the maximum number of sessions that any one party may have. Assume that the adversary $\mathcal{A}1$ activates at most n_1 distinctive honest parties. Assume that the adversary $\mathcal{A}1$ activates at most n_2 distinctive hash queries. Assume also that $Adv_{\mathcal{A}1}(k)$ is non-negligible. Before the game starts, $\mathcal{C}$ tries to guess the test session and the strategy that the adversary $\mathcal{A}1$ will adopt. $\mathcal{C}$ randomly selects two indexes $I, J \in \{1,\cdots,n_1\}$: $I \neq J$, which represent the I th and the J th distinct honest party that the adversary initially chooses. Also, $\mathcal{C}$ chooses $S \in \{1,\cdots,n_0\}$ and determines the *Test* session $\prod_{I,J}^{S}$, which is correct with probability larger than $\frac{1}{n_0 n_1^2}$. Let $\prod_{J,I}^{T}$ be the matching session of $\prod_{I,J}^{S}$. Since H_1 and H_2 are modeled as random oracles, after the adversary issues the test query, it has only three possible ways to distinguish the tested session key from a random string:

CASE 1: Forging attack: Assume that $\prod_{I,J}^{S}$ is the test session. At some point in its run, the adversary $\mathcal{A}1$ queries H_2 on the value $(ID_I, ID_J, T_I, T_J, K_{IJ}^1, K_{IJ}^2, K_{IJ}^3)$ in the test session owned by I communicating with J. Clearly, in this case $\mathcal{A}1$ computes the values K_{IJ}^1, K_{IJ}^2 and K_{IJ}^3 itself.

CASE 2: Guessing attack: $\mathcal{A}1$ correctly guesses the session key.

CASE 3: Key-replication attack: The adversary $\mathcal{A}1$ forces a non-matching session to have the same session key with the test session. In this case,

the adversary $\mathcal{A}1$ can simply learn the session key by querying the non-matching session.

Since H_2 is a random oracle, the probability of guessing the output of H_2 is $O(1/2^k)$, which is negligible. The input to the key derivation function H_2 includes all information that can uniquely identify the matching sessions. Since two non-matching sessions can not have the same identities and the same ephemeral public keys and H_2 is modeled as a random oracle, the success probability of **Key-replication attack** is also negligible. Thus **Guessing attack** and **Key-replication attack** can be ruled out, and the rest of the proof is mainly devoted to the analysis of **Forging attack**. As the attack that the adversary $\mathcal{A}1$ mounts is **Forging attack**, $\mathcal{A}1$ can not get an advantage in winning the game against the protocol unless it queries the H_2 oracle on the session key.

The rest of this section is mainly devoted to the analysis of the **Forging attack**. To relate the advantage of the adversary $\mathcal{A}1$ against our protocol to the GDH assumption, we use a classical reduction approach. In the following, a challenger $\mathcal{C}$ is interested to use the adversary $\mathcal{A}1$ to turn $\mathcal{A}1$'s advantage in distinguishing the tested session key from a random string into an advantage in solving the GDH problem. The following two sub-cases should be considered.

CASE 1.1: No honest party owns a matching session to the *Test* session.

CASE 1.2: The *Test* session has a matching session owned by another honest party.

- **The analysis of CASE 1.1**:

Since $\mathcal{A}1$ is strong type 1 adversary, then he can get any users' secret key x_i value through *ReplacePublicKey* query. According to Definition 1, $\mathcal{C}$ has the following two choices for $\mathcal{A}1$'s strategy:

CASE 1.1.1: At some point, the partial private key of party I has been revealed by the adversary $\mathcal{A}1$. According to Definition 1, $\mathcal{A}1$ is not permitted to reveal the ephemeral private key of the *Test* session.

CASE 1.1.2: The partial private key of party I has never been revealed by the adversary $\mathcal{A}1$. According to Definition 1, $\mathcal{A}1$ may reveal the ephemeral private key of the *Test* session.

CASE 1.1.1:

Let $Adv_{\mathcal{C}}^{GDH}(k)$ be the advantage that the challenger $\mathcal{C}$ gets in solving the GDH problem given the security parameter k. Given a GDH problem instance($U = uP, V = vP, \mathcal{O}_{ddhp}$) and $\mathcal{C}$'s task is to compute $cdh(U,V) = uvP$, where $\mathcal{O}_{ddhp}$ is a decision oracle that on input (aP, bP, cP), answers 1 if $cdh(aP, bP) = cP$; answers 0, otherwise. $\mathcal{C}$ first chooses $P_0 \in G$ at random, sets P_0 as the system public key P_{pub}, selects the system parameter $params = \{F_p, E/F_p, G, P, P_{pub}, H_1, H_2\}$, and sends $params$ to $\mathcal{A}1$. Then, $\mathcal{C}$ simulates the game outlined in Section 2.3 as follows.

Create(i): $\mathcal{C}$ maintains an initially empty list L_C consisting of tuples of the form (ID_i, s_i, R_i, x_i, P_i). If $i = J$, $\mathcal{C}$ chooses two random numbers $h_i, x_i \in Z_n^*$, computes $R_i = U - h_i P_0$, $P_i = x_i P$, sets $H_1(ID_i, R_i) \leftarrow h_i$ and stores ($ID_i, \perp, R_i, x_i, P_i$) and ($ID_i, R_i, h_i$) in L_C and L_{H_1} separately. Otherwise, $\mathcal{C}$ chooses three random numbers $s_i, h_i, x_i \in Z_n^*$, computes $R_i = s_i P - h_i P_{pub}$, $P_i = x_i P$, sets $H_1(ID_i, R_i) \leftarrow h_i$ and stores (ID_i, s_i, R_i, x_i, P_i) and (ID_i, R_i, h_i) in L_C and L_{H_1} separately.

$H_1(ID_i, R_i)$: $\mathcal{C}$ maintains an initially empty list L_{H_1} which contains tuples of the form (ID_i, R_i, h_i). If (ID_i, R_i) is on the list L_{H_1}, $\mathcal{C}$ returns h_i. Otherwise, $\mathcal{C}$ chooses a random number h_i, stores (ID_i, R_i, h_i) in L_{H_1} and returns h_i.

$H_2(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$: $\mathcal{C}$ maintains an initially empty list L_{H_2} with entries of the form ($ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk$). If the tuple is in the list L_{H_2}, $\mathcal{C}$ responds with sk. Otherwise, $\mathcal{C}$ responds to these queries in the following way:

- If $ID_i = ID_J$,
  - $\mathcal{C}$ looks the list L_S for entry ($ID_i, ID_j, T_i, T_j, *$). If $\mathcal{C}$ finds the entry, he computes $\overline{Z}_1 = Z_1 - t_i(T_j + R_j + H_1(ID_j, R_j)) - s_j(R_i + H_1(ID_i, R_i))$.
  - Then $\mathcal{C}$ checks whether Z_1 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple ($R_i + H_1(ID_i, R_i)P_{pub}, T_j, \overline{Z}_1$) is

inputted. $\mathcal{C}$ also checks whether Z_2 and Z_3 are equal by checking if the equations $Z_2 = (t_i + x_i)(T_j + P_j)$ and $Z_3 = t_i T_j$ hold separately. If Z_1, Z_2 and Z_3 are correct, $\mathcal{C}$ stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}, where the value sk comes from L_S. Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.

- Otherwise,
  - $\mathcal{C}$ looks up the list L_S for entry $(ID_i, ID_j, T_i, T_j, *)$. If $\mathcal{C}$ finds the entry, he stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}, where the value sk comes from L_S.
  - Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.

RevealPartialPrivateKey(i): $\mathcal{C}$ answers $\mathcal{A}1$'s queries as follows.

- If $ID_i = ID_J$ then $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ looks up the list L_E and returns the corresponding partial private key s_i to the adversary $\mathcal{A}1$.

RevealSecretValue(i): $\mathcal{C}$ looks up the table L_C for entry $(ID_i, *, *, *, *)$. If $\mathcal{C}$ finds the entry, he returns x_i. Otherwise, $\mathcal{C}$ carries out the query *Create*(i) and returns the corresponding x_i.

ReplacePublicKey(i, pk): Upon receiving the query, $\mathcal{C}$ looks up the table L_C for entry $(ID_i, *, *, *, *)$. If $\mathcal{C}$ finds the entry, he replaces x_i and P_i with x_i' and P_i' separately, where $pk = (P_i')$ and $P_i' = x_i' P$. Otherwise, $\mathcal{C}$ carries out *Create*(i) and replaces x_i and P_i with x_i' and P_i' separately.

RevealEphemeralKey$(\prod_{i,j}^{s})$: $\mathcal{C}$ answers $\mathcal{A}1$'s queries as follows.

- If $\prod_{i,j}^{s} = \prod_{I,J}^{S}$, then $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ returns the stored ephemeral private key to $\mathcal{A}1$.

RevealMasterKey: $\mathcal{C}$ stops the simulation.

RevealSessionKey$(\prod_{i,j}^{s})$: $\mathcal{C}$ answers $\mathcal{A}1$'s queries as follows.

- If $\prod_{i,j}^{s} = \prod_{I,J}^{S}$ or $\prod_{i,j}^{s} = \prod_{J,I}^{T}$, then $\mathcal{C}$ stops the simulation.
- Otherwise, if $\mathcal{C}$ returns the session key sk to $\mathcal{A}1$.

$Send(\prod_{i,j}^{t}, m): \mathcal{C}$ maintains an initially empty list L_S with entries of the form $(ID_i, ID_j, T_i, T_j, sk)$ and answers $\mathcal{A}1$'s queries as follows.

- If $\prod_{i,j}^{t} = \prod_{I,J}^{S}$, then $\mathcal{C}$ returns $T_i = V$ to $\mathcal{A}1$.
- Otherwise, if $ID_i = ID_J$, he generates a random $t_i \in Z_n$, computes $\overline{Z}_1 = Z_1 - t_i(T_j + R_j + H_1(ID_j, R_j)) - s_j(R_i + H_1(ID_i, R_i))$. Then $\mathcal{C}$ checks whether Z_1 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple $(R_i + H_1(ID_i, R_i)P_{pub}, T_j, \overline{Z}_2)$ is inputted. $\mathcal{C}$ also checks whether Z_2 and Z_3 are equal by checking whether the equations $Z_2 = (t_i + x_i)(T_j + P_j)$ and $Z_3 = t_i T_j$ hold separately. If Z_1, Z_2 and Z_3 are correct, $\mathcal{C}$ stores the tuple $(ID_i, ID_j, T_i, T_j, sk)$ into L_S, where the value sk comes from L_{H_2}. Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, sk)$ into L_S.
- Otherwise, $\mathcal{C}$ replies according to the specification of the protocol.

$Test(\prod_{i,j}^{t}): \mathcal{C}$ answers $\mathcal{A}1$'s queries as follows.

- If $\prod_{i,j}^{s} \neq \prod_{I,J}^{S}$, then $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ generates a random number $\xi \in \{0,1\}^k$ and returns it to $\mathcal{A}1$.

As the adversary $\mathcal{A}1$ mounts the forging attack, if $\mathcal{A}1$ succeeds, it must have queried oracle H_2 on the form $Z_1 = (t_I + s_I)(T_J + R_J + H_1(ID_J, R_J)P_{pub}) = (t_I + s_I)(T_J + U)$, $Z_2 = (t_I + x_I)(T_J + P_J)$ and $Z_3 = t_I T_J$, where $T_I = V$ is the outgoing message of *Test* session by the simulator and T_J is the incoming message from the adversary $\mathcal{A}1$. To solve $cdh(U,V)$, for all entries in L_{H_2}, $\mathcal{C}$ randomly chooses one entry with the probability $\frac{1}{n_2}$ and computes

$$\begin{aligned} \overline{Z}_1 &= Z_1 - t_J(T_I + R_I + H_1(ID_I, R_I)) - s_I(R_J + H_1(ID_J, R_J)) \\ &= t_I(R_J + H_1(ID_J, R_J)) = cdh(U,V) \end{aligned} \quad (16)$$

The advantage of $\mathcal{C}$ solving GDH problem with the advantage

$$Adv_{\mathcal{C}}^{GDH}(k) \geq \frac{1}{n_0 n_1^2 n_2} Adv_{\mathcal{A}1}(k). \quad (17)$$

Then $Adv_{\mathcal{C}}^{GDH}(k)$ is non-negligible since we assume that $Adv_{\mathcal{A}1}(k)$ is non-negligible. This contradicts the GDH assumption.

CASE 1.1.2:

Let $Adv_{\mathcal{C}}^{GDH}(k)$ be the advantage that the challenger $\mathcal{C}$ gets in solving the GDH problem given the security parameter k. Given a GDH problem instance($U = uP, V = vP, \mathcal{O}_{ddhp}$) and $\mathcal{C}$'s task is to compute $cdh(U,V) = uvP$, where $\mathcal{O}_{ddhp}$ is a decision oracle that on input (aP, bP, cP), answers 1 if $cdh(aP, bP) = cP$; answers 0, otherwise. $\mathcal{C}$ first chooses $P_0 \in G$ at random, sets P_0 as the system public key P_{pub}, selects the system parameter $params = \{F_p, E/F_p, G, P, P_{pub}, H_1, H_2\}$, and sends $params$ to $\mathcal{A}1$. Then, $\mathcal{C}$ simulates the game outlined in Section 2.3 as follows. Then, $\mathcal{C}$ simulates the game outlined in Section 2.3. During the game, $\mathcal{C}$ simulates $\mathcal{A}1$'s $H_1(ID_i, R_i)$, *RevealMasterKey*, *RevealSecretValue*(i), *ReplacePublicKey*(i, pk), *RevealSessionKey*($\prod_{i,j}^{s}$) and *Test*($\prod_{i,j}^{s}$) queries as that of CASE 1.1.1. $\mathcal{C}$ simulates other oracles as follows.

Create(i): $\mathcal{C}$ simulates the oracle in the same way as that of CASE 1.1 except for $i = I$. If $i = I$, $\mathcal{C}$ chooses two random numbers $h_i, x_i \in Z_n^*$, computes $R_i = V - h_i P_0$, $P_i = x_i P$, sets $H_1(ID_i, R_i) \leftarrow h_i$ and stores ($ID_i, \perp, R_i, x_i, P_i$) and ($ID_i, R_i, h_i$) in L_C and L_{H_1} separately.

$H_2(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, h)$: $\mathcal{C}$ simulates the oracle in the same way as that of CASE 1.1.1 except for the form ($ID_I, ID_J, T_I, T_J, Z_1, Z_2, Z_3$) and ($ID_J, ID_I, T_J, T_I, Z_1, Z_2, Z_3$). $\mathcal{C}$ responds to these queries in the following way:

- If ($ID_I, ID_J, T_I, T_J, Z_1, Z_2, Z_3, h$) or ($ID_J, ID_I, T_J, T_I, Z_1, Z_2, Z_3, h$) is in L_{H_2}, $\mathcal{C}$ responds with the stored value h.
- Otherwise, $\mathcal{C}$ looks up the table L_S for entry ($ID_i, ID_j, T_i, T_j, *$). If there is no such entry, $\mathcal{C}$ choose a random number $h \in \{0,1\}^k$ and

stores the new entry $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, h)$ in L_{H_2}. Otherwise, $\mathcal{C}$ compute $\overline{Z}_1 = Z_1 - t_i(T_j + R_j + H_1(ID_j, R_j)) - t_j(R_i + H_1(ID_i, R_i))$. Then $\mathcal{C}$ checks whether Z_1 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple $(R_i + H_1(ID_i, R_i)P_{pub}, R_j + H_1(ID_j, R_j)P_{pub}, \overline{Z}_1)$ is inputted. $\mathcal{C}$ also checks whether Z_2 and Z_3 are equal by checking if the equations $Z_2 = (t_i + x_i)(T_j + P_j)$ and $Z_3 = t_i T_j$ hold separately. If Z_1, Z_2 and Z_3 are correct, $\mathcal{C}$ stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}, where the value sk comes from L_S. Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.

RevealPartialPrivateKey(i): $\mathcal{C}$ answers $\mathcal{A}1$'s queries as follows.

- If $i = I$ or $i = J$, $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ looks up the list L_C and returns the corresponding partial private key D_i to the adversary $\mathcal{A}1$.

RevealEphemeralKey$(\prod_{i,j}^s)$: $\mathcal{C}$ returns the stored ephemeral private key to $\mathcal{A}1$.

Send$(\prod_{i,j}^s, m)$: $\mathcal{C}$ simulates the oracle in the same way as that of CASE 1.1 except for the following queries:

- If $\prod_{i,j}^s = \prod_{I,J}^S$, $\mathcal{C}$ chooses $t_i \in Z_n$ and returns $T_i = t_i P$ to $\mathcal{A}1$.
- If $i = I$ and $j = J$ (the case that $i = J$ and $j = I$ could be deal with similarly).
  - ◆ $\mathcal{C}$ chooses $t_i \in Z_n$ and returns $T_i = t_i P$ to $\mathcal{A}1$.

  $\mathcal{C}$ looks up the list L_{H_2} for entry $(ID_i, ID_j, T_i, T_j, *, *, *, *)$ (If $\prod_{i,j}^s$ is responder session, $\mathcal{C}$ will look up for $(ID_j, ID_i, T_j, T_i, *, *, *, *)$). If there is no such entry, $\mathcal{C}$ choose a random number $sk \in \{0,1\}^k$ and stores the new entry $(ID_i, ID_j, T_i, T_j, sk)$ in L_S. Otherwise, $\mathcal{C}$ computes $\overline{Z}_1 = Z_1 - t_i(T_j + R_j + H_1(ID_j, R_j)) - t_j(R_i + H_1(ID_i, R_i))$. Then $\mathcal{C}$ checks whether Z_1 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$

outputs 1 when the tuple $(R_i + H_1(ID_i, R_i)P_{pub}, R_j + H_1(ID_j, R_j)P_{pub}, \overline{Z}_1)$ is inputted. $\mathcal{C}$ also checks whether Z_2 and Z_3 are equal by checking if the equations $Z_2 = (t_i + x_i)(T_j + P_j)$ and $Z_3 = t_i T_j$ hold separately. If all of the equations are equal, $\mathcal{C}$ stores $(ID_i, ID_j, T_i, T_j, h)$ into L_S, where h comes from L_{H_2}. Otherwise, $\mathcal{C}$ chooses a random number sk and stores $(ID_i, ID_j, T_i, T_j, sk)$ into L_S.

As the adversary $\mathcal{A}1$ mounts the forging attack, if $\mathcal{A}1$ succeeds, it must have queried oracle H_2 on the form $Z_1 = (t_I + s_I)(T_J + R_J + H_1(ID_J, R_J)P_{pub}) = (t_I + s_I)(T_J + U)$ $Z_2 = (t_I + x_I)(T_J + P_J)$ and $Z_3 = t_I T_J$ where $T_I = t_I P$ is the outgoing message of *Test* session by the simulator $\mathcal{A}1$. To solve $cdh(U,V)$, for all entries in L_{H_2}, $\mathcal{C}$ randomly chooses one entry with the probability $\frac{1}{n_2}$ and computes

$$\begin{aligned} \overline{Z}_1 &= Z_1 - t_I(T_J + R_J + H_1(ID_J, R_J)) - t_J(R_I + H_1(ID_I, R_I)) \\ &= s_I(R_J + H_1(ID_J, R_J)) = s_I U = cdh(U,V) \end{aligned}. \quad (18)$$

We can conclude that

$$Adv_{\mathcal{C}}^{GBCDH}(k) \geq \frac{1}{n_0 n_1^2 n_2} Adv_{\mathcal{A}1}(k). \quad (19)$$

Then $Adv_{\mathcal{C}}^{GBCDH}(k)$ is non-negligible since we assume that $Adv_{\mathcal{A}1}(k)$ is non-negligible. This contradicts the GCDH assumption.

- **The analysis of CASE 1.2**:

In this case, the *Test* session $\prod_{I,J}^{S}$ has a matching session owned by another honest party J. According to Definition 1, the adversary $\mathcal{A}1$ has four ways to mount the attacks.

CASE 1.2.1. The adversary $\mathcal{A}1$ makes ephemeral key query to both the *Test* session and the matching session of the *Test* session (The adversary does not reveal their corresponding partial private key). In this case, the proof is identical to that of CASE 1.1.2. To save space, we omit the details.

CASE 1.2.2. The adversary $\mathcal{A}1$ makes queries to the partial private key of the owner of *Test* session and its peer's ephemeral private key. In this case, the proof is identical to that of CASE 1.1.1. To save space, we omit the details.

CASE 1.2.3. The adversary $\mathcal{A}1$ makes queries to the ephemeral private key of the owner of *Test* session and its peer's partial private key. In this case, the proof is identical to that of CASE 1.1.1. To save space, we omit the details.

CASE 1.2.4. The adversary $\mathcal{A}1$ learns the partial private key of both the owner of *Test* session and its peer. (The adversary does not reveal their corresponding ephemeral private key).

$\mathcal{C}$ answers $H_1(ID_i, R_i)$, *ReplacePublicKey*(i, pk), *RevealSecretValue*(i), *RevealMasterKey* *RevealSessionKey*$(\prod_{i,j}^{t})$ and *Test*$(\prod_{i,j}^{t})$ as he does in the above case. He also answers other queries as follows.

Create(i): $\mathcal{C}$ maintains an initially empty list L_C consisting of tuples of the form $(ID_i, s_i, R_i, x_i, P_i)$. $\mathcal{C}$ chooses three random numbers $s_i, h_i, x_i \in Z_n^*$, computes $R_i = s_i P - h_i P_{pub}$, $P_i = x_i P$, sets $H_1(ID_i, R_i) \leftarrow h_i$ and stores $(ID_i, s_i, R_i, x_i, P_i)$ and (ID_i, R_i, h_i) in L_C and L_{H_1} separately.

$H_2(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$: $\mathcal{C}$ maintains an initially empty list L_{H_2} with entries of the form $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$. If the tuple is in the list L_{H_2}, $\mathcal{C}$ responds with sk. Otherwise, $\mathcal{C}$ responds to these queries in the following way:

- $\mathcal{C}$ looks the list L_S for entry $(ID_i, ID_j, T_i, T_j, *)$. If $\mathcal{C}$ finds the entry, he computes

$$\overline{Z}_1 = Z_1 - s_i(T_j + R_j + H_1(ID_j, R_j)) - s_j T_i \qquad (20)$$

$$\overline{Z}_2 = Z_2 \qquad (21)$$

and

$$\overline{Z}_2 = Z_2 - x_i(T_j + P_j) - x_j T_i \qquad (21)$$

Then $\mathcal{C}$ checks whether Z_i is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple $(T_i, T_j, \overline{Z}_i)$ is inputted, where $i = 1, 2, 3$. If Z_1, Z_2 and Z_3 are correct, $\mathcal{C}$ stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}, where the value sk comes from L_S. Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.

- Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.

RevealPartialPrivateKey(i) : $\mathcal{C}$ looks up the list L_C and returns the corresponding partial private key s_i to the adversary $\mathcal{A}1$.

RevealEphemeralKey$(\prod_{i,j}^s)$: $\mathcal{C}$ answers $\mathcal{A}1$'s queries as follows.

- If $\prod_{i,j}^s = \prod_{I,J}^S$ or $\prod_{i,j}^s = \prod_{J,I}^T$, then $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ returns the stored ephemeral private key to $\mathcal{A}1$.

Send$(\prod_{i,j}^s, m)$: $\mathcal{C}$ maintains an initially empty list L_S with entries of the form $(ID_i, ID_j, T_i, T_j, sk)$ and answers $\mathcal{A}1$'s queries as follows.

- If $\prod_{i,j}^s = \prod_{I,J}^T$, $\mathcal{C}$ returns $T_i = U$ to $\mathcal{A}1$.
- Otherwise, if $\prod_{i,j}^s = \prod_{I,J}^T$, $\mathcal{C}$ returns $T_i = V$ to $\mathcal{A}1$.
- Otherwise, $\mathcal{C}$ replies according to the specification of the protocol.

As the attack that adversary $\mathcal{A}1$ mounts the forging attack, if $\mathcal{A}1$ succeeds, it must have queried oracle H_2 on the form $Z_1 = (t_I + s_I)(T_J + R_J + H_1(ID_J, R_J)P_{pub})$, $Z_2 = (t_I + x_I)(T_J + P_J)$ and $Z_3 = t_I T_J$, where $T_I = U$ is the outgoing message of *Test* session by the simulator and $T_J = V$ is the incoming message from the adversary $\mathcal{A}1$. To solve $cdh(U,V)$, for all entries in L_{H_2}, $\mathcal{C}$ randomly chooses one entry with the probability $\frac{1}{n_2}$ and returns Z_3 as the solution to $cdh(U,V)$.

The advantage of $\mathcal{C}$ solving GDH problem with the advantage

$$Adv_{\mathcal{C}}^{GDH}(k) \geq \frac{1}{n_0 n_1^2 n_2} Adv_{\mathcal{A}1}(k). \qquad (22)$$

Then $Adv_{\mathcal{C}}^{GDH}(k)$ is non-negligible since we assume that $Adv_{\mathcal{A}1}(k)$ is non-negligible. This contradicts the GDH assumption.

We could conclude that the advantage of a type 1 adversary against our protocol is negligible if the GCDH problem is intractable.

Lemma 2. Assuming that the GDH problem is intractable, the advantage of a type 2 adversary against our protocol is negligible.

Proof. Suppose that there is a type 2 adversary $\mathcal{A}2$ who can win the game defined in subsection 2.3 with a non-negligible advantage $Adv_{\mathcal{A}2}(k)$ in polynomial-time t. Then, we will show how to use the ability of $\mathcal{A}2$ to construct an algorithm $\mathcal{C}$ to solve the GDH problem.

Let n_0 be the maximum number of sessions that any one party may have. Assume that the adversary $\mathcal{A}2$ activates at most n_1 distinctive honest parties. Assume that the adversary $\mathcal{A}2$ activates at most n_2 distinctive hash queries. Assume also that $Adv_{\mathcal{A}2}(k)$ is non-negligible. Before the game starts, $\mathcal{C}$ tries to guess the test session and the strategy that the adversary $\mathcal{A}2$ will adopt. $\mathcal{C}$ randomly selects two indexes $I, J \in \{1, \cdots, n_1\}$: $I \neq J$, which represent the I th and the J th distinct honest party that the adversary initially chooses. Also, $\mathcal{C}$ chooses $S \in \{1, \cdots, n_0\}$ and determines the *Test* session $\prod_{I,J}^{S}$, which is correct with probability larger than $\frac{1}{n_0 n_1^2}$. Let $\prod_{J,I}^{T}$ be the matching session of $\prod_{I,J}^{S}$. Since H_1 and H_2 are modeled as random oracles, after the adversary issues the test query, it has only three possible ways to distinguish the tested session key from a random string:

CASE 1: Forging attack: Assume that $\prod_{I,J}^{S}$ is the test session. At some point in its run, the adversary $\mathcal{A}1$ queries H_2 on the value $(ID_I, ID_J, T_I, T_J, K_{IJ}^1, K_{IJ}^2, K_{IJ}^3)$ in the test session owned by I communicating with J. Clearly, in this case $\mathcal{A}2$ computes the values K_{IJ}^1, K_{IJ}^2 and K_{IJ}^3 itself.

CASE 2: Guessing attack: $\mathcal{A}2$ correctly guesses the session key.

CASE 3: Key-replication attack: The adversary $\mathcal{A}2$ forces a non-matching session to have the same session key with the test session. In this case, the adversary $\mathcal{A}2$ can simply learn the session key by querying the non-matching session.

Through the same analysis, we know the success probability of **Key-replication attack** and **Guessing attack** is also negligible. Thus **Guessing attack** and **Key-replication attack** can be ruled out. As the attack that the adversary

$\mathcal{A}2$ mounts is **Forging attack**, $\mathcal{A}2$ can not get an advantage in winning the game against the protocol unless it queries the H_2 oracle on the session key.

In the following, a challenger $\mathcal{C}$ is interested to use the adversary $\mathcal{A}2$ to turn $\mathcal{A}2$'s advantage in distinguishing the tested session key from a random string into an advantage in solving the GDH problem. The following two sub-cases should be considered.

CASE 1.1: No honest party owns a matching session to the *Test* session.

CASE 1.2: The *Test* session has a matching session owned by another honest party.

- **The analysis of CASE 1.1**:

Since $\mathcal{A}2$ is strong type 2 adversary, then he can get any users' partial private key since he is a malicious KGC. According to Definition 2, $\mathcal{C}$ has the following two choices for $\mathcal{A}2$'s strategy:

CASE 1.1.1: At some point, the secret value of party I has been revealed by the adversary $\mathcal{A}2$. According to Definition 2, $\mathcal{A}2$ is not permitted to reveal the ephemeral private key of the *Test* session.

CASE 1.1.2: The secret value of party I has never been revealed by the adversary $\mathcal{A}2$. According to Definition 2, $\mathcal{A}2$ may reveal the ephemeral private key of the *Test* session.

CASE 1.1.1:

Let $Adv_{\mathcal{C}}^{GDH}(k)$ be the advantage that the challenger $\mathcal{C}$ gets in solving the GDH problem given the security parameter k. Given a GDH problem instance $(U = uP, V = vP, \mathcal{O}_{ddhp})$ and $\mathcal{C}$'s task is to compute $cdh(U,V) = uvP$, where $\mathcal{O}_{ddhp}$ is a decision oracle that on input (aP, bP, cP), answers 1 if $cdh(aP, bP) = cP$; answers 0, otherwise. $\mathcal{C}$ first chooses a random number $x \in Z_n^*$, sets xP as the system public key P_{pub}, selects the system parameter $params = \{F_p, E/F_p, G, P, P_{pub}, H_1, H_2\}$, and sends $params$ to $\mathcal{A}2$. Then, $\mathcal{C}$ simulates the game outlined in Section 2.3 as follows.

Create(i): $\mathcal{C}$ maintains an initially empty list L_C consisting of tuples of the form $(ID_i, s_i, R_i, x_i, P_i)$. If $i = J$, $\mathcal{C}$ chooses two random numbers $r_i \in Z_n^*$, computes $R_i = r_i P$, $h_i = H_1(ID_i, R_i)$, $s_i = r_i + h_i x$, $P_i = U$ and stores $(ID_i, s_i, R_i, \perp, P_i)$in L_C. Otherwise, $\mathcal{C}$ chooses two random numbers $r_i, x_i \in Z_n^*$,

computes $R_i = r_i P$, $h_i = H_1(ID_i, R_i)$, $s_i = r_i + h_i x$, $P_i = x_i P$ and stores $(ID_i, s_i, R_i, x_i, P_i)$ in L_C.

$H_1(ID_i, R_i)$: $\mathcal{C}$ maintains an initially empty list L_{H_1} which contains tuples of the form (ID_i, R_i, h_i). If (ID_i, R_i) is on the list L_{H_1}, $\mathcal{C}$ returns h_i. Otherwise, $\mathcal{C}$ chooses a random number h_i, stores (ID_i, R_i, h_i) in L_{H_1} and returns h_i.

$H_2(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$: $\mathcal{C}$ maintains an initially empty list L_{H_2} with entries of the form $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$. If the tuple is in the list L_{H_2}, $\mathcal{C}$ responds with sk. Otherwise, $\mathcal{C}$ responds to these queries in the following way:

- If $ID_i = ID_J$,
  - $\mathcal{C}$ looks the list L_S for entry $(ID_i, ID_j, T_i, T_j, *)$. If $\mathcal{C}$ finds the entry, he computes $\overline{Z}_2 = Z_2 - t_i(T_j + P_j) - x_j P_i$.
  - Then $\mathcal{C}$ checks whether Z_2 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple $(P_i, T_j, \overline{Z}_2)$ is inputted. $\mathcal{C}$ also checks whether Z_1 and Z_3 are equal by checking whether the equations $Z_1 = (t_i + s_i)(T_j + R_j + H_1(ID_j, R_j))$ and $Z_3 = t_i T_j$ hold separately. If Z_1, Z_2 and Z_3 are correct, $\mathcal{C}$ stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}, where the value sk comes from L_S. Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.
- Otherwise,
  - $\mathcal{C}$ looks the list L_S for entry $(ID_i, ID_j, T_i, T_j, *)$. If $\mathcal{C}$ finds the entry, he stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}, where the value sk comes from L_S.
  - Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.

RevealPartialPrivateKey(i): $\mathcal{C}$ looks up the list L_E and returns the corresponding partial private key s_i to the adversary $\mathcal{A}2$.

RevealSecretValue(i): $\mathcal{C}$ answers $\mathcal{A}2$'s queries as follows.

- If $ID_i = ID_J$, then $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ looks up the table L_C for entry $(ID_i,*,*,*,*)$ and returns x_i.

RevealEphemeralKey$(\prod_{i,j}^{t})$: $\mathcal{C}$ answers $\mathcal{A}2$'s queries as follows.

- If $\prod_{i,j}^{s} = \prod_{I,J}^{S}$, then $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ returns the stored ephemeral private key to $\mathcal{A}2$.

RevealMasterKey: $\mathcal{C}$ returns the master key x to $\mathcal{A}2$.

RevealSessionKey$(\prod_{i,j}^{t})$: $\mathcal{C}$ answers $\mathcal{A}2$'s queries as follows.

- If $\prod_{i,j}^{s} = \prod_{I,J}^{S}$ or $\prod_{i,j}^{s} = \prod_{I,J}^{T}$, then $\mathcal{C}$ stops the simulation.
- Otherwise, if $\mathcal{C}$ returns the session key sk to $\mathcal{A}2$.

Send$(\prod_{i,j}^{s}, m)$: $\mathcal{C}$ maintains an initially empty list L_S with entries of the form $(ID_i, ID_j, T_i, T_j, sk)$ and answers $\mathcal{A}2$'s queries as follows.

- If $\prod_{i,j}^{s} = \prod_{I,J}^{S}$, then $\mathcal{C}$ returns $T_i = V$ to $\mathcal{A}2$.
- Otherwise, if $ID_i = ID_J$, he generates a random $t_i \in Z_n$, computes $\overline{Z}_2 = Z_2 - t_i(T_j + P_j) - x_j P_i$. Then $\mathcal{C}$ checks whether Z_2 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple $(P_i, T_j, \overline{Z}_2)$ is inputted. $\mathcal{C}$ also checks whether Z_1 and Z_3 are equal by checking whether the equations $Z_1 = (t_i + s_i)(T_j + R_j + H_1(ID_j, R_j))$ and $Z_3 = t_i T_j$ hold separately. If Z_1, Z_2 and Z_3 are correct, $\mathcal{C}$ stores the tuple $(ID_i, ID_j, T_i, T_j, sk)$ into L_S, where the value sk comes from L_{H_2}. Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, sk)$ into L_S.
- Otherwise, $\mathcal{C}$ replies according to the specification of the protocol.

Test$(\prod_{i,j}^{t})$: $\mathcal{C}$ answers $\mathcal{A}2$'s queries as follows.

- If $\prod_{i,j}^{t} \neq \prod_{I,J}^{S}$, then $\mathcal{C}$ stops the simulation.
- Otherwise, $\mathcal{C}$ generates a random number $\xi \in \{0,1\}^k$ and returns it to $\mathcal{A}2$.

As the adversary $\mathcal{A}2$ mounts the forging attack, if $\mathcal{A}2$ succeeds, it must have queried oracle H_2 on the form $Z_1 = (t_I + s_I)(T_J + R_J + H_1(ID_J, R_J)P_{pub})$, $Z_2 = (t_I + x_I)(T_J + U)$ and $Z_3 = t_I T_J$, where $T_I = V$ is the outgoing message of *Test* session by the simulator and T_J is the incoming message from the adversary $\mathcal{A}2$. To solve $GDH(U,V)$, for all entries in L_{H_2}, $\mathcal{C}$ randomly chooses one entry with the probability $\frac{1}{n_2}$ and computes

$$\overline{Z}_2 = Z_2 - x_I(T_J + U)) - Z_3 \tag{23}$$

It is easy to verify that the equation $\overline{Z}_2 = cdh(U,V)$ holds. The advantage of $\mathcal{C}$ solving GDH problem with the advantage

$$Adv_{\mathcal{C}}^{GDH}(k) \geq \frac{1}{n_0 n_1^2 n_2} Adv_{\mathcal{A}2}(k). \tag{24}$$

Then $Adv_{\mathcal{C}}^{GDH}(k)$ is non-negligible since we assume that $Adv_{\mathcal{A}2}(k)$ is non-negligible. This contradicts the GDH assumption.

CASE 1.1.2:

$\mathcal{C}$ answers $H_1(ID_i, R_i)$, *RevealPartialPrivateKey*(i), *RevealEphemeralKey*$(\prod_{i,j}^t)$, *RevealMasterKey*, *RevealSessionKey*$(\prod_{i,j}^t)$ and *Test*$(\prod_{i,j}^t)$ as he does in CASE 3.1.3 of Lemma 3. He also answers other queries as follows.

Create(i): $\mathcal{C}$ simulates the oracle in the same way as that of CASE 1.1.1 except for $i = I$. If $i = I$, $\mathcal{C}$ chooses two random numbers $r_i \in Z_n^*$, computes $R_i = r_i P$, $h_i = H_1(ID_i, R_i)$, $s_i = r_i + h_i x$, $P_i = V$ and stores $(ID_i, s_i, R_i, \perp, P_i)$in L_C. Otherwise, $\mathcal{C}$ chooses two random numbers $r_i, x_i \in Z_n^*$, computes $R_i = r_i P$, $h_i = H_1(ID_i, R_i)$, $s_i = r_i + h_i x$, $P_i = x_i P$ and stores $(ID_i, s_i, R_i, x_i, P_i)$ in L_C.

$H_2(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, h)$: $\mathcal{C}$ simulates the oracle in the same way as that of CASE 1.1.1 except for the form $(ID_I, ID_J, T_I, T_J, Z_1, Z_2, Z_3)$ and $(ID_J, ID_I, T_J, T_I, Z_1, Z_2, Z_3)$. $\mathcal{C}$ responds to these queries in the following way:

- If $(ID_I, ID_J, T_I, T_J, Z_1, Z_2, Z_3, h)$ or $(ID_J, ID_I, T_J, T_I, Z_1, Z_2, Z_3, h)$ is in L_{H_2}, $\mathcal{C}$ responds with the stored value h.
- Otherwise, $\mathcal{C}$ looks up the table L_S for entry $(ID_i, ID_j, T_i, T_j, *)$. If there is no such entry, $\mathcal{C}$ choose a random number $h \in \{0,1\}^k$ and stores the new entry $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, h)$ in L_{H_2}. Otherwise, $\mathcal{C}$ compute $\overline{Z}_2 = Z_2 - t_i(T_j + P_j) - t_j P_i$. Then $\mathcal{C}$ checks whether Z_2 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple $(P_i, P_j, \overline{Z}_1)$ is inputted. $\mathcal{C}$ also checks whether Z_1 and Z_3 are equal by checking if the equations $Z_1 = (t_i + s_i)(T_j + R_j + H_1(ID_j, R_j)P_{pub})$ and $Z_3 = t_i T_j$ hold separately. If Z_1, Z_2 and Z_3 are correct, $\mathcal{C}$ stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}, where the value sk comes from L_S. Otherwise, $\mathcal{C}$ chooses a random number $sk \in \{0,1\}^k$ and stores the tuple $(ID_i, ID_j, T_i, T_j, Z_1, Z_2, Z_3, sk)$ into L_{H_2}.

RevealSecretValue(i): : $\mathcal{C}$ simulates the oracle in the same way as that of CASE 1.1.1 except for $i = I$. If $i = I$, $\mathcal{C}$ stops the simulation.

Send$(\prod_{i,j}^s, m)$: $\mathcal{C}$ simulates the oracle in the same way as that of CASE 1.1 except for the following queries:

- If $\prod_{i,j}^s = \prod_{I,J}^S$, $\mathcal{C}$ chooses $t_i \in Z_n$ and returns $T_i = t_i P$ to $\mathcal{A}1$.
- If $i = I$ and $j = J$ (the case that $i = J$ and $j = I$ could be deal with similarly).
  - ◆ $\mathcal{C}$ chooses $t_i \in Z_n$ and returns $T_i = t_i P$ to $\mathcal{A}1$.

  $\mathcal{C}$ looks up the list L_{H_2} for entry $(ID_i, ID_j, T_i, T_j, *, *, *, *)$ (If $\prod_{i,j}^s$ is responder session, $\mathcal{C}$ will look up for $(ID_j, ID_i, T_j, T_i, *, *, *, *)$). If there is no such entry, $\mathcal{C}$ choose a random number $sk \in \{0,1\}^k$ and stores the new entry $(ID_i, ID_j, T_i, T_j, sk)$ in L_S. Otherwise, $\mathcal{C}$ computes $\overline{Z}_2 = Z_2 - t_i(T_j + P_j) - t_j P_i$. Then $\mathcal{C}$ checks whether Z_2 is correct by checking whether the oracle $\mathcal{O}_{ddhp}$ outputs 1 when the tuple $(P_i, P_j, \overline{Z}_1)$ is inputted. $\mathcal{C}$ also checks whether Z_2 and Z_3 are equal by checking if

the equations $Z_1 = (t_i + s_i)(T_j + R_j + H_1(ID_j, R_j)P_{pub})$ and $Z_3 = t_i T_j$ hold separately. If all of the equations are equal, $\mathcal{C}$ stores $(ID_i, ID_j, T_i, T_j, h)$ into L_S, where h comes from L_{H_2}. Otherwise, $\mathcal{C}$ chooses a random number sk and stores $(ID_i, ID_j, T_i, T_j, sk)$ into L_S.

As the adversary $\mathcal{A}2$ mounts the forging attack, if $\mathcal{A}2$ succeeds, it must have queried oracle H_2 on the form $Z_1 = (t_I + s_I)(T_J + R_J + H_1(ID_J, R_J)P_{pub})$, $Z_2 = (t_I + x_I)(T_J + U)$ and $Z_3 = t_I T_J$, where $P_I = U$, $P_J = V$ and T_J is the incoming message from the adversary $\mathcal{A}2$. To solve $GDH(U,V)$, for all entries in L_{H_2}, $\mathcal{C}$ randomly chooses one entry with the probability $\frac{1}{n_2}$ and proceeds with following steps:

$\mathcal{C}$ computes

$$\overline{Z}_2 = Z_2 - t_I(T_J + U) - t_J V = cdh(U,V) \qquad (25)$$

The advantage of $\mathcal{C}$ solving GDH problem with the advantage

$$Adv_{\mathcal{C}}^{GDH}(k) \geq \frac{1}{n_0 n_1^2 n_2} Adv_{\mathcal{A}2}(k).$$

Then $Adv_{\mathcal{C}}^{GDH}(k)$ is non-negligible since we assume that $Adv_{\mathcal{A}2}(k)$ is non-negligible. This contradicts the GDH assumption.

- **The analysis of CASE 1.2**:

In this case, the *Test* session $\prod_{I,J}^{S}$ has a matching session owned by another honest party J. According to Definition 1, the adversary $\mathcal{A}1$ has four ways to mount the attacks.

CASE 1.2.1. The adversary $\mathcal{A}1$ makes ephemeral key query to both the *Test* session and the matching session of the *Test* session (The adversary does not reveal their corresponding partial private key). In this case, the proof is identical to that of CASE 1.1.2. To save space, we omit the details.

CASE 1.2.2. The adversary $\mathcal{A}1$ makes queries to the partial private key of the owner of *Test* session and its peer's ephemeral private key. In this case, the proof is identical to that of CASE 1.1.1. To save space, we omit the details.

CASE 1.2.3. The adversary $\mathcal{A}1$ makes queries to the ephemeral private key of the owner of *Test* session and its peer's partial private key. In this case, the proof is identical to that of CASE 1.1.1. To save space, we omit the details.

CASE 1.2.4. The adversary $\mathcal{A}1$ learns the partial private key of both the owner of *Test* session and its peer. (The adversary does not reveal their corresponding ephemeral private key). In this case, the proof is identical to that of CASE 1.2.4 of the above lemma. To save space, we omit the details.

We could conclude that the advantage of a type 2 adversary against our protocol is negligible if the GCDH problem is intractable.

From the above three lemmas, we can get the following theorem.

Theorem 1. Our protocol is a secure CLAKA protocol in the eCK model under the GDH assumption.

5. Comparison with previous protocols

Let mBR and eCK denote the modified Bellare-Rogaway model [17] and the extended Canetti–Krawczyk (eCK) model [18] separately. For the convenience of evaluating the computational cost, we define some notations as follows.

- T_{mul} : The time of executing a scalar multiplication operation of point.
- T_{add} : The time of executing an addition operation of point.
- T_{inv} : The time of executing a modular invasion operation.
- T_h : The time of executing a one-way hash function.

We will compare the efficiency of our protocol with five CLAKA protocols without pairings, i.e. Geng et al.'s protocol [12], Hou et al.'s protocol [13], Yang et al.'s protocol[14], and He et al.'s protocols[15,16]. Table 1 shows the comparison between pairing-free CLAKA protocols in terms of efficiency, security model and underlying hardness assumptions.

Since the scalar multiplication operation of point is more complicated than the addition operation of points, modular invasion operation and the hash function operation, then our protocol has better performance than Geng et al.'s protocol [12], Hou et al.'s protocol [13] and He et al's protocol[15]. Moreover, Geng et al.'s protocol [12], Hou et al.'s protocol [13] and He et al's protocol[15] are not secure against type 1 adversary. Then our protocol has advantage in both the performance and the security over Geng et al.'s protocol [12], Hou et al.'s protocol [13] and He et al's protocol [15]. It is well known that the eCK model is much superior to the mBR model. Then Yang et al.'s protocol [15] and our protocol has advantage in security to He et al.'s protocol [16]. At the same time,

our protocol also has better performance than He et al.'s protocol [16]. Yang et al.'s proposed the first pairing-free CLAKA protocol, which is provably secure in the eCK model. However, in Yang et al.'s protocol, the user has to verify the validity of public keys. This does not only increase the burden of the user, but also reverse the thought of CLPKC. From Table 1, we know our protocol has much better performance than Yang et al.'s protocol [15]. We conclude that our protocol is more suitable for practical applications.

Table 1: Comparisons among different protocols

	Computational cost	Security model	Assumption	Message exchange
Geng et al.'s protocol [12]	$7T_{mul} + 2T_h$	mBR	GDH	2
Hou et al.'s protocol [13]	$6T_{mul} + 2T_h$	mBR	GDH	2
Yang et al.'s protocol [14]	$9T_{mul} + 2T_h$	eCK	GDH	2
He et al's protocol[15]	$5T_{mul} + 3T_{add} + T_{inv} + 2T_h$	mBR	GDH	3
He et al's protocol[16]	$5T_{mul} + 4T_{add} + 2T_h$	mBR	GDH	2
Our protocol	$5T_{mul} + 3T_{add} + 2T_h$	eCK	GDH	2

6. Conclusion

The certificateless public key cryptography is receiving significant attention because it is a new paradigm that simplifies the public key cryptography. Recently, several pairing-free CLAKA have been proposed. In this paper, we proposed a more efficient CLAKA protocol without pairings and proved its security in the eCK model under the GDH assumption. The proposed protocol has the best performance among the related protocols.

Acknowledgements

The authors thank Prof. Ervin Y. Rodin and the anonymous reviewers for their valuable comments. This research was supported by the Fundamental Research

Funds for the Central Universities and the Specialized Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20110141120003).

References

[1] K.Y. Choi, J.H. Park, D.H. Lee, A new provably secure certificateless short signature scheme, Computers and Mathematics with Applications 61(7)(2011) 1760-1768.

[2] A. Shamir, Identity-based cryptosystems and signature protocols, Proc. CRYPTO1984, LNCS, vol.196, 1984, pp.47–53.

[3] S. Al-Riyami, K.G. Paterson, Certificateless public key cryptography, Proceedings of ASIACRYPT 2003, LNCS 2894, Springer-Verlag, 2003, pp. 452 – 473.

[4] Z. Shao. Efficient authenticated key agreement protocol using self-certifed public keys from pairings. Wuhan University Journal of Natural Sciences, 10(1) (2005) 267-270.

[5] S. Wang, Z. Cao, X. Dong, Certificateless authenticated key agreement based on the MTI/CO protocol, Journal of Information and Computational Science 3 (2006) 575 – 581.

[6] T. Mandt, C. Tan, Certificateless authenticated two-party key agreement protocols, in: Proceedings of the ASIAN 2006, LNCS, vol. 4435, Springer-Verlag, 2008, pp. 37 – 44.

[7] Y. Shi, J. Li, Two-party authenticated key agreement in certificateless public key cryptography, Wuhan University Journal of Natural Sciences 12 (1) (2007) 71 – 74.

[8] C. Swanson. Security in key agreement: Two-party certificateless protocols. Master Thesis, University of Waterloo, 2008.

[9] G. Lippold, C. Boyd, J. Nieto. Strongly secure certificateless key agreement. In Pairing 2009, 2009, pp. 206-230.

[10] L. Zhang, F. Zhang, Q. Wua, J. Domingo-Ferrer, Simulatable certificateless two-party authenticated key agreement protocol, Information Sciences 180 (2010) 1020 – 1030.

[11] L. Chen, Z. Cheng, and N.P. Smart, Identity-based key agreement protocols from pairings, International Journal Information Security 6 (2007) 213–241.

[12] M. Geng and F. Zhang. Provably secure certificateless two-party authenticated key agreement protocol without pairing. In International Conference on Computational Intelligence and Security, 2009, pp. 208-212.

[13] M. Hou and Q. Xu. A two-party certificateless authenticated key agreement protocol without pairing. In 2nd IEEE International Conference on Computer Science and Information Technology, 2009, pp. 412-416.

[14] G. Yang, C. Tan, Strongly secure certificateless key exchange without pairing, 6th ACM Symposium on Information, Computer and Communications Security, 2011, pp. 71-79.

[15] D. He, J. Chen, J. Hu, A pairing-free certificateless authenticated key agreement protocol, International Journal of Communication Systems, (In press) DOI: 10.1002/dac.1265, 2011.

[16] D. He, Y. Chen, J. Chen, R. Zhang, W. Han, A new two-round certificateless authenticated key agreement protocol without bilinear pairings, Mathematical and Computer Modelling (2011), doi:10.1016/j.mcm.2011.08.004

[17] M. Bellare, P. Rogaway. Entity authentication and key distribution. In: Proceedings of the CRYPTO 1993. LNCS, vol. 773. Springer-Verlag; 1993. p. 232–49.

[18] B. LaMacchia, K. Lauter, A. Mityagin. Stronger security of authenticated key exchange. In: Proceedings of the ProvSection 2007. LNCS, vol. 4784. Springer-Verlag; 2007. p. 1–16.

[19] M. Bellare and P. Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, in Proc. 1st ACM Conf. Comput. Commun. Security, 1993, pp. 62–73.