An authenticated key agreement (AKA) protocol is extremely essential to secure communications over insecure public networks. It enables the communication parties to securely set up a shared session key in present of the malicious attackers. Certificate-based cryptography (CBC) is a novel public-key cryptographic primitive that has many attractive merits. It solves the certificate revocation problem in conventional public-key cryptography and the key-escrow problem in identity-based cryptography. Until now, four AKA protocols have been proposed in the setting of CBC. However, all of them adopt the costly bilinear pairings and are not suitable for the devices which have limited computing resources and battery power. Therefore, it is interesting and worthwhile to design a certificate-based AKA protocol without using the bilinear pairings. In this paper, we develop a pairing-free certificate-based AKA protocol. The proposed protocol is proven secure under the classic computational Diffie-Hellman assumption in the random oracle model. Compared with the previous pairing-based certificate-based AKA protocols, the proposed protocol enjoys obvious advantage in the computation efficiency.

**KEYWORDS:** wireless sensor network, authenticated key agreement protocol, certificate-based cryptography, random oracle model, bilinear pairing.
1. Introduction

To provide secure communications over insecure public networks, privacy and confidentiality should be guaranteed. An authenticated key agreement (AKA) protocol enables the communication parties to authenticate each other and securely set up a shared session key for their communications over an unreliable communication channel. Therefore, it can assure the privacy and data confidentiality of the later communications.

The famous Diffie-Hellman key-exchange protocol [8] is the first practical key agreement protocol. However, Diffie-Hellman protocol suffers from the man-in-the-middle (MITM) attack due to the reason that it does not provide authentication to the protocol participants. The protocols that can provide the authentication mechanism have attracted great attention from the research community (e.g. [14–17]).

Over the years, many AKA protocols have been proposed. However, most of the previous AKA protocols were constructed over either conventional public-key cryptography (PKC) [4, 11, 13, 20, 25] or identity-based cryptography (IBC) [5–7, 36, 41, 44]. It is well-known that conventional PKC suffers from the heavy certificate management problem while IBC has the key escrow and key distribution problems.

To solve the key escrow problem, Al-Riyami and Paterson [1] presented the concept of certificateless AKA (CL-AKA) protocol by extending AKA protocol into certificateless PKC. In a CL-AKA protocol, a partially trusted key generation center (KGC) is employed to assist every user to produce a private key from a secret value selected by the user. In this way, KGC does not know any user’s private key. As a result, CL-AKA protocols avoid the key escrow problem while reserving the certificateless property. Since the introduction of CL-AKA protocol, many CL-AKA protocols have been proposed [12, 19, 26, 40, 42, 47, 48]. However, KGC has to distribute a partial private key to every user over secure channels. This feature limits the application of CL-AKA protocols in the untrusted network environments, because setting up a secure channel in those environments is usually expensive.

In Eurocrypt 2003, Gentry [10] put forward a new public-key cryptographic paradigm called certificate-based cryptography (CBC). This new cryptography lies between conventional PKC and IBC. When using CBC, every user should produce a pair of private key and public key independently and then apply for a certificate from a trusted certification authority (CA). The certificate is sent to its owner and serves as a partial decryption key or a partial signing key. This functionality of the certificate supplies an implicit certificate property so that a user can execute some cryptographic operations (e.g., decryption and signing) correctly only when both his certificate and private key are known, while this user’s communication parties need not obtain the current status of his certificate. Therefore, CBC solves the certificate revocation problem in conventional PKC. Furthermore, CBC eliminates both the key escrow problem (as CA has no knowledge of users’ private keys) and the key distribution problem (as CA can send the certificates to their owners publicly). Since its invention, CBC has absorbed great interest from the cryptography community and numerous CBC schemes have been published, including certificate-based encryption (e.g. [9, 28, 31, 32, 39, 46]), certificate-based signature (e.g. [2, 18, 21, 22, 27, 30]) and certificate-based signature (e.g. [23, 29, 34]).

To overcome the problems imposed on the previous AKA protocols, Wang and Cao [45] proposed the notion of certificate-based AKA (CB-AKA) protocol following the idea of CBC. Compared with previous types of AKA protocol, CB-AKA protocol enjoys many attractive merits. Table 1 summarizes the properties of the different types of AKA protocol, including conventional AKA protocol over conventional PKC,

<table>
<thead>
<tr>
<th>Types of AKA Protocol</th>
<th>Implicit Certificate</th>
<th>Key-Eescrow Free</th>
<th>Secure-Channel Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional AKA protocol</td>
<td>×</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>IB-AKA protocol</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>CL-AKA protocol</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>CB-AKA protocol</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>
identity-based AKA (IB-AKA) protocol over IBC, CI-AKA protocol and CB-AKA protocol.

To our knowledge, there exist four CB-AKA protocols [24, 33, 35, 45] in the literature so far. In [45], Wang and Cao proposed the first CB-AKA protocol that is based on Gentry’s certificate-based encryption scheme [10] and Smart’s AKA protocol [41]. Unfortunately, Lim et al. [24] pointed out that Wang-Cao’s CB-AKA protocol is insecure against ephemeral secret leakage. To fix the weakness in Wang-Cao’s protocol, Lim et al. [24] proposed an improved CB-AKA protocol. They claim that the improved protocol is secure against all non-trivial secret leakages. However, no formal security proof is given in [24]. In [35], Luo et al. provide a security model for constructing provably secure CB-AKA protocols. They also present a CB-AKA protocol that can be proven secure in the random oracle model [3]. Recently, Lu et al. [33] pointed out that the CB-AKA protocols in [24, 35, 45] can not resist the public key replacement (PKR) attack. To fight against PKR attack, Lu et al. [33] designed a new CB-AKA protocol with provable security in the random oracle model.

The motivation of this paper is to design a CB-AKA protocol without costly bilinear pairing. In practice, the cryptographic operations are often performed on some devices which have very constrained resources, such as smart phone or PDA. Due to the limited computation or the constrained battery power, only the lightweight or power-saving cryptographic schemes can be employed on these devices. All the previous CB-AKA protocols [24, 33, 35, 45] are constructed with bilinear pairings. Compared with other common cryptographic operations such as scalar multiplications in the elliptic curve group, the bilinear pairing may be the most expensive one. Our experiment results show that the average computation cost of a bilinear pairing is about nine times as much as that of a scalar multiplication in elliptic curve group under the 1024-bit RSA security level. Since the computation-ally-heavy pairing operations will greatly aggravate the computation load of a device, they are extremely disliked by the computation-limited or power-constrained devices. Therefore, as far as the efficiency, the cryptographic schemes without bilinear pairing would be more attractive.

Inspired by Schnorr’s signature scheme [37], we develop a practical CB-AKA protocol without bilinear pairing. Under the classic complexity assumption computational Diffie-Hellman assumption, the proposed protocol is proven secure in the random oracle model. Without costly bilinear pairing operations, the proposed CB-AKA protocol significantly decreases the computation cost. Compared with the previous pairing-based CB-AKA protocols [24, 33, 35, 45], it enjoys obvious advantage in the computational efficiency and is more suitable for the power-constrained and computation-limited devices.

2. Preliminaries

2.1 Computational assumption

The security of our CB-AKA protocol is based on the computational Diffie-Hellman (CDH) assumption.

Definition 1. Let \( G \) be a cyclic group of prime order \( q \). The CDH problem over \( G \) is, given a tuple \((P, xP, yP) \in G^3\) for unknown values \( x, y \in \mathbb{Z}_q^* \), to compute \( xyP \). The CDH assumption holds if for any polynomial-time algorithm \( A \), the advantage \( Adv(A) = \Pr\{A(G, q, P, xP, yP) = xyP\} \) is negligible.

2.2 Bilinear pairing

Assume that \( G \) and \( G_T \) are two cyclic groups of prime order \( q \). A bilinear pairing \( e: G \times G \rightarrow G_T \) is a map that satisfies the following properties:

1. Bilinearity: \( e(xU, yV) = e(U, V)^{xy} \) for all \( U, V \in G \) and \( x, y \in \mathbb{Z}_q^* \).
2. Non-degeneracy: There exists \( U, V \in G \) such that \( e(U, V) \neq 1 \).
3. Computability: There exists an efficient algorithm to compute \( e(U, V) \) for all \( U, V \in G \).

Note that the construction of our CB-AKA protocol does not depend on the bilinear pairing. We only use this notion in the security proofs.

3. Formal model of CB-AKA protocol

3.1 Definition of CB-AKA protocol

Usually, a CB-AKA protocol consists of four algorithms: (1) System setup algorithm \( \text{Setup} \), which is
performed by a CA to produce a master secret key and a set of system public parameters; (2) User key generation algorithm UserKeyGen, which is performed by each user to produce a private key and a partial public key; (3) Certificate generation algorithm CertGen, which is performed by a CA to produce a full public key and a certificate for each user; (4) Key agreement algorithm KeyAgreement, which is performed by two communication users (an initiator and a responder) to generate a session key.

Figure 1 gives a more concrete functional description of a CB-AKA protocol.

3.2 Security model of CB-AKA protocol

As introduced in [33, 35], a CB-AKA protocol should satisfy some commonly desired security properties, including known-key security, unknown key-share resilience, basic impersonation attacks resilience, forward secrecy, key compromise impersonation resilience and key control security.

To capture these security properties, Luo et al. presented a formal security model for CB-AKA protocols in [35]. However, Lu et al. [33] show that Luo et al.’s CB-AKA protocol is insecure against the PKR attacks, although it was proven secure in their proposed security model. To fix this problem, Lu et al. [33] improved Luo et al.’s security model to capture the adversaries’ PKR actions.

There are two types of adversaries against the security of CB-AKA protocols. They are the Type 1 adversary and the Type 2 adversary. The Type 1 adversary who acts as a malicious uncertified user is able to replace any user’s public key, but does not know the target user’s certificate. The Type 2 adversary who acts as an honest-but-curious CA possesses the CA’s master secret key, but does not know the target user’s private key and is disallowed to replace public keys.

The security model of a CB-AKA protocol can be defined via an adversarial game which is played between a challenger with a Type 1 adversary or a Type 2 adversary (denoted by \( \mathcal{A} \)). In the description of the adversarial game, the symbol \( \Pi_{A,B}^n \) denotes an oracle that represents the \( n \)-th session between two participants \( A \) and \( B \), where \( A \) is the initiator and \( B \) is the responder. Let \( S_{U} = (ID_A, ID_B, M_{A,B}^n, M_{B,A}^n) \) be the session identity of the session \( \Pi_{A,B}^n \), where \( M_{A,B}^n \) and \( M_{B,A}^n \) are the incoming protocol message and the outgoing protocol message, respectively, in the \( n \)-th session of the protocol. Two sessions are said to have matching conversation with each other if they have the same session identity.

The security model is formally described as follows:

**Setup.** The challenger simulates the algorithm Setup to produce \( msk \) and \( params \). Then the adversary \( \mathcal{A} \) is given \( params \) if it is a Type 1 adversary or both \( msk \) and \( params \) if it is a Type 2 adversary.

**Phase 1.** The challenger answers the adversary \( \mathcal{A} \)’s various oracle queries as follows:

- **(1) Setup** \((k) \rightarrow (msk, params)\)
  - Input: a security parameter \( k \)
  - Output: a master secret key \( msk \) and a set of public parameters \( params \)

- **(2) UserKeyGen** \((params) \rightarrow (SK_U, PPK_U)\)
  - Input: \( params \)
  - Output: a private key and partial public key pair \((SK_U, PPK_U)\) for a user \( U \)

- **(3) CertGen** \((params, msk, ID_U, PPK_U) \rightarrow (PK_U, Cert_U)\)
  - Input: \( params, msk, \) an identity \( ID_U \) and a partial public key \( PPK_U \)
  - Output: a full public key \( PK_U \) and a certificate \( Cert_U \) for the user \( U \) with identity \( ID_U \)

- **(4) KeyAgreement** \((params, ID_A, PK_A, SK_A, Cert_A, ID_B, PK_B, SK_B, Cert_B) \rightarrow K\)
  - Input: \( params, msk, \) an initiator \( A \)’s identity \( ID_A \), public key \( PK_A \), private key \( SK_A \) and certificate \( Cert_A \), a responder \( B \)’s identity \( ID_B \), public key \( PK_B \), private key \( SK_B \) and certificate \( Cert_B \)
  - Output: a shared secret session key \( K = K_{AB} = K_{BA} \)
1. **CreateUser**(ID): On receiving an identity ID, the challenger outputs a public key PK. If there is no public key associated with ID, the challenger produces a key pair (SK, PK) and a certificate Cert, for ID, and then outputs PK. In this case, the identity ID is said to be created. For simplicity, it is assumed that an identity can be responded by other oracles only when it has been created.

2. **ReplacePublicKey**(ID, PK’): On receiving an identity ID, a false public key PK’, the challenger sets PK’ as the user i’s current public key. Such an oracle is merely queried by the Type 1 adversary. It models the Type 1 adversary’s ability to replace public keys and thus captures the PKR attacks.

3. **Corrupt**(ID): On receiving an identity ID, the challenger outputs a private key SK. For the Type 1 adversary, if it has made an oracle query **ReplacePublicKey**(ID, PK’), then it is disallowed to request the user i’s private key.

4. **Certificate**(ID): On receiving an identity ID, the challenger outputs a certificate Cert. Such an oracle is merely queried by the Type 1 adversary. If it has made an oracle query **ReplacePublicKey**(ID, PK’), then it is disallowed to request the user i’s certificate.

5. **Send**(PID, M): On receiving an oracle PID, and a message M, the challenger initiates a protocol session between the users i and j if M = λ or responds with an outgoing message according to the specification of the protocol otherwise. If the first message received by an oracle is λ, then PID is called an initiator; otherwise it is a responder oracle.

6. **Reveal**(PID): On receiving an oracle PID, the challenger responds with the shared session key associated with PID.

**Test.** Once Phase 1 is over, the adversary A makes one Test query on an oracle PID which is fresh (see the following Definition 2). To respond, the challenger picks a random bit b ∈ {0, 1}. It outputs the shared session key SKPID associated with PID if b = 0 or a random key chosen from the session key space otherwise.

**Phase 2.** A continues to make a sequence of adaptive queries.

**Guess.** A outputs its guess b’ ∈ {0, 1}. It wins the game if and only if b = b’ and the following constraints are satisfied: (1) A is unable to make an oracle query **Certificate**(ID) if it is a Type 1 adversary or an oracle query **Corrupt**(ID) if it is a Type 2 adversary.

The advantage of the adversary A in winning the game is defined to be AdvA = Pr[b = b’] − 1/2.

**Definition 2.** An oracle PID is fresh if (1) It has established a shared session key; (2) The adversary A has not queried the oracle **Reveal** on it and its matching conversation; (3) The adversary A has not queried the oracle **Certificate** on the identity ID if it is a Type 1 adversary or the oracle **Corrupt** on the identity ID if it is a Type 2 adversary.

**Definition 3.** We say that a CB-AKA protocol is secure if the following two conditions are both satisfied: (1) Two oracles PID and PID, always establish the same session key in the presence of a benign adversary, and the session key is distributed uniformly at random in the session key space; (2) Adv(A) is negligible for any adversary A.

### 4. The proposed CB-AKA protocol

The proposed CB-AKA protocol is described as follows:

**Setup**(k): On input a security parameter k, this algorithm generates a cyclic group G of prime order q with generator P. It then chooses a random integer s ∈ Zq as the CA’s master secret key msk and calculates the master public key Ppub = sP. Furthermore, it chooses two cryptographic hash functions H1: {0, 1}×G → Zq and H2: {0, 1}×G → G → Zq and H2: {0, 1}×G → G → G. Finally, it outputs msk = s and params = {b, q, G, P, Ppub, H1, H2}.

**UserKeyGen**(params): On input params, this algorithm chooses a random integer xU ∈ Zq as a private key SKU for a user U with identity ID and then computes a partial public key PPUU = xuP.

**CertGen**(params, msk, IDU, PPK): On input params, msk = s and a user U’s identity ID, and partial public key PPUU, this algorithm sets PPKU = PPUU. Then it chooses a random integer yU ∈ Zq and computes PPUU = yU. Finally, it sets the user U’s public key PKU = (PPUU, PPKU) and certificate CertU = yU + sH1(ID, PPKU).

**KeyAgreement**(params, IDA, PKA, SKA, CertA, IDB, PKB, SKB, CertB): Assume that two participants A and B want to establish a shared session key, where the
participant A is the initiator with identity ID_A, public key PK_A, private key SK_A and certificate Cert_A, while the participant B is the responder with identity ID_B, public key PK_B, private key SK_B and certificate Cert_B. As described in Figure 2, they can do as follows:

1. The participant A chooses a random integer \( t_A \in Z_q \), computes \( T_A = t_A P \) and then sends a message \( M_A = (ID_A, T_A) \) to B.

2. After receiving \( M_A \), the participant B chooses a random integer \( t_B \in Z_q \), computes \( T_B = t_B P \) and sends a message \( M_B = (ID_B, T_B) \) to A.

3. A and B, respectively, calculate four shared secrets as follows:

   A calculates
   
   \[
   \begin{align*}
   K_{AB}^{(1)} &= (SK_A + Cert_A + t_A)(PK_B^{(1)} + W_B), \\
   K_{AB}^{(2)} &= (SK_A + Cert_A + t_A)(T_B + W_B), \\
   K_{AB}^{(3)} &= t_A PK_B^{(1)} + SK_A T_B, \\
   K_{AB}^{(4)} &= t_A T_B,
   \end{align*}
   \]

   where \( W_B = PK_B^{(2)} + H(ID_B, PK_B)P_{pub} \).

   B calculates
   
   \[
   \begin{align*}
   K_{BA}^{(1)} &= (SK_B + Cert_B)W_A, \\
   K_{BA}^{(2)} &= (t_B + Cert_B)W_A,
   \end{align*}
   \]

Figure 2

Key agreement phase of the proposed CB-AKA protocol

\[
K_{AB}^{(3)} = t_A PK_A^{(1)} + SK_B T_A, \\
K_{BA}^{(4)} = t_B T_A,
\]

where \( W_A = PK_A^{(2)} + H(ID_A, PK_A)P_{pub} + T_A \).

4. A and B, respectively, compute the shared session key as follows:

\[
\begin{align*}
K_{AB}^{(3)} &= H(ID_A, ID_B, PK_A, PK_B, T_A, T_B, K_{AB}^{(1)}, K_{AB}^{(2)}, K_{AB}^{(3)}, K_{AB}^{(4)}), \\
K_{BA}^{(4)} &= H(ID_A, ID_B, PK_A, PK_B, T_A, T_B, K_{BA}^{(1)}, K_{BA}^{(2)}, K_{BA}^{(3)}, K_{BA}^{(4)}).
\end{align*}
\]

Thus, we have \( K_{AB} = K_{BA} \).

For the correctness of the proposed protocol, we can deduce that

\[
\begin{align*}
K_{AB}^{(1)} &= (SK_A + Cert_A + t_A)(PK_A^{(1)} + W_A), \\
K_{AB}^{(2)} &= (SK_A + Cert_A + t_A)(T_B + W_A), \\
K_{AB}^{(3)} &= t_A PK_A^{(1)} + SK_A T_A,
\end{align*}
\]

Thus, we have \( K_{AB} = K_{BA} \).
5. Security analysis

The security of the proposed CB-AKA protocol can be formally proved by combining the following three theorems.

Theorem 1. In the presence of a benign adversary, any two oracles \( \prod_{ij}^1 \) and \( \prod_{ij}^2 \), always establish the same shared session key that is distributed uniformly at random.

Proof. According to the correctness verification of the proposed protocol in Section 4, we can see that if two oracles \( \prod_{ij}^1 \) and \( \prod_{ij}^2 \) are matching, then they must establish the same shared session key. In addition, since the values \( t_a, t_b \) are randomly picked and \( T_a, T_b \) can be viewed as the random input of the hash function \( H_a \), the shared session key can be viewed as the output of \( H_a \). Thus, the session key is uniformly distributed due to the properties of hash functions.

Theorem 2. Suppose that \( H_1 \) and \( H_2 \) are two random oracles. If there is a Type 1 adversary \( A \), against the security of our CB-AKA protocol with advantage \( \varepsilon \) when running in time \( \tau \), making at most \( q_a \) queries to the oracles, \( q_o \) queries to the oracle CreateUser, \( q_{co} \) queries to the oracle ReplacePublicKey, \( q_o \) queries to the oracle Corrupt, \( q_{co} \) queries to the oracle Certificate, \( q_{co} \) queries to the oracle Send, \( q_{co} \) queries to the oracle Reveal and \( q_i \) queries to the random oracles \( H_i (1 \leq i \leq 2) \), then there exists an algorithm \( C \) to solve the CDH problem in \( G \) with advantage \( \varepsilon' \geq \frac{\varepsilon}{q_i q_{co} + q_{co} O(1) + q_{co} (4 \tau_m + 9 \tau_r + O(1)) + q_{co} (3 \tau_r + O(1)) + q_o (10 \tau_m + 2 \tau_r + O(1))}, n \) denotes the maximal number of the protocol sessions that each participant participates in, \( \tau \) and \( \tau_m \) respectively, denote the time for computing a pairing and a scalar multiplication in \( G \).

Proof. Suppose that the algorithm \( C \) takes as input a random CDH instance \( \langle g, G, P, uP, vP \rangle \), where \( u, v \in Z_p^* \) are unknown to the algorithm \( C \). To compute \( wP \), \( C \) interacts with the adversary \( A \), as below:

At the beginning of the game, the algorithm \( C \) chooses at random \( P_{pub} \in G \) as the system master public key and sends the params = \{k, q, G, P, P_{pub}, H_1, H_2\} to \( A \). Furthermore, it picks three different indices \( I, J \in \{1, 2, \ldots, q_i\} \) and \( T \in \{1, 2, \ldots, n\} \) at random.

In Phase 1 and Phase 2, the algorithm \( C \) answers the adversary \( A \)'s queries as below:

\[ H_i(ID, PK) \]: The algorithm \( C \) keeps a list \( L_i \) of tuples \( \langle ID, PK, h_i \rangle \). Upon receiving a \( H_i \) query on \( (ID, PK) \), it returns \( h_i \) if a tuple \( \langle ID, PK, h_i \rangle \) is already in the list \( L_i \). Otherwise, it randomly chooses \( h_i \in Z_p^* \) and adds a new tuple \( \langle ID, PK, h_i \rangle \) to the list \( L_i \) and returns \( h_i \).

\[ H_i(ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4)) \]: The algorithm \( C \) keeps a list \( L_i \) of tuples \( \langle ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4) \rangle \). Upon receiving a \( H_i \) query on \( (ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4)) \), it returns \( h_i \) if a tuple \( \langle ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4) \rangle \) \( h_i \) is already in the list \( L_i \). Otherwise, \( C \) does as below:

1. If there exists a tuple \( \langle ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4) \rangle \) \( h_i \) in the list \( L_i \) (maintained by the oracle \( Send \) ) such that \( K_i(2) = \bot, ID^A = ID^B \), and
2. Case 1: \( \prod_{ij}^1 \) is an initiator, \( C \) searches for the list \( L_i \) to see if there exists a tuple \( \langle ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4) \rangle \) \( h_i \) such that \( ID^A = ID^B \), \( T^A = T^B \), \( PK^A = PK^B \), \( PK_i(2) = PK_i(3) \), \( PK_i(4) = PK_i(2) \) and \( T^A = T^B \). If it does, \( C \) computes \( K_i(2) = (SK_i + Cert_i)(PK_i^2 + PK_i^1 + P_{pub}) - t_iT^A \) and checks whether the following equations hold given a proper bilinear map \( e \) for the group \( G \):

\[ e(PK_i^2 + h_iP_{pub}, T^A) = e(K_i^2, P) \],
\[ K_i(1) = (SK_i + Cert_i + t_iPK_i^1 + PK_i^2 + H_i(ID, PK)P_{pub}) \],
\[ K_i(3) = t_iPK_i^1 + SK_iT^A \],
\[ K_i(4) = t_iP_{pub} - T^A \].

If \( K_i(1), K_i(2), K_i(3) \) and \( K_i(4) \) pass the above validations, then \( C \) sets \( h_i = K_i^2 \) and adds a new tuple \( \langle ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4) \rangle \) \( h_i \) to the list \( L_i \) and returns \( h_i \).

Case 2: \( \prod_{ij}^2 \) is a responder, \( C \) searches for the list \( L_i \) to see if there exists a tuple \( \langle ID^A, ID^B, PK^A, PK^B, T^A, T^B, K_i(2), K_i(3), K_i(4) \rangle \) \( h_i \) such that \( ID^A = ID^B \), \( ID^B = ID^A \), \( PK^A = PK^B \), \( PK_i(2) = PK_i(3) \), \( PK_i(4) = PK_i(2) \) and \( T^A = T^B \). If it does, \( C \) computes \( K_i(2) = (SK_i + Cert_i + t_iPK_i^1 + PK_i^2 + h_iP_{pub} + T^A) - SK_iT^A - t_iT^A \) and checks whether the following equations hold given a proper bilinear map \( e \) for the group \( G \):

\[ e(PK_i^2 + h_iP_{pub}, T^A) = e(K_i^2, P) \],
\[ K_i(1) = (SK_i + Cert_i + t_iPK_i^1 + PK_i^2 + H_i(ID, PK)P_{pub}) \],
\[ K_i(3) = t_iPK_i^1 + SK_iT^A \],
\[ K_i(4) = t_iP_{pub} - T^A \].
\[ K_i^{(3)} = SK_i T_i^n + t_{i,j} PK_i^{(3)}, \]
\[ K_i^{(4)} = t_{i,j} T_j^n. \]

If \( K_i^{(3)}, K_i^{(2)}, K_i^{(3)} \) and \( K_i^{(4)} \) pass the above validations, then \( \mathcal{C} \) sets \( h_i = K_i^{(4)} \), puts a new tuple \( < ID_i^4, ID_j^8, PK_i^4, PK_j^8, T_i^4, T_j^8, K_i^{(1)}, K_i^{(2)}, K_i^{(3)}, K_i^{(4)}, h_i > \) in the list \( L_2 \) and returns \( h_i \).

2 Else, if there exists a tuple \( \langle \Pi_{i,j}^n, ID_i, ID_j, PK_i, PK_j, t_i^n, T_i^n, T_j^n, K_{i,j} > \rangle \) on \( L_2 \) such that \( K_{i,j}^{(3)} \neq \perp, ID_i \neq ID_j, \) and the following equations hold given a proper bilinear map \( e \) for the group \( G: \)

\[
e(K_i^{(2)}, P) = e(PK_i^{(2)} + H_i(ID_i, PK_i)P_{pub} + T_i^n, PK_j^{(2)} + H_j(ID_j, PK_j)P_{pub}),
\]
\[
e(K_i^{(3)}, P) = e(T_i^n, PK_i^{(3)})e(PK_j^{(4)}, T_j^n),
\]
\[
e(K_i^{(4)}, P) = e(T_i^n, T_j^n),
\]

then \( \mathcal{C} \) sets \( h_i = K_i^{(4)} \), puts a new tuple \( < ID_i^4, ID_j^8, PK_i^4, PK_j^8, T_i^4, T_j^8, K_i^{(1)}, K_i^{(2)}, K_i^{(3)}, K_i^{(4)}, h_i > \) in the list \( L_2 \) and returns \( h_i \).

3 Otherwise, the algorithm \( \mathcal{C} \) picks a random value \( h_i \in [0, 1]^k \), puts a new tuple \( < ID_i^4, ID_j^8, PK_i^4, PK_j^8, T_i^4, T_j^8, K_i^{(1)}, K_i^{(2)}, K_i^{(3)}, K_i^{(4)}, h_i > \) in the list \( L_2 \) and returns \( h_i \).

CreateUser(ID): The algorithm \( \mathcal{C} \) keeps a list \( L_c \) of tuples \( < ID_i, SK_i, PK_i, Cert, > \). Upon receiving a CreateUser query on \( (ID_i) \), it returns \( PK_i \) if a tuple \( < ID_i, SK_i, PK_i, Cert, > \) is already in the list \( L_c \). Otherwise, it performs as below:

1 If \( ID_i \) is the \( j \)-th identity submitted to this oracle (i.e., \( ID_j = ID_j \)), it randomly chooses \( i_x \in Z_q^\ast \), computes \( PK_i = (x, P, aP - h_i P_{pub}) \) and sets \( Cert_i = \perp \) and \( SK_i = x \) respectively. It then adds new tuples \( < ID_i, x, PK_i, \perp > \) and \( < ID_i, PK_i, h_i > \) to the lists \( L_c \) and \( L_i \), respectively, and returns \( PK_i \).

2 Otherwise, it randomly chooses \( x, h_i \in Z_q^\ast \), computes \( PK_i = (x, P, aP - h_i P_{pub}) \) and sets \( Cert_i = \perp \) and \( SK_i = x \) respectively. It then adds new tuples \( < ID_i, x, PK_i, s > \) and \( < ID_i, PK_i, h_i > \) to the lists \( L_c \) and \( L_i \), respectively, and returns \( PK_i \).

Certificate(ID): Upon receiving a Certificate query on \( (ID_i) \), the algorithm \( \mathcal{C} \) aborts the game if \( ID_i = ID_j \). Otherwise, it searches for the list \( L_c \) to get a tuple \( < ID_i, SK_i, PK_i, Cert, > \) and returns \( Cert_i \).

Corrupt(ID): Upon receiving a Corrupt query on \( (ID_i) \), the algorithm \( \mathcal{C} \) searches for the list \( L_c \) to get a tuple \( < ID_i, SK_i, PK_i, Cert, > \) and returns \( SK_i \).

ReplacePublicKey(ID, PK'): Upon receiving a query on \( (ID_i, PK') \), the algorithm \( \mathcal{C} \) searches for the list \( L_c \) to get a tuple \( < ID_i, SK_i, PK_i, Cert, > \) and replaces it with \( < ID_i, \perp, PK', \perp > \).

Send(\( \Pi_{i,j}^n, M \)): The algorithm \( \mathcal{C} \) keeps a list \( L_c \) of tuples \( < \Pi_{i,j}^n, ID_i, ID_j, PK_i, PK_j, t_{i,j}^n, T_{i,j}^n, K_{i,j} > \) on \( L_2 \). Upon receiving a Send query on \( (\Pi_{i,j}^n, M) \) \( \mathcal{C} \) sets \( T_{i,j}^n = M \) if \( M \neq \perp \), \( \mathcal{C} \) returns \( T_{i,j} \) if a tuple \( < \Pi_{i,j}^n, ID_i, ID_j, PK_i, PK_j, t_{i,j}^n, T_{i,j}^n, K_{i,j} > \) is already in the list \( L_c \). Otherwise, \( \mathcal{C} \) does as follows:

1 If \( \Pi_{i,j}^n = \Pi_{i,j}^n, \) it sets \( K_{i,j}^n = t_{i,j}^n = \perp, T_{i,j}^n = v^P, T_{i,j}^n = M, \) then puts a new tuple \( < \Pi_{i,j}^n, ID_i, ID_j, PK_i, PK_j, t_{i,j}^n, T_{i,j}^n, K_{i,j}^n > \) in the list \( L_c \) and returns \( T_{i,j}^n \).

2 Otherwise, it randomly chooses \( K_{i,j}^n \in [0, 1]^k \), \( t_{i,j}^n \in Z_q^\ast \) and sets \( T_{i,j}^n = t_{i,j}^n P, \) then puts a new tuple \( < \Pi_{i,j}^n, ID_i, ID_j, PK_i, PK_j, t_{i,j}^n, T_{i,j}^n, K_{i,j}^n > \) in the list \( L_c \) and returns \( T_{i,j}^n \).

Reveal(\( \Pi_{i,j}^n \)): Upon receiving a Reveal query on \( (\Pi_{i,j}^n) \), the algorithm \( \mathcal{C} \) aborts the game if \( \Pi_{i,j}^n = \Pi_{i,j}^n \) or \( \Pi_{i,j}^n \) is a matching conversation of \( \Pi_{i,j}^n \). Otherwise, \( \mathcal{C} \) searches for a tuple \( < \Pi_{i,j}^n, ID_i, ID_j, PK_i, PK_j, t_{i,j}^n, T_{i,j}^n, K_{i,j}^n > \) in the list \( L_c \) and does the following:

1 If \( < \Pi_{i,j}^n, ID_i, ID_j, PK_i, PK_j, t_{i,j}^n, T_{i,j}^n, K_{i,j}^n > \) is on the list \( L_c \) and \( K_{i,j}^n \neq \perp, \mathcal{C} \) returns \( K_{i,j}^n \).

2 Else if \( < \Pi_{i,j}^n, ID_j, ID_j, PK_j, PK_j, t_{i,j}^n, T_{i,j}^n, K_{i,j}^n > \) is on the list \( L_c \) and \( K_{i,j}^n \neq \perp, \mathcal{C} \) searches for a tuple \( < \Pi_{i,j}^n, SK_i, PK_i, Cert, > \) in the list \( L_c \) performs as follows:

Case 1: \( ID_i = ID_j \): \( \Pi_{i,j}^n \) is an initiator and there exists a tuple \( < ID_i^4, ID_j^8, PK_i^4, PK_j^8, T_i^4, T_j^8, K_i^{(1)}, K_i^{(2)}, K_i^{(3)}, K_i^{(4)}, h_i > \) in the list \( L_c \) such that \( ID_i = ID_j^4, ID_j = ID_j^8, PK_i = PK_j, PK_j^8 = PK_j, PK_j^8 = PK_j, T_i^4 = T_j^4, T_j^4 = T_j^8. \) If it does, \( \mathcal{C} \) computes \( K_{i,j}^n = (SK_i + Cert_i)(T_i^n + PK_i^{(2)} + h_i P_{pub}) - t_{i,j}^n T_j^n, \) and checks whether the following equations hold:

\[
e(PK_i^{(2)} + h_i P_{pub}, T_j^n) = e(K_{i,j}^{(2)}, P),
\]
\[
K_i^{(1)} = (SK_i + Cert_i)(t_{i,j}^n)^n + PK_i^{(2)}, \]
\[
H_1(ID_j, PK_j P_{pub}) = \]

Information Technology and Control 2017/3/46
\[ K_i^{(3)} = t_{ij}^{(3)}PK_i^{(1)} + SK_iT_{ij}^{n} , \]
\[ K_i^{(4)} = t_{ij}^{(4)}T_{ij}^{n} . \]

If \( K_j^{(1)} , K_j^{(2)} , K_j^{(3)} \) and \( K_j^{(4)} \) pass the above validations, then \( C \) sets \( K_j^{n} = h_i \) and returns \( h_i \).

Case 2: \( ID = ID_j , \prod_i^{n} \) is a responder and there exists a tuple \(< ID_j^{4} , ID_j^{5} , PK_i^{4} , PK_i^{5} , T_i^{j} , T_i^{ji} , K_i^{(1)} , K_i^{(2)} , K_i^{(3)} , K_i^{(4)} , h_i > \) in the list \( L_j \) such that \( ID_j^{4} = ID_j^{1} , ID_j^{5} = ID_j^{2} , PK_i^{4} = PK_i^{1} , PK_i^{5} = PK_i^{2} , T_i^{j} = T_i^{j} , T_i^{ji} = T_i^{ji} , K_i^{(3)} , K_i^{(4)} , h_i > \) If so, \( C \) computes \( K_i = (SK_i + Cert_i)(PK_j^{(1)} + PK_j^{(2)} + H_i(ID_j , PK_j)P_{pub} + T_{ij}^{n} , K_j^{(3)} = t_{ij}^{(3)}PK_i^{(1)} + SK_iT_{ij}^{n} , K_j^{(4)} = t_{ij}^{(4)}T_{ij}^{n} . \)

If \( K_i^{(1)} , K_i^{(2)} , K_i^{(3)} \) and \( K_i^{(4)} \) pass the above validations, then \( C \) sets \( K_i^{n} = h_i \) and returns \( h_i \).

Case 3: \( ID \neq ID_j , \prod_i^{n} \) is a responder and there exists a tuple \(< ID_j^{4} , ID_j^{5} , PK_i^{4} , PK_i^{5} , T_i^{j} , T_i^{ji} , K_i^{(1)} , K_i^{(2)} , K_i^{(3)} , K_i^{(4)} , h_i > \) in the list \( L_j \) such that \( ID_j^{4} = ID_j^{1} , ID_j^{5} = ID_j^{2} , PK_i^{4} = PK_i^{1} , PK_i^{5} = PK_i^{2} , T_i^{j} = T_i^{j} , T_i^{ji} = T_i^{ji} . \) If it does, \( C \) checks whether the following equations hold:

\[ e(PK_j^{(2)} + h_iP_{pub} + T_{ij}^{n} ) = e(PK_j^{(2)} , P) \]
\[ K_i^{(1)} = (SK_i + Cert_i + t_{ij}^{(2)})(PK_j^{(1)} + PK_j^{(2)} + H_i(ID_j , PK_j)P_{pub} , K_i^{(3)} = t_{ij}^{(3)}PK_j^{(1)} + SK_iT_{ij}^{n} , K_i^{(4)} = t_{ij}^{(4)}T_{ij}^{n} . \]

If \( K_i^{(1)} , K_i^{(2)} , K_i^{(3)} \) and \( K_i^{(4)} \) pass the above validations, then \( C \) sets \( K_j^{n} = h_i \) and returns \( h_i \).

Case 3: \( ID \neq ID_j , \prod_i^{n} \) is a responder and there exists a tuple \(< ID_i^{4} , ID_i^{5} , PK_i^{4} , PK_i^{5} , T_i^{j} , T_i^{ji} , K_i^{(1)} , K_i^{(2)} , K_i^{(3)} , K_i^{(4)} , h_i > \) in the list \( L_i \) such that \( ID_i^{4} = ID_i^{1} , ID_i^{5} = ID_i^{2} , PK_i^{4} = PK_i^{1} , PK_i^{5} = PK_i^{2} , T_i^{j} = T_i^{j} , T_i^{ji} = T_i^{ji} . \) If it does, \( C \) checks whether the following equations hold:

\[ K_i^{(1)} = (SK_i + Cert_i + t_{ij}^{(2)})(PK_j^{(1)} + PK_j^{(2)} + H_i(ID_j , PK_j)P_{pub} , K_i^{(3)} = t_{ij}^{(3)}PK_j^{(1)} + SK_iT_{ij}^{n} , K_i^{(4)} = t_{ij}^{(4)}T_{ij}^{n} . \]

If \( K_i^{(1)} , K_i^{(2)} , K_i^{(3)} \) and \( K_i^{(4)} \) pass the above validations, then \( C \) sets \( K_j^{n} = h_i \) and returns \( h_i \).

Case 4: \( ID \neq ID_j , \prod_i^{n} \) is a responder and there exists a tuple \(< ID_j^{4} , ID_j^{5} , PK_i^{4} , PK_i^{5} , T_i^{j} , T_i^{ji} , K_i^{(1)} , K_i^{(2)} , K_i^{(3)} , K_i^{(4)} , h_i > \) in the list \( L_j \) such that \( ID_j^{4} = ID_j^{1} , ID_j^{5} = ID_j^{2} , PK_i^{4} = PK_i^{1} , PK_i^{5} = PK_i^{2} , T_i^{j} = T_i^{j} , T_i^{ji} = T_i^{ji} . \) If it does, \( C \) checks whether the following equations hold:

\[ Z = K_j^{(2)} - (SK_j + Cert_j)(T_{ij}^{n} + PK_j^{(2)} + H_i(ID_j , PK_j)P_{pub}) - t_{ij}^{(4)}T_{ij}^{n} \]

where \( K_j^{(4)} = t_{ij}^{(4)}T_{ij}^{n} \) which can be found in the list \( L_j \).

Else if \( \prod_i^{n} \) is a responder, then there exists a tuple \(< ID_j^{4} , ID_j^{5} , PK_i^{4} , PK_i^{5} , T_i^{j} , T_i^{ji} , K_i^{(1)} , K_i^{(2)} , K_i^{(3)} , K_i^{(4)} , h_i > \) in the list \( L_i \) such that \( ID_i^{4} = ID_i^{1} , ID_i^{5} = ID_i^{2} , PK_i^{4} = PK_i^{1} , PK_i^{5} = PK_i^{2} , T_i^{j} = T_i^{j} , T_i^{ji} = T_i^{ji} . \) If it does, \( C \) checks whether the following equations hold:

\[ Z = K_i^{(2)} - Cert_i(PK_j^{(1)} + PK_j^{(2)} + H_i(ID_j , PK_j)P_{pub} + T_{ij}^{n} - SK_jV - t_{ij}^{(4)}T_{ij}^{n} \]

where \( K_i^{(4)} = t_{ij}^{(4)}T_{ij}^{n} \) which can be found in the list \( L_i \).
\[
(t^i_{ij} + \text{Cert}) (PK^{(i)} + U + T^i_{ij}) - SK_j V - K^{(i)} = t^i_{ij} uP = uvP,
\]

where \( K^{(i)} = t^i_{ij} T^i_{ij} \), which can be found in the list \( L_{ij}^s \).

For both cases, we have that \( Z = uvP \). Therefore, the algorithm \( \mathcal{C} \) can return \( Z \) as the solution to the given CDH problem.

To derive the algorithm \( \mathcal{C} \)'s advantage in solving the CDH problem, we define the following events: (1) \( E_1: A_1 \) does not choose \( \prod_{i,j}^t \) as the test oracle; (2) \( E_2: A_1 \) makes an oracle query \( \text{Certificate}(ID_j) \); (3) \( E_3: A_1 \) makes an oracle query \( \text{Reveal}(\prod_{i,j}^t) \).

According to the above simulation, the algorithm \( \mathcal{C} \) aborts the game only when one of the above events happens. It is clear that \( \Pr[-E_1] \geq 1/n(q_m^2) \) as \( I, J \in \{1, 2, \ldots, n_q \} \) and \( T \in \{1, 2, \ldots, n_t \} \). In addition, because \( -E_1 \) implies \( -E_2 \) and \( -E_3 \), we get \( \Pr[-E_1 \land -E_2 \land -E_3] \geq 1/(n_q q_m^2) \).

Since the algorithm \( \mathcal{C} \) selects the correct tuple from the list \( L_{ij}^s \) with probability \( 1/q_m \), it can correctly solve the given CDH problem with advantage \( \epsilon' \geq \frac{\epsilon}{n_q q_m^2} \).

The time complexity of the algorithm \( \mathcal{C} \) is mainly dominated by the running time \( t \) of the adversary \( A_1 \) and the scalar multiplications and pairings performed in the queries. From the simulation above, we obtain that the time complexity of the algorithm \( \mathcal{C} \) is bounded by \( t' \leq t + (q_1 + q_{re} + q_c q_m)O(1) + q_2(4r_m + 9r_p + O(1)) + q_3(3r_m + O(1)) + q_4(r_m + O(1)) + q_5(10r_m + 2r_p + O(1)) \).

**Theorem 3.** Suppose that \( H_1 \) and \( H_2 \) are two random oracles. If there is a Type 2 adversary \( A_2 \) against the security of our CB-AKA protocol with advantage \( \epsilon \) when running in time \( t \), making at most \( q_m \) queries to the oracle \( \text{CreateUser} \), \( q_c \) queries to the oracle \( \text{Corrupt} \), \( q_m \) queries to the oracle \( \text{Send} \), \( q_2 \) queries to the oracle \( \text{Reveal} \) and \( q_4 \) queries to the random oracles \( H_1 \) (1 \( \leq i \leq 2 \), then there exists an algorithm \( \mathcal{C} \) to solve the CDH problem in \( G \) with advantage \( \epsilon' \geq \frac{\epsilon}{n_q q_m^2} \) and running time \( t' \leq t + (q_1 + q_c q_m)O(1) + q_2(4r_m + 9r_p + O(1)) + q_3(3r_m + O(1)) + q_4(r_m + O(1)) + q_5(7r_m + O(1)) \), where \( n \) denotes the maximal number of the protocol sessions that each participant participates in, \( r_m \) and \( r_p \), respectively, denote the time for computing a pairing and a scalar multiplication in \( G \).

**Proof:** Suppose that the algorithm \( \mathcal{C} \) takes as input a random CDH instance \((q, G, P, aP, bP)\), where \( a, b \in Z_q^* \) are unknown to the algorithm \( \mathcal{C} \). To compute \( abP \), \( \mathcal{C} \) interacts with the adversary \( A_2 \) as below:

At the beginning of the game, the algorithm \( \mathcal{C} \) chooses a random value \( s \in Z_q^* \) as the master key \( \text{msk} \) and computes \( P_{\text{pub}} = SP \). It then outputs the public parameters \( \text{params} = \{h, q, G, P, P_{\text{pub}}, H_1, H_2, \text{msk} = s \} \) and the master secret key \( \text{msk} = s \) to \( A_2 \). Furthermore, it selects three different indices \( I, J \in \{1, 2, \ldots, n_q \} \) and \( T \in \{1, 2, \ldots, n_t \} \) at random.

In Phase 1 and Phase 2, the algorithm \( \mathcal{C} \) answers the adversary \( A_2 \)'s queries as below:

\( H(ID_j, PK_i): \) The algorithm \( \mathcal{C} \) keeps a list \( L_i \) of tuples \( (ID_j, PK_i, h) \). Upon receiving a \( H_i \) query on \((ID_j, PK_i)\), the algorithm \( \mathcal{C} \) returns \( h \) if there exists a tuple \((ID_j, PK_i, h) \) in the list \( L_i \). Otherwise, it selects a random value \( h \in Z_q^* \), adds a new tuple \((ID_j, PK_i, h) \) to the list \( L_i \), and returns \( h \).

\( H(ID_j', ID_j, PK_i', PK_i, T_i^1, T_i^2, K^{(i)}_1, K^{(i)}_2, K^{(i)}_3, K^{(i)}_4, h_i): \) The algorithm \( \mathcal{C} \) keeps a list \( L_i \) of tuples \((ID_j, ID_j', PK_i, PK_i', T_i^1, T_i^2, K^{(i)}_1, K^{(i)}_2, K^{(i)}_3, K^{(i)}_4, h) \). Upon receiving a \( H_2 \) query on \((ID_j, ID_j', PK_i, PK_i', T_i^1, T_i^2, K^{(i)}_1, K^{(i)}_2, K^{(i)}_3, K^{(i)}_4, h) \), the algorithm \( \mathcal{C} \) returns \( h \) if there exists a tuple \((ID_j', ID_j, PK_i, PK_i', T_i^1, T_i^2, K^{(i)}_1, K^{(i)}_2, K^{(i)}_3, K^{(i)}_4, h) \) in the list \( L_i \). Otherwise, the algorithm \( \mathcal{C} \) searches for a tuple \((ID_j, ID_j', PK_i, PK_i') \) and performs as follows:

1. If such a tuple exists and satisfies the equations:

\[
e(K^{(i)}_1, P) = e(PK^{(i)}_1 + PK^{(i)}_2 + H_i(ID_j, PK_i) P_{\text{pub}} + T_{ij}^a, PK^{(i)}_1 + PK^{(i)}_2 + H_i(ID_j, PK_i) P_{\text{pub}}),
\]

\[
e(K^{(i)}_2, P) = e(PK^{(i)}_1 + PK^{(i)}_2 + H_i(ID_j, PK_i) P_{\text{pub}} + T_{ij}^a, T_{ij}^a, PK^{(i)}_1 + PK^{(i)}_2 + H_i(ID_j, PK_i) P_{\text{pub}}),
\]

\[
e(K^{(i)}_3, P) = e(T_{ij}^a, PK^{(i)}_1) e(PK^{(i)}_1, T_{ij}^a),
\]

\[
e(K^{(i)}_4, P) = e(T_{ij}^a, T_{ij}^a),
\]

then the algorithm \( \mathcal{C} \) obtains \( K^{(i)}_1 \) and sets \( h = K^{(i)}_1 \).

It adds a new tuple \((ID_j, ID_j', PK_i, PK_i', T_i^1, T_i^2, K^{(i)}_1, K^{(i)}_2, K^{(i)}_3, K^{(i)}_4, h) \) to the list \( L_i \) and returns \( h \).

2. Otherwise, \( \mathcal{C} \) randomly selects \( h \in \{0, 1\}^t \), adds a new tuple \((ID_j, ID_j', PK_i, PK_i', T_i^1, T_i^2, K^{(i)}_1, K^{(i)}_2, K^{(i)}_3, K^{(i)}_4, h) \) to the list \( L_i \) and returns \( h \).
CreateUser($ID$): The algorithm $C$ keeps a list $L_e$ of tuples $<ID, SK, PK, Cert>$. Upon receiving a CreateUser query on ($ID$), the algorithm $C$ returns $PK_i$ if a tuple $<ID, SK, PK, Cert>$ is already in the list $L_e$. Otherwise, it does as below:

1. If $ID = ID_j$, it randomly selects $Cert_i, h_i \in Z^*$, sets $PK_i = (aP, Cert, P - h_3P)$, adds a new tuple $<ID, \perp, PK_i, Cert>$ to the list $L_e$ and returns $PK_i$.

2. Otherwise, it randomly selects $SK_i, Cert_i, h_i \in Z^*_*$, sets $PK_i = (SK, P, Cert, P - h_3P)$, adds a new tuple $<ID, SK, PK, Cert>$ to the list $L_e$ and returns $PK_i$.

Corrupt($ID$): Upon receiving a Corrupt query on ($ID$), the algorithm $C$ aborts if $ID = ID_j$. Otherwise, it retrieves a tuple $<ID, SK, PK, Cert>$ in the list $L_e$ and returns $SK_i$.

Send($\prod_{i,j}^n, M$): The algorithm $C$ keeps a list $L_t$ of tuples $<\prod_{i,j}^n, ID, ID_p, PK_i, PK_j, t_{ij}^*, T_{ij}^*, T_{ij}'^*, K_{ij}^*>$. Upon receiving a Send query on ($\prod_{i,j}^n, M$) ($C$ sets $T_{ij}' = M$ if $M \neq \perp$), $C$ returns $T_{ij}^* \rightarrow A_2$ if a tuple $<\prod_{i,j}^n, ID, ID_p, PK_i, PK_j, t_{ij}^*, T_{ij}^*, T_{ij}'^*, K_{ij}^*>$ is already in the list $L_e$. Otherwise, it does as follows:

1. If $\prod_{i,j}^n = \prod_{i,j}^n$, it sets $K_{ij}^* = t_{ij}^* = \perp, T_{ij}'^* = hP$ and $T_{ij}' = M$. Then it adds a new tuple $<\prod_{i,j}^n, ID, ID_p, PK_i, PK_j, t_{ij}^*, T_{ij}^*, T_{ij}'^*, K_{ij}^*>$ to the list $L_t$ and returns $T_{ij}'$.

2. Otherwise, it randomly chooses $K_{ij}^* \in \{0, 1\}^4$, $t_{ij}^* \in Z^*$ and sets $T_{ij}' = t_{ij}^*P$. Then it puts a new tuple $<\prod_{i,j}^n, ID, ID_p, PK_i, PK_j, t_{ij}^*, T_{ij}^*, T_{ij}'^*, K_{ij}^*>$ in the list $L_t$ and returns $T_{ij}'$.

Reveal($\prod_{i,j}^n$): Upon receiving a Reveal query on ($\prod_{i,j}^n$), the algorithm $C$ aborts the game if $\prod_{i,j}^n = \prod_{i,j}^n$ or $\prod_{i,j}^n$ is a matching conversation of $\prod_{i,j}^n$. Otherwise, $C$ searches for a tuple $<\prod_{i,j}^n, ID, ID_p, PK_i, PK_j, t_{ij}^*, T_{ij}^*, T_{ij}'^*, K_{ij}^*>$ in the list $L_t$.

1. If such a tuple exists and $K_{ij}^* \neq \perp$, it returns $K_{ij}^*$.

2. Else if such a tuple exists and $K_{ij}^* = \perp$, it searches for a tuple $<ID, SK, PK, Cert>$ in the list $L_e$ and does as below:

   Case 1: $\prod_{i,j}^n$ is an initiator and a tuple $<ID_i^3, ID_j^3, PK_i^3, PK_j^3, t_i^3, T_{ij}^*, K_{ij}^1(3), K_{ij}^2(3), K_{ij}^3(3), h_i>$ can be found in the list $L_e$ such that $ID = ID_i^3, ID = ID_j^3, PK_i^3 = PK_i, PK_j^3 = PK_j, T_{ij}^* = T_i^3$ and $T_{ij}^* = T_j^3$. If it does, $C$ checks whether the following equations hold:

\[
K_{ij}^1 = (SK_i + Cert_i + t_{ij}^s)(PK_j^3 + PK_k^2) + H_i(ID_j, PK_j)P_{pub},
\]

\[
K_{ij}^2 = (SK_i + Cert_i + t_{ij}^s)(T_{ij}^* + PK_j^2) + H_i(ID_j, PK_j)P_{pub},
\]

\[
K_{ij}^3 = t_{ij}^s PK_i^3 + SK_i T_{ij}^*,
\]

\[
K_{ij}^4 = t_{ij}^s T_{ij}^*.
\]

If $K_{ij}^1, K_{ij}^2, K_{ij}^3$ and $K_{ij}^4$ pass the above validations, then $C$ sets $K_{ij}^* = h_i$ and returns $h_i$.

- Case 2: $\prod_{i,j}^n$ is a responder and there exists a tuple $<ID_i^4, ID_j^4, PK_i^4, PK_j^4, T_i^4, T_j^4, K_i^1(4), K_i^2(4), K_i^3(4), K_i^4(4), h_i>$ in the list $L_e$ such that $ID = ID_i^4, ID = ID_j^4, PK_i^4 = PK_j^4, PK_j^4 = PK_j, T_i^4 = T_j^4$ and $T_{ij}^* = T_i^4$. If it does, $C$ checks whether the following equations hold:

\[
K_{ij}^1 = (SK_i + Cert_i)(PK_j^4 + PK_k^2) + H_i(ID_j, PK_j)P_{pub} + T_{ij},
\]

\[
K_{ij}^2 = (SK_i + Cert_i)(PK_j^3 + PK_k^2) + H_i(ID_j, PK_j)P_{pub} + T_{ij},
\]

\[
K_{ij}^3 = t_{ij}^s PK_i^4 + SK_i T_{ij}^*,
\]

\[
K_{ij}^4 = t_{ij}^s T_{ij}^*.
\]

If $K_{ij}^1, K_{ij}^2, K_{ij}^3$ and $K_{ij}^4$ pass the above validations, then $C$ sets $K_{ij}^* = h_i$ and returns $h_i$.

3. Otherwise, it randomly selects $K_{ij}^* \in \{0, 1\}^4$ and returns $K_{ij}^*$.  

At the test phase, if the adversary $A_2$ does not ask a Test query on the oracle $\prod_{i,j}^n$, then the algorithm $C$ aborts the game. Otherwise, the algorithm $C$ outputs a random value $x \in \{0, 1\}^4$.

Once the adversary $A_2$ finishes its queries, it returns its guess. Clearly, if the adversary $A_2$ can win the game with non-negligible advantage $\epsilon$, then there must be a tuple $<\prod_{i,j}^n, ID, ID_j, PK_i, PK_j, \perp, bP, T_{ij}^*, T_{ij}', \perp >$ in the list $L_e$. According to the above simulation, if $\prod_{i,j}^n$ is an initiator oracle, then there exists a tuple $<ID_i^4, ID_j^4, PK_i^4, PK_j^4, T_i^4, T_j^4, K_i^1(4), K_i^2(4), K_i^3(4), K_i^4(4), h_i>$ in the list $L_e$ such that $PK_i = PK_j$, where $PK_j^4 = aP$ and $T_i^4 = T_j^4 = bP$; else if $\prod_{i,j}^n$ is a responder oracle, there exist a tuple $<ID_i^3, ID_j^3, PK_i^3, PK_j^3, T_i^3, T_j^3, K_i^1(3), K_i^2(3), K_i^3(3), K_i^4(3), h_i>$.
$K_i^{(2)}, K_i^{(3)}, K_i^{(4)}, h >$ in the list $L_2$ such that $PK_i^t = PK_j^s$, where $PK_i^{(2)} = aP$ and $T_i^h = T_j^h = bP$. For both cases, the algorithm $C$ returns $Z = K_i^{(3)} - SK_i T_j^T$ as a solution to the given CDH problem, where $SK_i$ can be retrieved from the tuple $<ID_i, SK_i, PK_i, Cert_i>$ in the list $L_i$. We can easily deduce that $abP = K_i^{(3)} - SK_i T_j^T$, as $K_j^{(3)} = T_j^h PK_j^{(1)} + SK_i T_j^T$.

To derive $C$’s advantage, we define the following events: (1) $E_i$: $A_2$ does not select $\prod_{i,j}^T$ as the test oracle; (2) $E_{\omega}$: $A_2$ makes an oracle query $\text{Corrupt}(ID_j)$; (3) $E_{\omega}$: $A_2$ makes an oracle query $\text{Reveal}((\prod_{i,j}^T))$.

According to the above simulation, $C$ aborts the game only when one of the above events happens. It is clear that $\Pr[\neg E_i] \geq 1/(n_1 q_{i,n})$ as $I, J \in \{1, 2, \ldots, n\}$ and $T \in \{1, 2, \ldots, n\}$. Because $\neg E_i$ implies both $\neg E_{\omega}$ and $\neg E_{\omega}$, we deduce that $Pr[\neg E_i \land \neg E_{\omega} \land \neg E_{\omega}] \geq 1/(n_1 q_{i,n})$. Since $C$ selects the correct tuple from the list $L_i$ with probability $1/q_{i,n}$, it can solve the CDH problem with advantage $\varepsilon' \geq \frac{\epsilon}{n_1 q_{i,n}^2}$.

The time complexity of the algorithm $C$ is mainly dominated by the running time $\tau$ of the adversary $A_2$ and the scalar multiplications and pairings performed in the queries. From the simulation above, we obtain that the time complexity of the algorithm $C$ is bounded by $\tau' \leq \tau + (q_{m + n} + q_{m + n})O(1) + q_{m}(4\tau_m + 9\tau_p + O(1)) + q_{m}(3\tau_m + O(1)) + q_{m}(\tau_m + O(1)) + q_{m}(\tau_m' + O(1))$.

### 6. Comparison and experimental results

In this section, the computation cost comparison of our protocol and the previous pairing-based CB-AKA protocols [24, 33, 35, 45] is offered.

As described in Table 2, we consider five main cryptographic operations in the comparison: bilinear pairing, exponentiation in the group $G$, scalar multiplication in the group $G$, map-to-point hash and general hash. The computation costs of the compared protocols are listed in Table 3.

To provide a clearer comparison, we implement all the compared protocols by using the multiprecision integer and rational arithmetic cryptographic library (MIRACAL) [38]. The experimental platform is a laptop running Windows XP with 3GHz Intel PIV CPU and 512-MB memory. For the pairing-based CB-AKA protocols in [24, 33, 35, 45], we use the Tate pairing defined over the supersingular elliptic curve $E(F_p^r)$: $y^2 = x^3 + ax + b$, where $a$ and $b$ are two elements such that $\Delta = 4a^3 + 27b^2 \neq 0$ in the finite field $F_p^r$. To achieve the same 1024-bits RSA security level, we use the security parameter secp160r1 recommended by the Standards for Efficient Cryptography Group [43], where $p = 2^{160} - 2^{95} - 1$, $a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF F7 FFFFFFFF$ and $b = 1C97BEFC 54BD7A8B 65ACF89F 81D4D4AD C565FA45$.

We run each protocol ten times and calculate the average running time required for each protocol part.
participant. The experiment results are shown in Table 4. The running time of our protocol is about 19.32% of the protocol in [24], 18.66% of the protocol in [33], 19.62% of the protocol in [35] and 21.30% of the protocol in [45]. The comparison shows that our protocol outperforms the previous pairing-based CB-AKA protocols in the computation efficiency. Therefore, it is more suitable for the computation-limited devices.

7. Conclusion

In this paper, a practical CB-AKA protocol without bilinear pairing is proposed. The proposed protocol is proven secure under the classic CDH assumption in the random oracle model. Due to avoiding the pairing operations, it significantly reduces the computation cost. Compared with the previous pairing-based CB-AKA protocols, it enjoys obvious advantage in the computation performance.

Acknowledgments

We would like to present our thanks to the anonymous reviewers for their helpful comments. This work is supported by the National Natural Science Foundation of China (grant number 61672207), the Fundamental Research Funds for the Central Universities (grant number 2016B10114) and the Natural Science Foundation of Jiangsu Province (grant number BK20161511) and a Project Funded by Jiangsu Collaborative Innovation Center on Atmospheric Environment and Equipment Technology.

References


36. Ni, L., Chen, G., Li, J. H., Hao, Y. Strongly Secure Identity-Based Authenticated Key Agreement Protocols in the
An authenticated key agreement (AKA) protocol is extremely essential to secure communications over insecure public networks. It enables the communication parties to securely set up a shared session key in present of the malicious attackers. Certificate-based cryptography (CBC) is a novel public-key cryptographic primitive that has many attractive merits. It solves the certificate revocation problem in conventional public-key cryptography and the key-escrow problem in identity-based cryptography. Until now, four AKA protocols have been proposed in the setting of CBC. However, all of them adopt the costly bilinear pairings and are not suitable for the devices which have limited computing resources and battery power. Therefore, it is interesting and worthwhile to design a certificate-based AKA protocol without using the bilinear pairings. In this paper, we develop a pairing-free certificate-based AKA protocol. The proposed protocol is proven secure under the classic computational Diffie-Hellman assumption in the random oracle model. Compared with the previous pairing-based certificate-based AKA protocols, the proposed protocol enjoys obvious advantage in the computation efficiency.


Summary / Santrauka