An efficient certificateless two-party authenticated key agreement scheme from pairings

Debiao He*, Jin Hu

School of Mathematics and Statistics, Wuhan University, Wuhan, 430072, China

Abstract*:* Key agreement (KA) allows two or more users to negotiate a secret session key among them over an open network. Authenticated key agreement (AKA) is a KA protocol enhanced to prevent active attacks. AKA can be achieved using a public-key infrastructure (PKI) or identity-based cryptography. However, the former suffers from a heavy certificate management burden while the latter is subject to the so-called key escrow problem. Recently, certificateless cryptography was introduced to mitigate these limitations. We propose an efficient certificateless two-party AKA protocol. Security is proven under the standard computational Diffie-Hellman (CDH) and bilinear Diffie-Hellman (BDH) assumptions. Our protocol is efficient and practical, because it requires only one pairing operation and three scale multiplications by each party. Moreover, the pairing operation and one scale multiplication scale can be pre-computed, then only two scale multiplications are needed to finished the key agreement.

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

1. Introduction

Public key cryptography is an important technique to realize network and information security. Traditional public key infrastructure requires a trusted certification authority to issue a certificate binding the identity and the public key of an entity. Hence, the problem of certificate management arises. To solve the problem, Shamir defined a new public key paradigm called identity-based public key cryptography [1]. However, identity-based public key cryptography needs a

*Corresponding author.

E-mail: hedebiao@163.com

trusted Private Key Generator(PKG) to generate a private key for an entity according to his identity. So we are confronted with the key escrow problem. Fortunately, the two problems in traditional public key infrastructure and identity-based public key cryptography can be prohibited by introducing certificateless public key cryptography (CLPKC) [2], which can be conceived as an intermediate between traditional public key infrastructure and identity-based cryptography.

Key agreement(KA) schemes are designed to provide secure communications between two or more parties in a hostile environment. A two-party key agreement scheme, for example, allows two communicating parties to establish a common session key via a public communication channel. The session key can subsequently be used to establish a secure communication channel between both parties.

KA schemes can also be implemented in the certificateless cryptographic setting. Following the pioneering work due to Al-Riyami and Paterson , several certificateless two-party authenticated key agreement(CTAKA) schemes [3-8] have been proposed. However, these schemes need to compute at least one pairing on-line. In order to improve the performance, we present an efficient CTAKA scheme from pairings. In our scheme the pairing operation and one scale multiplication scale can be pre-computed, then only two scale multiplications are needed to finished the key agreement. Our scheme’s overhead is lower than that of previous schemes [3-8] in computation and more suitable for the practical applications.

2. Preliminaries

2.1.Mathematical background

Let G_1 be a cyclic additive group of prime order q, and G_2 be a cyclic multiplicative group of the same order q. We let P denote the generator of G_1. A bilinear pairing is a map $e: G_1 \times G_1 \to G_2$ which satisfies the following properties:

(1) Bilinearity

$e(aQ, bR) = e(Q, R)^{ab}$, where $Q, R \in G_1$, $a, b \in Z_q^*$.

(2) Non-degeneracy

$e(P, P) \neq 1_{G_2}$.

(3) Computability

There is an efficient algorithm to compute $e(Q, R)$ for all $Q, R \in G_1$.

The Weil and Tate pairings associated with supersingular elliptic curves or abelian varieties can be modified to create such admissible pairings, as in [9]. The following problems are assumed to be intractable within polynomial time.

Definition 1 (**Bilinear Diffie-Hellman (BDH) problem**). Let G_1, G_2, P and e be as above. The BDH problem in $< G_1, G_2, e >$ is as follows: Given $< P, aP, bP, cP >$ with uniformly random choices of $a, b, c \in Z_q^*$, compute $e(P, P)^{abc} \in G_2$.

Definition 2 (**Computational Diffie-Hellman (CDH) problem**). Let G_1 and P be as above. The CDH problem in G_1 is as follows: Given $< P, aP, bP >$ with uniformly random choices of $a, b \in Z_q^*$, compute $abP \in G_1$.

2.2. CTAKA scheme

A CTAKA scheme consists of six polynomial-time algorithms[8]: *Setup*, *Partial-Private-Key-Extract*, *Set-Secret-Value*, *Set-Private-Key*, *Set-Public-Key* and *Key-Agreement*. These algorithms are defined as follows.

Setup: Taking security parameter k as input and returns the system parameters *params* and master key.

Partial-Private-Key-Extract: It takes *params*, master key and a user's identity ID_i as inputs. It returns a partial private key D_i.

Set-Secret-Value: Taking as inputs *params* and a user's identity ID_i, this algorithm generates a secret value x_i.

Set-Private-Key: This algorithm mtakes *params*, a user's partial private key D_i and his secret value x_i as inputs, and outputs the full private key S_i.

Set-Public-Key: Taking as inputs *params* and a user's secret value x_i, and generates a public key P_i for the user.

Key-Agreement: This is a probabilistic polynomial-time interactive algorithm which involves two entities A and B. The inputs are the system parameters *params* for both A and B, plus (S_A, ID_A, P_A) for A, and (S_B, ID_B, P_B) for B. Here, S_A, S_B are the respective private keys of A and B; ID_A is the identity of A and ID_B is the identity of B; P_A, P_B are the public keys of A and B, respectively. Eventually, if the scheme does not fail, A and B obtain a secret session key $K_{AB} = K_{BA} = K$.

2.3.Security model for CTAKA schemes

In CTAKA, as defined in [2], there are two types of adversaries with different capabilities, we assume *Type 1 Adversary*, $\mathscr{A}1$ acts as a dishonest user while *Type 2 Adversary*, $\mathscr{A}2$ acts as a malicious KGC:

Type 1 Adversary: Adversary $\mathscr{A}1$ does not have access to the master key, but $\mathscr{A}1$ can replace the public keys of any entity with a value of his choice, since there is no certificate involved in CLPKC.

Type 2 Adversary: Adversary $\mathscr{A}2$ has access to the master key, but cannot replace any user's public key.

Very recently, Zhang et al.'s [8] present a security model for AKA schemes in the setting of CLPKC. The model is defined by the following game between a challenger $\mathcal{C}$ and an adversary $\mathscr{A} \in \{\mathscr{A}1, \mathscr{A}2\}$. In their et al.'s model, $\mathscr{A}$ is modeled by a probabilistic polynomial-time turing machine. All communications go through the adversary $\mathscr{A}$. Participants only respond to the queries by $\mathscr{A}$ and do not communicate directly among themselves. $\mathscr{A}$ can relay, modify, delay, interleave or delete all the message flows in the system. Note that $\mathscr{A}$ can act as a benign adversary, which means that $\mathscr{A}$ is deterministic and restricts her action to choosing a pair of oracles $\prod_{i,j}^{n}$ and $\prod_{j,i}^{t}$ and then faithfully conveying each message flow from one oracle to the other. Furthermore, $\mathscr{A}$ may ask a polynomially bounded number of the following queries as follows.

$Create(ID_i)$: This allows $\mathscr{A}$ to ask $\mathcal{C}$ to set up a new participant i with identity ID_i. On receiving such a query, $\mathcal{C}$ generates the public/private key pair for i.

$Public-Key(ID_i)$: $\mathscr{A}$ can request the public key of a participant i whose identity is ID_i. To respond, $\mathcal{C}$ outputs the public key P_i of participant i.

$Partial\text{-}Private\text{-}Key(ID_i)$: $\mathscr{A}$ can request the partial private key of a participant i whose identity is ID_i. To respond, $\mathcal{C}$ outputs the partial private key D_i of participant i.

$Corrupt(ID_i)$: $\mathscr{A}$ can request the private key of a participant i whose identity is ID_i. To respond, $\mathcal{C}$ outputs the private key S_i of participant i.

$Public-Key-Replacement(ID_i, P_i')$: For a participant i whose identity is ID_i; $\mathscr{A}$ can choose a new public key P' and then set P' as the new public key of this participant. $\mathcal{C}$ will record these replacements which will be used later.

$Send(\prod_{i,j}^{n}, M)$: $\mathscr{A}$ can send a message M of her choice to an oracle, say $\prod_{i,j}^{n}$, in which case participant i assumes that the message has been sent by participant j. $\mathscr{A}$ may also make a special *Send* query with $M \neq \lambda$ to an oracle $\prod_{i,j}^{n}$, which instructs i to initiate a scheme run with j. An oracle is an initiator oracle if the first message it has received is λ. If an oracle does not receive a message λ as its first message, then it is a responder oracle.

$Reveal(\prod_{i,j}^{n})$: $\mathscr{A}$ can ask a particular oracle to reveal the session key (if any) it currently holds to $\mathscr{A}$.

$Test(\prod_{i,j}^{n})$: At some point, $\mathscr{A}$ may choose one of the oracles, say $\prod_{I,J}^{T}$, 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 $\prod_{I,J}^{T}$ if $b=0$, or a random sample from the distribution of the session key if $b=1$.

After a *Test* query, the adversary can continue to query the oracles except that it cannot make a Reveal query to the test oracle $\prod_{I,J}^{T}$ or to $\prod_{J,I}^{t}$ who has a matching conversation with $\prod_{I,J}^{T}$ (if it exists), and it cannot corrupt participant J. In addition, if $\mathscr{A}$ is a Type 1 adversary, $\mathscr{A}$ cannot request the partial private key of the participant J; and if $\mathscr{A}$ is a Type 2 adversary, J cannot replace the public key of the participant J. At the end of the game, $\mathscr{A}$ 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 $Advantage^A(k)$, is defined as: $Advantage^A(k) = \left|\Pr[b' - b] - \frac{1}{2}\right|$.

Definition 1. A CTAKA scheme is said to be secure if:

(1) In the presence of a benign adversary on $\prod_{i,j}^{n}$ 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, $Advantage^A(k)$ is negligible.

3. Our scheme

3.1.Scheme Description

In this section, we present a CTAKA scheme without pairing. Our scheme consists of six algorithms: *Setup*, *Partial-Private-Key-Extract*, *Set-Secret-Value*, *Set-Private-Key*, *Set-Public-Key* and *Key-Agreement*.

Setup: On input a security parameter l, this algorithm runs as follows.

(1) Select a cyclic additive group G_1 of prime order q, a cyclic multiplicative group G_2 of the same order, a generator P of G_1, and a bilinear map $e: G_1 \times G_1 \rightarrow G_2$.

(2) Choose a random master-key $s \in Z_q^*$ and set the master public key $P_{pub} = sP$.

(3) Choose cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow G_1$, $H_2 : \{0,1\}^* \times \{0,1\}^* \times G_1 \times G_1 \times G_1 \times G_2 \times G_1 \rightarrow \{0,1\}^l$.

The system parameters are $params = \{G_1, G_2, e, P, P_{pub}, H_1, H_2, l\}$. The master-key is $s \in Z_q^*$.

Partial-Private-Key-Extract: This algorithm takes system parameters, master key and a user's identifier ID_i as inputs, generates the partial private key as follows.

1) Choose $Q_i = H_1(ID_i)$.
2) Output the partial private key $D_i = sQ_i$.

Set-Secret-Value: The user with identity ID_i picks randomly $x_i \in Z_n^*$ sets x_i as his secret value.

Set-Private-Key: Given *params*, the user's partial private key D_i and his secret value x_i, and output a pair $S_i = (x_i, D_i)$ as the user's private key.

Set-Public-Key: Taking as inputs *params*, the user's secret value x_i, and generates the user's public key as $P_i = x_i \cdot P$.

Key-Agreement: Assume that an entity A with identity ID_A has private key $S_A = (x_A, D_A)$ and public key $P_A = x_A \cdot P$ and an entity B with identity ID_B has private key $S_B = (x_B, D_B)$ and public key $P_B = x_B \cdot P$ want to establish a session key, then they can do, as shown in Fig.1, as follows.

1) A chooses at random the ephemeral key $a \in Z_n^*$ and computes $T_A = a \cdot P$, then A send $M_1 = \{ID_A, T_A\}$ to B.

2) After receiving M_1, B chooses at random the ephemeral key $b \in Z_n^*$ and computes $T_B = b \cdot P$, then B send $M_2 = \{ID_B, T_B\}$ to A.

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

A computes

$$K_{AB}^1 = x_A \cdot P_B,\ K_{AB}^2 = e(D_A, Q_B) \text{ and } K_{AB}^3 = a \cdot T_A \tag{1}$$

B computes

$$K_{BA}^1 = x_B \cdot P_A, K_{BA}^2 = e(D_B, Q_A) \text{ and } K_{BA}^3 = b \cdot T_B \tag{2}$$

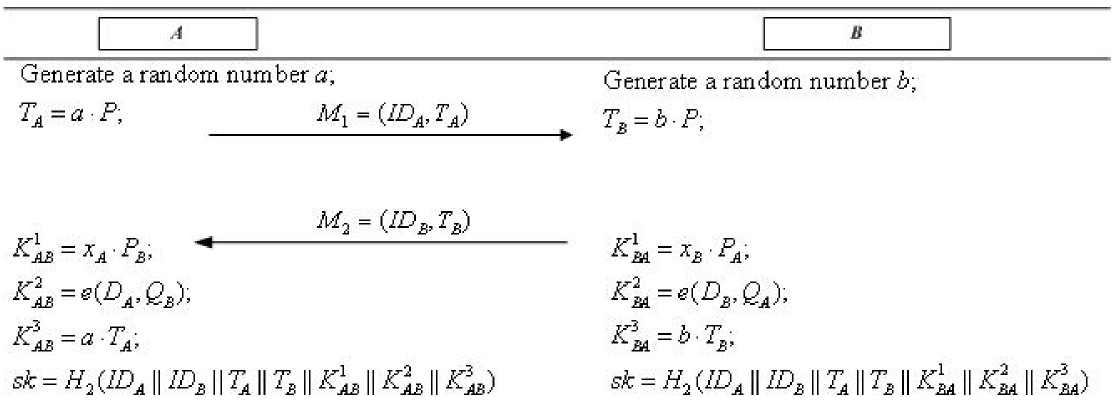

Fig. 1. Key agreement of our scheme

The shared secrets agree because:

$$K_{AB}^1 = x_A \cdot P_B = x_A \cdot x_B \cdot P = x_B \cdot x_A \cdot P = x_B \cdot P_A = K_{BA}^1 \tag{3}$$

$$K_{AB}^2 = e(D_A, Q_B) = e(sQ_A, Q_B) = e(Q_A, Q_B)^s$$
$$= e(Q_A, sQ_B) = e(D_B, Q_A) = K_{BA}^2 \quad (4)$$

and

$$K_{AB}^3 = abP = baP = K_{BA}^3 \quad (5)$$

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

$$sk = H_2(ID_A \| ID_B \| T_A \| T_B \| K_{AB}^1 \| K_{AB}^2 \| K_{AB}^3)$$
$$= H_2(ID_A \| ID_B \| T_A \| T_B \| K_{BA}^1 \| K_{BA}^2 \| K_{BA}^3) \quad (6)$$

3.2.Security Analysis

We prove the security of our scheme in the random oracle model which treats H_1 and H_2 as two random oracles [10] using the model defined in [8]. As for the security of , the following lemmas and theorems are provided.

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

Proof. From the correction analysis of our scheme in section 3.1, we know if two oracles are matching, then both of them are accepted and have the same session key. The session key is distributed uniformly since the exponent a and b are selected uniformly during the scheme execution.

Lemma 2. Under the assumption that the BDH problem is intractable, the advantage of a Type 1 adversary against our scheme is negligible in the random oracle model.

Proof. For a contradiction, assume that the Type 1 adversary $\mathcal{A}1$ has non-negligible advantage ε, and makes at most q_i queries to H_i, where $i = 1, 2$. Let q_s be the total number of the oracles that $\mathcal{A}1$ creates, i.e., for any oracle $\prod_{A,B}^n$, $n \in \{1, \cdots, q_s\}$. We shall slightly abuse the notation $\prod_{A,B}^n$ to refer to the nth one among all the q_s participant instances, instead of the n-th instance of participant A. As n is only used to help identify oracles, this notation change will not affect the soundness of the model.

We show how to construct a simulator S that uses $\mathcal{A}1$ as a sub-routine to solves the BDH problem with non-negligible probability. Given input of the two groups G_1, G_2, the bilinear map e, a generator P of G_1, and a triple of elements $aP, bP, cP \in G_1$ with $a, b, c \in Z_q^*$ where q is the prime order of G_1 and G_2, S's task is to compute and output the value $e(P,P)^{abc} \in G_2$.

The algorithm S selects two random integers I, J from $\{1, \cdots, q_1\}$ and a random integer m from $\{1, \cdots, q_s\}$. S guesses that the m th oracle (i.e. $\prod_{I,J}^m$) will be asked the $Test$ query and works by interacting with $\mathscr{A}$1 as follows:

$Setup$: S treats the unknown value a as the PKG's master key. S starts $\mathscr{A}$1, and answers all $\mathscr{A}$1's queries as follows.

$H_1(ID_i)$: S simulates the random oracle H_1 by keeping a list of tuples (r_i, ID_i, Q_i) which is called the L_{H_1}. When the H_1 oracle is queried with an input ID_i, S responds as follows.

- If ID_i is already on L_{H_1} in the tuple (r_i, ID_i, Q_i), then S outputs Q_i.
- Otherwise, if ID_i is the I-th distinct H_1 query, then the oracle outputs $Q_i = bP$; If ID_i is the J-th distinct H_1 query, then the oracle outputs $Q_i = cP$. S adds the tuple ($\perp, ID_i, Q_i$) to L_{H_1}.
- Otherwise S selects a random $r_i \in Z_q^*$ and outputs $Q_i = r_i P$, and then adds the tuple (r_i, ID_i, Q_i) to L_{H_1}.

$Create(ID_i)$: S maintains an initially empty list L_C consisting of tuples of the form (ID_i, D_i, x_i, P_i). When queried with an input ID_i, S query the random oracle H_1 with ID_i, gets a tuple (r_i, ID_i, Q_i) and responds as follows.

- If ID_i is already on L_C in the tuple (ID_i, D_i, x_i, P_i), then S outputs P_i.
- Otherwise, if $r_i = \perp$, S generates a random number $x_i \in Z_q^*$ as the secret key, computes the public key $P_i = x_i P$, set the partial secret key $D_i \leftarrow \perp$. S adds the tuple ($ID_i, \perp, x_i, P_i$) to L_C and outputs P_i.
- Otherwise S S generates a random number $x_i \in Z_q^*$ as the secret key, computes the public key $P_i = x_i P$, set the partial secret key $D_i \leftarrow r_i P_{pub}$. S adds the tuple (ID_i, D_i, x_i, P_i) to L_C and outputs P_i.

$Public-Key(ID_i)$: On receiving this query, S first searches for a tuple (ID_i, D_i, x_i, P_i) in L_C which is indexed by ID_i, then returns P_i as the answer.

$Partial-Private-Key(ID_i)$: Whenever S receives this query, if $ID_i = I$ or J, S aborts (Event 1); else, S searches for a tuple (ID_i, D_i, x_i, P_i) in L_C which is indexed by ID_i and returns D_i as the answer.

$Corrupt(ID_i)$: Whenever S receives this query, if $ID_i = I$ or J, S aborts (Event 2); else, S searches for a tuple (ID_i, D_i, x_i, P_i) in L_C which is indexed by ID_i and if $x_i = \perp$, S returns null; otherwise, S returns (D_i, x_i) as the answer.

$Public-Key-Replacement(ID_i, P_i')$: On receiving this query, S searches for a tuple (ID_i, D_i, x_i, P_i) in L_C which is indexed by ID_i, then updates P_i to P_i' and sets $x_i = \perp$.

$Send(\prod_{i,j}^n, M)$: S maintains an initially empty list L_S consisting of tuples of the form ($\prod_{i,j}^n, trans_{i,j}^n, r_{i,j}^n$), where $trans_{i,j}^n$ is the transcript of $\prod_{i,j}^n$ so far and $r_{i,j}^n$ will be described later. S chooses at random $r_{i,j}^n \in Z_n^*$ and computes the reply as $r_{i,j}^n P$. Then S updates the tuple indexed by $\prod_{i,j}^n$ in L_S.

$Reveal(\prod_{i,j}^n)$: S maintains a list L_R of the form ($\prod_{i,j}^n, ID_{ini}^n, ID_{resp}^n, T_{ini}^n, T_{resp}^n, SK_{i,j}^n$) where ID_{ini}^n is the identification of the initiator in the session which $\prod_{i,j}^n$ engages in and ID_{resp}^n is the identification of the responder. The description of the other items will be given later. S answers the query as follows:

- If $n = m$, $ID_i = I$ and $ID_j = J$ or $\prod_{i,j}^n$ is the oracle who has a matching conversion with $\prod_{I,J}^m$, S aborts((Event 3)).
- Else if $ID_i \neq I$ and $ID_i \neq J$
  - ✧ S looks up the list L_S and L_C for corresponding tuple ($\prod_{i,j}^n, r_{i,j}^n, T_{i,j}^n, T_{j,i}^n, P_i^n, P_j^n$) and (ID_i, D_i, x_i, P_i) separately. Then S computes $K_{i,j}^1 = x_i \cdot P_{j,i}^n$, $K_{i,j}^2 = e(D_i, Q_j)$, $K_{i,j}^3 = r_{j,i}^n T_{j,i}^n$.
  - ✧ S makes a H_2 query. If $\prod_{i,j}^n$ is the initiator oracle then the query is of the form ($ID_i \| ID_j \| T_i \| T_j \| K_{i,j}^1 \| K_{i,j}^2 \| K_{i,j}^3$) or else of the form ($ID_j \| ID_i \| T_j \| T_i \| K_{i,j}^1 \| K_{i,j}^2 \| K_{i,j}^3$).
- Else if $ID_i = I$ or $ID_i = J$
  - ✧ S looks up the list L_S for corresponding tuple ($\prod_{i,j}^n, r_{i,j}^n, T_{i,j}^n, T_{j,i}^n, R_i^n, R_j^n, P_i^n, P_j^n$).

- S looks up the list L_{H_2} to see if there exists a tuple index by (ID_i, ID_j, T_i, T_j) if $\prod_{i,j}^n$ is an initiator, otherwise index by (ID_j, ID_i, T_j, T_i).
- If there exists such tuple and the corresponding $K_{i,j}^1$, $K_{i,j}^2$ and $K_{i,j}^3$ satisfies the equation $e(K_{i,j}^1, P) = e(P_i^n, P_j^n)$, $e(Q_i, P_j^n) = K_{i,j}^2$ and $e(K_{i,j}^3, P) = e(T_i^n, T_j^n)$, then S obtains the corresponding h_i and sets $SK_{i,j}^n = h_i$. Otherwise S chooses at random $SK_{i,j}^n \in \{0,1\}^k$.

- Else
  - S looks up the list L_S for corresponding tuple $(\prod_{i,j}^n, r_{i,j}^n, T_{i,j}^n, T_{j,i}^n, R_i^n, R_j^n, P_i^n, P_j^n)$.
  - S looks up the list L_{H_2} to see if there exists a tuple index by (ID_i, ID_j, T_i, T_j) if $\prod_{i,j}^n$ is an initiator, otherwise index by (ID_j, ID_i, T_j, T_i).
  - If there exists such tuple and the corresponding $K_{i,j}^1$, $K_{i,j}^2$ and $K_{i,j}^3$ satisfies the equation $e(K_{i,j}^1, P) = e(P_i^n, P_j^n)$, $e(Q_i, P_j^n) = K_{i,j}^2$ and $e(K_{i,j}^3, P) = e(T_i^n, T_j^n)$, then S obtains the corresponding h_i and sets $SK_{i,j}^n = h_i$. Otherwise S chooses at random $SK_{i,j}^n \in \{0,1\}^k$.

H_2 query: S maintains a list L_{H_2} of the form $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3, h_u)$ and responds with H_2 queries $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3)$ as follows:

- If a tuple indexed by $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3)$ is already in L_{H_2}, reply with the corresponding h_u.
- Else, if there is not such a tuple,
  - If there is a tuple indexed by $(ID_u^i, ID_u^j, T_u^i, T_u^j)$ in the list L_R such that the equation $e(K_u^1, P) = e(P_i^n, P_j^n)$, $e(Q_i, P_j^n) = K_{i,j}^2$ and $e(K_u^3, P) = e(T_u^i, T_u^j)$ hold, then S obtain the corresponding $SK_{i,j}^n$ and sets $SK_{i,j}^n = h_u$.Otherwise choose at random $h_u \in \{0,1\}^k$.
  - Else if the equations do not hold for $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3)$, S chooses at random $h_u \in \{0,1\}^k$.
  - S inserts the tuple $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3, h_u)$ into the list L_{H_2}.

$Test(\prod_{I,J}^m)$: At some point, S will ask a *Test* query on some oracle. If S does not choose one of the oracles $\prod_{I,J}^m$ to ask the *Test* query, then S aborts (Event 4). Otherwise, S simply outputs a random value $x \in \{0,1\}^k$.

The probability that S chooses $\prod_{I,J}^{m}$ as the *Test* oracle and that $\frac{1}{q_1^2 q_s}$. In this case, S would not have made $Corrupt(\prod_{I,J}^{m})$ or $Reveal(\prod_{I,J}^{m})$ queries, and so S would not have aborted. If S can win in such a game, then S must have made the corresponding H_2 query of the form ($ID_m^i, ID_m^j, T_m^i, T_m^j, K_m^1, K_m^2, K_m^3$) if $\prod_{I,J}^{m}$ is the initiator oracle or else ($ID_m^j, ID_m^i, T_m^j, T_m^i, K_m^1, K_m^2, K_m^3$) with overwhelming probability because H_2 is a random oracle. Thus S can find the corresponding item in the H_2-list with the probability $\frac{1}{q_2}$ and output K_m^2 as a solution to the BDH problem. So if the adversary computes the correct session key with non-negligible probability ε, then the probability that S solves the BDH problem is $\frac{\varepsilon}{q_1^2 q_2 q_s}$ (which is non-negligible in the security parameter l), contradicting to the hardness of the BDH problem.

Lemma 3. Under the assumption that the CDH problem is intractable, the advantage of a Type 1 adversary against our scheme is negligible in the random oracle model.

Proof. For a contradiction, assume that the Type 2 adversary $\mathscr{A}2$ has non-negligible advantage ε, and makes at most q_i queries to H_i, where $i = 1, 2$. Let q_s be the total number of the oracles that $\mathscr{A}1$ creates, i.e., for any oracle $\prod_{A,B}^{n}$, $n \in \{1, \cdots, q_s\}$.

We show how to construct a simulator S that uses $\mathscr{A}2$ as a sub-routine to solves the CDH problem with non-negligible probability. Given input of the group G_1, a generator P of G_1, and a triple of elements $aP, bP \in G_1$ with $a, b \in Z_q^*$ where q is the prime order of G_1, S's task is to compute and output the value $abP \in G_1$.

The algorithm S selects two random integers I, J from $\{1, \cdots, q_1\}$ and a random integer m from $\{1, \cdots, q_s\}$. S guesses that the mth oracle (i.e. $\prod_{I,J}^{m}$) will be asked the $Test$ query and works by interacting with $\mathscr{A}2$ as follows:

$Setup$: S selects a random number $s \in Z_q^*$ as the PKG's master key and computes $P_{pub} = sP$ as the PKG's public key. S starts $\mathcal{A}$2, gives the master key s to $\mathcal{A}$2, and answers all $\mathcal{A}$2's queries as follows.

$H_1(ID_i)$: S simulates the random oracle H_1 by keeping a list of tuples (r_i, ID_i, Q_i) which is called the L_{H_1}. When the H_1 oracle is queried with an input ID_i, S responds as follows.

- If ID_i is already on L_{H_1} in the tuple (r_i, ID_i, Q_i), then S outputs Q_i.
- Otherwise S selects a random $r_i \in Z_q^*$ and outputs $Q_i = r_i P$, and then adds the tuple (r_i, ID_i, Q_i) to L_{H_1}.

$Create(ID_i)$: $\mathcal{C}$ maintains an initially empty list L_C consisting of tuples of the form (ID_i, D_i, x_i, P_i). When queried with an input ID_i, S query the random oracle H_1 with ID_i, gets a tuple (r_i, ID_i, Q_i) and responds as follows.

- If ID_i is already on L_C in the tuple (ID_i, D_i, x_i, P_i), then S outputs P_i.
- Otherwise, if $ID_i = I$, S computes the partial secret key $D_i = sQ_i$ and sets the secret key $x_i \leftarrow \perp$, the public key $P_i \leftarrow aP$. S adds the tuple (ID_i, D_i, x_i, P_i) to L_C and outputs P_i.
- Otherwise, if $ID_i = J$, S computes the partial secret key $D_i = sQ_i$ and sets the secret key $x_i \leftarrow \perp$, the public key $P_i \leftarrow bP$. S adds the tuple (ID_i, D_i, x_i, P_i) to L_C and outputs P_i.
- Otherwise S generates a random number $x_i \in Z_q^*$ as the secret key, computes the public key $P_i = x_i P$, computes the partial secret key $D_i = sQ_i$. S adds the tuple (ID_i, D_i, x_i, P_i) to L_C and outputs P_i.

$Public-Key(ID_i)$: On receiving this query, S first searches for a tuple (ID_i, D_i, x_i, P_i) in L_C which is indexed by ID_i, then returns P_i as the answer.

$Partial-Private-Key(ID_i)$: Whenever S receives this query, S searches for a tuple (ID_i, D_i, x_i, P_i) in L_C which is indexed by ID_i and returns D_i as the answer.

$Corrupt(ID_i)$: Whenever S receives this query, if $ID_i = I$ or J, S aborts (Event 2); else, S searches for a tuple (ID_i, D_i, x_i, P_i) in L_C which is indexed by ID_i and returns (D_i, x_i) as the answer.

$Send(\prod_{i,j}^{n}, M)$: S maintains an initially empty list L_S consisting of tuples of the form ($\prod_{i,j}^{n}, trans_{i,j}^{n}, r_{i,j}^{n}$), where $trans_{i,j}^{n}$ is the transcript of $\prod_{i,j}^{n}$ so far and $r_{i,j}^{n}$ will be described later. S chooses at random $r_{i,j}^{n} \in Z_n^*$ and computes the reply as $r_{i,j}^{n}P$. Then S updates the tuple indexed by $\prod_{i,j}^{n}$ in L_S.

$Reveal(\prod_{i,j}^{n})$: S maintains a list L_R of the form ($\prod_{i,j}^{n}, ID_{ini}^{n}, ID_{resp}^{n}, T_{ini}^{n}, T_{resp}^{n}, SK_{i,j}^{n}$) where ID_{ini}^{n} is the identification of the initiator in the session which $\prod_{i,j}^{n}$ engages in and ID_{resp}^{n} is the identification of the responder. The description of the other items will be given later. S answers the query as follows:

- If $n = m$, $ID_i = I$ and $ID_j = J$ or $\prod_{i,j}^{n}$ is the oracle who has a matching conversion with $\prod_{I,J}^{m}$, S aborts((Event 3)).
- Else if $ID_i \neq I$ and $ID_i \neq J$
  - S looks up the list L_S and L_C for corresponding tuple ($\prod_{i,j}^{n}, r_{i,j}^{n}, T_{i,j}^{n}, T_{j,i}^{n}, P_i^n, P_j^n$) and ($ID_i, D_i, x_i, P_i$) separately. Then S computes $K_{i,j}^1 = x_i \cdot P_{j,i}^n$, $K_{i,j}^2 = e(D_i, Q_j)$, $K_{i,j}^3 = r_{j,i}^n T_{j,i}^n$.
  - S makes a H_2 query. If $\prod_{i,j}^{n}$ is the initiator oracle then the query is of the form ($ID_i \| ID_j \| T_i \| T_j \| K_{i,j}^1 \| K_{i,j}^2 \| K_{i,j}^3$) or else of the form ($ID_j \| ID_i \| T_j \| T_i \| K_{i,j}^1 \| K_{i,j}^2 \| K_{i,j}^3$).
- Else if $ID_i = I$ or $ID_i = J$
  - S looks up the list L_S for corresponding tuple ($\prod_{i,j}^{n}, r_{i,j}^{n}, T_{i,j}^{n}, T_{j,i}^{n}, R_i^n, R_j^n, P_i^n, P_j^n$).
  - S looks up the list L_{H_2} to see if there exists a tuple index by (ID_i, ID_j, T_i, T_j) if $\prod_{i,j}^{n}$ is an initiator, otherwise index by (ID_j, ID_i, T_j, T_i).
  - If there exists such tuple and the corresponding $K_{i,j}^1$, $K_{i,j}^2$ and $K_{i,j}^3$ satisfies the equation $e(K_{i,j}^1, P) = e(P_i^n, P_j^n)$, $e(Q_i, P_j^n) = K_{i,j}^2$ and $e(K_{i,j}^3, P) = e(T_i^n, T_j^n)$, then S obtains the corresponding h_i and sets $SK_{i,j}^n = h_i$. Otherwise S chooses at random $SK_{i,j}^n \in \{0,1\}^k$.
- Else
  - S looks up the list L_S for corresponding tuple ($\prod_{i,j}^{n}, r_{i,j}^{n}, T_{i,j}^{n}, T_{j,i}^{n}, R_i^n, R_j^n, P_i^n, P_j^n$).
  - S looks up the list L_{H_2} to see if there exists a tuple index by (ID_i, ID_j, T_i, T_j) if $\prod_{i,j}^{n}$ is an initiator, otherwise index by (ID_j, ID_i, T_j, T_i).

- If there exists such tuple and the corresponding $K_{i,j}^1$, $K_{i,j}^2$ and $K_{i,j}^3$ satisfies the equation $e(K_{i,j}^1, P) = e(P_i^n, P_j^n)$, $e(Q_i, P_j^n) = K_{i,j}^2$ and $e(K_{i,j}^3, P) = e(T_i^n, T_j^n)$, then S obtains the corresponding h_i and sets $SK_{i,j}^n = h_i$. Otherwise S chooses at random $SK_{i,j}^n \in \{0,1\}^k$.

H_2 query: S maintains a list L_{H_2} of the form

$(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3, h_u)$ and responds with H_2 queries

$(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3)$ as follows:

- If a tuple indexed by $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3)$ is already in L_{H_2}, reply with the corresponding h_u.
- Else, if there is not such a tuple,
  - If there is a tuple indexed by $(ID_u^i, ID_u^j, T_u^i, T_u^j)$ in the list L_R such that the equation $e(K_u^1, P) = e(P_i^n, P_j^n)$, $e(Q_i, P_j^n) = K_{i,j}^2$ and $e(K_u^3, P) = e(T_u^i, T_u^j)$ hold, then S obtain the corresponding $SK_{i,j}^n$ and sets $SK_{i,j}^n = h_u$ .Otherwise choose at random $h_u \in \{0,1\}^k$.
  - Else if the equations do not hold for $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3), S$ chooses at random $h_u \in \{0,1\}^k$.
  - S inserts the tuple $(ID_u^i, ID_u^j, T_u^i, T_u^j, K_u^1, K_u^2, K_u^3, h_u)$ into the list L_{H_2}.

$Test(\prod_{I,J}^m)$: At some point, S will ask a *Test* query on some oracle. If S does not choose one of the oracles $\prod_{I,J}^m$ to ask the *Test* query, then S aborts (Event 4). Otherwise, S simply outputs a random value $x \in \{0,1\}^k$.

The probability that S chooses $\prod_{I,J}^m$ as the *Test* oracle and that $\frac{1}{q_1^2 q_s}$. In this case, S would not have made $Corrupt(\prod_{I,J}^m)$ or $Reveal(\prod_{I,J}^m)$ queries, and so S would not have aborted. If S can win in such a game, then S must have made the corresponding H_2 query of the form $(ID_m^i, ID_m^j, T_m^i, T_m^j, K_m^1, K_m^2, K_m^3)$ if $\prod_{I,J}^m$ is the initiator oracle or else $(ID_m^j, ID_m^i, T_m^j, T_m^i, K_m^1, K_m^2, K_m^3)$ with overwhelming probability because H_2 is a random oracle. Thus S can find the corresponding item in the H_2-list with the probability $\frac{1}{q_2}$ and output K_m^1 as a solution to the CDH problem. So if the adversary computes the correct session key with non-negligible probability ε, then the probability that S solves the

CDH problem is $\frac{\varepsilon}{q_1^2 q_2 q_s}$ (which is non-negligible in the security parameter l), contradicting to the hardness of the CDH problem.

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

Theorem 1. Our scheme is a secure CTAKA scheme.

Through the similar method, we can prove our scheme could provide forward secrecy property. We describe it as the following theorem.

4. Comparison with previous scheme

We summaries the security properties and performances of the proposed scheme and several related schemes from pairings. Table 1 compares the total time complexity of those schemes while Table 2 compares the reduced time complexity with pre-computation.

For simplicity, we only consider the following computationally expensive operations.

- P : pairing.
- M : scalar point multiplication in G_1.
- E : exponentiation in G_2.
- A : point addition in G_1.

Table 1. Comparisons of other CTAKA schemes from pairings(without pre-computation).

Schemes	Items					
	P	M	E	A	Bandwidth	Formal proof
Wang et al.[3]	2	3	1	0	1 point	No
Shi et al.[4]	1	2	1	0	1 point	No
Luo et al.[5]	2	4	0	0	1 point	No
Mandt et al.[6]	2	3	1	2	1 point	No
Wang et al.[7]	2	2	1	0	1 point	No
Zhang et al.[8]	1	5	0	2	1 point	Yes
This paper	1	3	0	0	1 point	Yes

Table 2. Comparisons of other CTAKA schemes from pairings(with pre-computation).

Schemes	Items					
	P	*M*	*E*	*A*	Bandwidth	Formal proof
Wang et al.[3]	2	3	1	0	1 point	No
Shi et al.[4]	1	2	1	0	1 point	No
Luo et al.[5]	2	3	0	0	1 point	No
Mandt et al.[6]	2	2	0	2	1 point	No
Wang et al.[7]	2	2	1	0	1 point	No
Zhang et al.[8]	1	4	0	2	1 point	Yes
This paper	0	2	0	0	1 point	Yes

From Table 1, we know each party in our scheme just needs one pairing operation and three scale multiplications. Considering pairing evaluation is far more computationally expensive that other operations, our scheme has the better performance than other schemes[3-8]. Moreover, the party A in our scheme can pre-compute K_{AB}^1, K_{AB}^2 since x_A, P_B, D_A and Q_B are constant. At the same time, the party B can pre-compute K_{BA}^1, K_{BA}^2. Then each party just needs to compute two scale multiplications on-line in order to finish the key agreement. From Table 2, we observe that only our scheme eliminates on-line pairing evaluation. Then our scheme appears to be the most efficient scheme, especially when we consider that certain computations can be performed off-line.

5. Conclusion

In this paper, we have proposed an efficient CTAKA scheme from pairings. We also prove the security of the scheme under random oracle. Compared with previous scheme, the new scheme reduces the running time. Therefore, our scheme is more practical than the previous related schemes for practical application.

6. References

[1]. Shamir, Identity-based cryptosystems and signature schemes, Proceedings of CRYPTO 84 on Advances in cryptology, Springer-Verlag, New York, USA, 1984, pp.47–53.

[2]. S. Al-Riyami, K.G. Paterson, Certificateless public key cryptography, Proceedings of ASIACRYPT 2003, Springer-Verlag, Taiwan, China, 2003, pp. 452－473.

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

[4]. 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.

[5]. M. Luo, Y. Wen, H. Zhao, An enhanced authentication and key agreement mechanism for SIP using certificateless public-key cryptography, Proceedings of the IEEE ICYCS 2008, IEEE, Hunan, China, 2008, pp. 1577 – 1582.

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

[7]. F. Wang, Y. Zhang, A new provably secure authentication and key agreement mechanism for SIP using certificateless public-key cryptography, Computer Communications 31 (10) (2008) 2142 – 2149.

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

[9]. D. Boneh, M. Franklin, Identity-based encryption from the Weil pairing, Crypto'01, Springer-Verlag, Santa Barbara, California, USA, 2001, pp. 213 – 229.

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