Research Article

A General Architecture for Multiserver Authentication Key Agreement with Provable Security

Yunru Zhang,1,2 Min Luo,1 Kim-Kwang Raymond Choo,3 and Debiao He1,2

1Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China
2Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
3Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, USA

Correspondence should be addressed to Min Luo; mluo@whu.edu.cn

Received 13 September 2018; Accepted 15 October 2018; Published 6 November 2018

Academic Editor: Petros Nicopolitidis

Copyright © 2018 Yunru Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In a typical single-server architecture, when a user wishes to access multiple servers to obtain different services, the user needs to register with every single server. This results in multiple identities and password pairs. To eliminate the limitation of the user having to possess and remember multiple identities and password pairs, a number of multiserver authentication protocols have been proposed where a user only needs to register once. However, most existing protocols are subsequently found to be insecure and this topic remains one of the ongoing research interests. Thus, in this paper, we present a multiserver authentication key agreement protocol. We then demonstrate the security of the protocol under the random oracle model, as well as the practicality of the protocol in terms of low computation and communication costs, minimal storage requirements, and operation costs.

1. Introduction

With the constant development in information and communications technologies (ICT) and digitization of our society, there is an increased trend of users accessing online service (e.g., electronic commerce transactions). Generally speaking, the user interacts with the remote server (e.g., service provider) in an open environment, where an adversary can intercept, modify, or delete data-in-transmit (e.g., messages and transactions between devices and users) [1–4]; see Figure 1. One solution is using authentication or key agreement protocols to authenticate parties involved [5–9]. If the identities of both parties can be verified, then the protocol is said to provide mutual authentication.

Due to the popularity of smart cards, a number of authentication protocols for smart cards in single-server environments have been proposed [10–12]. However, in single-server environments, a user must register on every server that the user wishes to access. This results in the need of users to maintain and ensure the security of the corresponding identities and passwords pairs. Thus, designing secure authentication protocols for smart cards in a multiserver environment has been studied by security researchers [13–16].

Once a user registered (and only once) in a multiserver setting, the user can access services from participating servers using the (one) identities and passwords pair. This benefits both users and servers, in the sense that it reduces the number of identities and password pairs for the user and the size of the verification tables for the servers. For example, in 2001, Li et al. [17] proposed a multiserver architecture based on neural networks. Later in 2003, Lin et al. [18] revealed weaknesses in Li et al.’s protocol, and the time required to train the underpinning neural network limits the utility of the protocol. Hence, Lin et al. proposed a multiserver architecture based on discrete logarithms to achieve better performance which was subsequently found to be insecure against impersonation attack [19]. Juang et al. [20] also presented a protocol designed for symmetric cryptosystem, but this protocol does not provide insider attack resilience and has a high computation cost. The protocol of Chang and Lee [21] for multiserver structure requires that all participants be trusted, which may not be a realistic assumption in practice. In addition, due to
In a latter work, Tsai et al. [22] presented a protocol and the authors use a one-way hash function to reduce the computation cost (since no verification table is required). However, Chen et al. [23] pointed out that Tsai et al.'s protocol is vulnerable to server spoofing attacks. In order to achieve user anonymity, Liao et al. [24] proposed a multiserver authentication protocol with dynamic identities, although it was subsequently pointed out that this protocol does not forward secrecy [25]. Independently, Hsiang et al. [26] revealed other security vulnerabilities in the protocol of Liao et al. [24] (e.g., insecurity against insider attack and server spoofing attack) and proposed an improved protocol. The latter protocol was found to be insecure against server spoofing attack [27], contrary to its security claims. An improved protocol is again proposed [27]. While pointing out that the (improved) protocol in [27] cannot resist impersonation attack and smart card stolen attack, the authors in [28] presented an enhanced multiserver architecture with dynamic identities. Separately, Li et al. [29] revealed that the protocol of Lee et al. [27] cannot resist server spoofing attack and the protocol of Sood et al. [28] cannot resist smart card stolen attack, prior to presenting a smart card-based protocol for a multiserver architecture. However, to achieve mutual authentication, Li et al.’s scheme requires a control server.

It is clear that most existing protocols are not able to resist a range of attacks and this is the contribution we seek to make in this paper. Specifically, we present a general secure architecture for multiserver authentication key agreement protocol. We then prove that the protocol is secure under the random oracle model. We also demonstrate that the architecture enjoys better performance, in the sense of lower computation and communication costs, minimal storage requirements, and lower operation costs.

In the next section, we describe the general architecture of the proposed multiserver authentication protocol. Then in Sections 3 and 4, we analyze the security and performance of the proposed protocol, respectively. We conclude the paper in the last section.

2. Preliminaries

We define the computationally hard mathematical problems.

**Computational Diffie-Hellman (CDH) Problem.** Given a tuple \((g, e, P, Q, aP, bQ)\) in which \((a, b) \in \mathbb{Z}_q^*\), \(P, Q\), and \(g\) are the generators of \(G_1, G_2,\) and \(G_T\), respectively, the purpose of CDH problem is to compute \(\omega = g^{ab} \in G_T\), in which \((a, b) \in \mathbb{Z}_q^*\) are unknown.

2.1. Security Requirements

**Single Registration.** For the convenience of the user, the proposed protocol for multiserver architecture should provide single registration. The user only needs to register at registration center and can freely access services.

**Mutual Authentication.** To protect the safety of participants, the proposed protocol should provide mutual authentication. The communicating participants should authenticate each other.

**User Anonymity.** To protect user’s privacy, the proposed protocol should provide user anonymity. Even though the adversary can interact with the messages, he/she cannot get user’s identity.

**Untraceability.** To provide better privacy protection, the proposed protocol for multiserver architecture should support untraceability. The adversary cannot find any relation and trace users from the messages sent by users.

**Secure Session Key Agreement.** To ensure the security of the messages transmitted in the continuous communication, the proposed protocol should provide a secure session key shared between the participants to encrypt messages.

**Backward and Forward Secrecy.** To ensure the security of the messages transmitted in the previous and future communication, the proposed protocol should provide backward and forward secrecy. Even though the adversary can get the current session key, he/she cannot get the session key generated in the previous and future session.

**Resistance of Various Attacks.** To withstand various attacks in the real environment, the proposed protocol for multiserver architecture should provide resistance to various attacks.


In this section, we propose the general architecture for multiserver authentication protocol. The protocol comprises three parties (i.e., user, server, and registration center) and three phases (i.e., user registration, server registration, and authentication). Prior to communication between a user, say \(U_i\), and a server, say \(S_j\), both parties need to register with a registration center \(RC\); see Figure 2.

The protocol is based on elliptic curve cryptosystem (ECC). \(RC\) chooses an elliptic curve \(E_p\) in the finite field \(GF(p)\), where \(P \in G\). There is an additive cyclic group \(G\) which has \(n\) elements in \(E_p\). \(RC\) then selects a random number \(s \in \{0, 1\}^{160}\) as its private key and computes its public key \(P_{pub} = s \cdot P\). Table 1 describes the notations used in the remaining of this paper.

![Figure 1: An example model of communication between users and remote servers.](image-url)
### Table 1: Summary of notations.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(U_i)</td>
<td>user (i)</td>
</tr>
<tr>
<td>(S_j)</td>
<td>server (j)</td>
</tr>
<tr>
<td>RC</td>
<td>the registration center</td>
</tr>
<tr>
<td>IDS(_i)</td>
<td>the identity of the user (i)</td>
</tr>
<tr>
<td>PW(_U_i)</td>
<td>the password of the user (i)</td>
</tr>
<tr>
<td>IDS(_S_j)</td>
<td>the identity of the server (j)</td>
</tr>
<tr>
<td>(r)</td>
<td>random number selected by the user (i)</td>
</tr>
<tr>
<td>(s)</td>
<td>the secret key of (RC)</td>
</tr>
<tr>
<td>(n)</td>
<td>the number of elements on (E_p)</td>
</tr>
<tr>
<td>(p)</td>
<td>the point on the elliptic curve</td>
</tr>
<tr>
<td>(KU_i)</td>
<td>a secret value of the user (i)</td>
</tr>
<tr>
<td>(KS_j)</td>
<td>a secret value of the server (j)</td>
</tr>
<tr>
<td>(h(\cdot))</td>
<td>one-way hash function</td>
</tr>
<tr>
<td>(\oplus)</td>
<td>bitwise XOR operation</td>
</tr>
<tr>
<td>(|)</td>
<td>concatenation operation</td>
</tr>
</tbody>
</table>

### Figure 2: Atypical multiserver communication architecture.

#### 3.1. User Registration Phase.

The user \(U_i\) sends the registration request \(IDS_{U_i}\) and \(PW_{U_i}\) to \(RC\) and receives a smart card including the private key in this phase. \(U_i\) stores the random number on the smart card, and \(RC\) stores \(IDS_{U_i}\) and \(h(IDS_{U_i} \| s)\) in the list. Figure 3 outlines the user registration phase.

**Step 1 (\(U_i \rightarrow RC : \{IDS_{U_i}, h(PW_{U_i} \| r)\})\).** The user \(U_i\) chooses \(IDS_{U_i}\) and \(PW_{U_i}\) at first and then selects a random number \(r \in [0, 1)^{160}\) and computes \(h(PW_{U_i} \| r)\). At last he sends \(\{IDS_{U_i}, h(PW_{U_i} \| r)\}\) to \(RC\).

**Step 2 (\(RC \rightarrow U_i : \{B_i\})\).** Upon receiving the messages form \(U_i\), \(RC\) computes \(A_i = h(IDS_{U_i} \| s)\) and \(B_i = A_i \oplus h(PW_{U_i} \| r)\). Then \(RC\) stores \(IDS_{U_i}\) and \(h(IDS_{U_i} \| s)\) into the Hash List and sends smart card including \(B_i\) to \(U_i\) through a secure channel.

**Step 3.** When receiving smart card form \(RC\), the user \(U_i\) stores his identity \(IDS_{U_i}\) and the random number \(r\) in the smart card.

#### 3.2. Server Registration Phase.

The server \(S_j\) sends its registration request that is \(IDS_{S_j}\) to \(RC\) and receives its secret key in this phase. \(RC\) stores \(IDS_{S_j}\) and \(h(IDS_{S_j} \| s)\) into the list. Figure 4 outlines the server registration phase.

**Step 1 (\(S_j \rightarrow RC : \{IDS_{S_j}\})\).** The server \(S_j\) chooses \(IDS_{S_j}\) and then sends \(\{IDS_{S_j}\}\) to \(RC\).

**Step 2 (\(RC \rightarrow S_j : \{A_j\})\).** Upon receiving the messages form \(S_j\), \(RC\) computes \(A_j = h(IDS_{S_j} \| s)\) and sends \(A_j\) to \(S_j\). At last \(RC\) stores \(IDS_{S_j}\) and \(h(IDS_{S_j} \| s)\) into the list.

**Step 3.** When receiving the messages form \(RC\), \(S_j\) stores the message \(A_j\).

#### 3.3. Authentication Phase.

\(U_i\) and \(S_j\) authenticate each other by means of \(RC\), and transmitted messages between both parties are verified. A session key is also established between \(U_i\) and \(S_j\). The detailed authentication phase is shown in Figure 5.

**Step 1 (\(U_i \rightarrow S_j : \{m_1\})\).** \(U_i\) inserts the smart card into the terminal and then enters the identity \(IDS_{U_i}\) and password \(PW_{U_i}\). Afterwards the smart card computes the value of \(A_j\); that is, \(A_j = B_i \oplus h(PW_{U_i} \| r)\). The smart card also selects a secret value \(K_{U_i}\) (e.g., \(g^a\) or \(aQ\), where \(g\) is the generator in a multiplicative group and \(Q\) is the generator in elliptic curve). Finally, he uses \(RC\)’s public key \(P_{pub}\) to encrypt the secret value and user’s identity, computes \(m_1 = E_{P_{pub}}(A_j, IDS_{U_i}, K_{U_i}, IDS_{S_j})\), and sends the message \(m_1\) to \(RC\).

**Step 2 (\(S_j \rightarrow RC : \{m_2\})\).** Upon receiving the message from \(U_i\), \(S_j\) selects another secret value \(K_{S_j}\). Afterwards \(S_j\) computes \(m_2 = E_{P_{pub}}(A_j, IDS_{S_j}, K_{S_j}, m_1)\) using \(RC\)’s public key \(P_{pub}\) and sends \(m_2\) to \(RC\).
Step 3 (RC → S_j : \{c_1, Auth_{RC}\}).

(a) Upon receiving the message from S_j, RC decrypts \(m_2\) with its secret key, that is, \(D_s(m_2) = \{A_j, ID_{S_j}, K_{S_j}, m_1\}\). Then RC decrypts \(m_1\) with its secret key, that is, \(D_s(m_1) = \{A_i, ID_{U_i}, K_{U_i}\}\).

(b) RC computes \(h(ID_{S_j} \parallel s)\) and compares it with \(A_j\); if the messages are not equal, then RC terminates this session. Otherwise RC authenticates S_j and proceeds as per protocol specification. Then, it computes \(h(ID_{U_i} \parallel s)\) and compares it with \(A_i\). If they are not equal, RC cancels this session. Otherwise RC authenticates U_i and proceeds as per protocol specification.

(c) RC encrypts some information with S_j's secret value \(K_{S_j}\) and computes the authentication message \(Auth_{RC}\), that is \(c_1 = E_{K_{S_j}}(ID_{U_j}, K_{U_i}, h(ID_{U_i} \parallel s))\). Finally, RC sends \(Auth_{RC}\) and \(c_1\) to S_j.

Step 4 (S_j → U_i : \{c_2, Auth_{S_j}\}).

(a) Upon receiving the messages from RC, S_j decrypts \(c_1\) with its secret value \(K_{S_j}\), that is, \(D_{K_{S_j}}(c_1) = \{K_{U_i}, ID_{U_i}, h(ID_{U_i} \parallel s)\}\), verifies \(Auth_{RC}\). If the verification fails, S_j cancels this session. Otherwise, S_j authenticates RC and proceeds as per protocol specification.

(b) S_j computes the session key and the authentication message \(Auth_{S_j}\) and encrypts some information with U_i's secret value \(K_{U_i}\), i.e., \(SK = K_{U_i} \cdot P\), \(Auth_{S_j} = MAC_{K_{U_i}}(K_{U_i}, K_{S_j}, h(ID_{S_j} \parallel s))\), \(c_2 = \)
Step 5 ($U_i$ verifies $S_j$).

(a) Upon receiving the messages from $S_j$, $U_i$ decrypts $c_2$ with its secret value $K_{U_j}$, that is, $D_{K_{U_j}}(c_2) = \{ID_{S_j}, h(ID_{U_i} \parallel s)\}$. Since only $U_i$ and $RC$ know the value of $h(ID_{U_i} \parallel s)$, $U_i$ compares $A_i$ with $h(ID_{U_i} \parallel s)$ and cancels this session if the verification fails. Otherwise, $U_i$ authenticates $RC$.

(b) $U_i$ computes the session key $SK = K_{U_i} \star K_{S_j} \star P$ and verifies the validity of $Auth_{S_j}$. If it is invalid, then $U_i$ cancels this session. Otherwise, $U_i$ authenticates $S_j$ and proceeds as per protocol specification.


Prior to demonstrating the security of the protocol, we describe the model we work with.

4.1. Security Model. Similar to the approaches in [30, 31], we assume that there are three entities in the model, namely, (protocol) participants, initialization, and the adversary capabilities respectively.

Participants: The parties involved in the protocol are the user $U_i$, server $S_j$, and registration center $RC$. Let $U_i^k/S_j^k$ represent the $k$-th instance of $U_i/S_j$ and execute a protocol, and each instance is also known as an oracle. In general there are three states of an oracle: Accept, Reject, *.

Accept: the $k$-th instance received correct message.

Reject: the $k$-th instance received wrong message.

*: the decision has not been achieved.

Initialization: $U_i$ receives a private key and a smart card from $RC$, and server $S_j$ receives its private key similarly from $RC$. Participants can authenticate each other and establish a session key by executing the protocol. Once $U_i^k$ and $S_j^k$ obtain Accept and a session key is established, it can be said that the three participants authenticate each other and $SK_{U_i^k} = SK_{S_j^k}$.

Adversary capabilities: The adversary $E$ has the capability to eavesdrop, intercept, and modify the messages during the protocol execution, with the aim of obtaining the session key.

(i) $h(m)$: When $E$ executes the query with the message $m$, $RC$ generates a random $r_j \in Z_q^*$, stores $(m, r_j)$ into the Hash List, and returns $r_j$ to $E$.

(ii) ExtractUser(ID$_{U_i}$): When $E$ executes the query with the user $U_i$’s identity $ID_{U_i}$, $RC$ generates $U_i$’s private key and stores it in the list $L_{U_i}^k$.

(iii) ExtractServer(ID$_{S_j}$): When $E$ executes the query with $S_j$’s identity $ID_{S_j}$, $RC$ generates $S_j$’s private key and stores it in the list $L_{S_j}^k$.

(iv) $Send(U_i/S_j, m)$: $E$ sends a message $m$ to $U_i/S_j$. If $m$ is valid and received by $U_i^k/S_j^k$, then $U_i/S_j$ sends a response to $E$, while accepting the session.

(v) Execute($U_i/S_j$): $U_i^k$ and $S_j^k$ output the real messages transmitted in protocol process.

(vi) $Reveal(U_i)$: $E$ queries and obtains session key(s) of session(s) other than the target session.

(vii) $Corrupt(U_i, PW_{U_i}$): $E$ will receive $U_i$’s password $PW_{U_i}$, as long as the user is not associated with the target session (key).

(viii) $Corrupt(U_i, smart card)$: $E$ will receive $U_i$’s secret value $B_i$ as long as the user is not associated with the target session (key).

(ix) $Test(U_i)$: To obtain the session key, $E$ selects a target instance and initiates this query only once. The query is answered as blow:

1. The queried instance $U_i^k/S_j^k$ randomly chooses a number $b \in \{0,1\}$. If $b = 1$, then $E$ will receive the session key. Otherwise, $U_i^k/S_j^k$ randomly chooses a value and returns it back to $E$.

2. In other cases, the queried instance $U_i^k/S_j^k$ does not have the session key and returns ⊥ to adversary $E$.

Let $|D_{U_i}|$ denote the length of the Hash List and the protocol assume the intractability of the Elliptic Curve Diffie-Hellman Problem (ECDH) described below. Definition 1

Elliptic Curve Diffie-Hellman Problem (ECDH) [32]: There are two points $aP$ and $bP$ in an additive group $G$, and it is infeasible to compute point $abP$.

4.2. Security Proof (AKA Security). $E$ executes all the steps in time $t$; hence, he can make execution within $q_h$ hash-queries, $q_a$ send-queries, and $q_e$ execute-queries. Thus the advantage is as follows.

$$\text{Adv}^{\text{ECDH}}_{n,D}(t) = \text{Pr}[\text{Succ}_{c} | \text{Game}_{G_0} | \text{Game}_{G_1}]$$

Proof: We demonstrate the security of the protocol using a series of games. $\text{Succ}_{c}$ indicates that in the query of $\text{Test}(U_i)$ $E$ can correctly guess the value of $b$ for each game $G_i$ ($0 \leq i \leq 5$), and the probability of $E$ winning the game $G_i$ ($0 \leq i \leq 5$) is $\text{Pr} \left[ \text{Succ}_{c} \right]$.

The series of games start from game $G_0$ to game $G_5$:

(i) Game $G_0$: $G_0$ is a game describing the real word with random oracles. In this game, $E$ can access the messages transmitted during authentication process by executing the oracles. If $E$ can win this game, then we can get:

$$\text{Adv}^{\text{ECDH}}_{n,D}(E) = 2 \times \text{Pr} \left[ \text{Succ}_{c} \right] - 1$$

(ii) Game $G_1$: In this game, $E$ simulates the send- and hash-oracle-queries by accessing the Hash List. The send-queries contain five cases: Send($S_j$, $m_1$), Send($RC$, $m_2$), Send($S_j$, $Auth_{RC}$), Send($U_i$, $Auth_{S_j}$), and Send($S_j$, $Auth_{U_i}$). The Hash List consists of $\{x, h(x)\}$, where $x$ is the input of the hash function and $h(x)$ is the output of the value. If $x$ is in the record, then $h(x)$ is the returned value. Otherwise, a value $h(x)$ randomly chosen from $\{0,1\}^{160}$ is returned and $\{x, h(x)\}$ is recorded in the Hash List. $E$ cannot distinguish $G_1$ from $G_0$. If the adversary $E$ can win this game, then we can get:

$$\text{Pr} \left[ \text{Succ}_{c} \right] = \text{Pr} \left[ \text{Succ}_{0} \right] = 0$$
(iii) Game $G_2$: Assume that in the previous games, all send- and hash-oracle-queries are executed correctly, except when a valid $ID_{U_i}$ is queried. As discussed in Sections 3.1 and 3.2, we know that RC stores $h(ID_{U_i}/ID_{S_j} \parallel s)$. Once $E$ queries a valid $ID_{U_i}$, $E$ obtains $U_i/S_j$’s private key $h(ID_{U_i}/ID_{S_j} \parallel s)$. The distribution of the Hash List is restricted for any hash function; thus, the collisions may occur in hash oracle [33], or in the messages transmitted during protocol execution. $G_2$ and $G_1$ are not distinguishable; only if the above conditions occur, we can get the following.

$$\Pr[\text{Succ}_2] - \Pr[\text{Succ}_1] \leq \frac{q_h^2}{2^n} + \frac{(q_a + q_s)^2}{2^n} \tag{4}$$

(iv) Game $G_3$: We consider that all the oracle queries are executed correctly in the previous games, except the authentication messages $\{m_1,m_2,Auth_{RC},Auth_{U_i},Auth_{S_j}\}$ guessed by $E$. In other words, $E$ executes $\text{Send}(U_i/S_j,m)$ and $\text{Execute}(U_i,S_j)$ oracles and receives the corresponding correct answers: $\text{Send}(S_j,m_1),\text{Send}(RC,m_1),\text{Send}(S_j,Auth_{RC}),\text{Send}(U_i,Auth_{S_j}),\text{Send}(S_j,Auth_{U_i}),\text{Execute}(U_i,S_j),\text{Execute}(S_j,RC)$. $G_3$ and $G_2$ are not distinguishable unless the oracle queries are rejected. If $E$ wins this game, we can get the following.

$$\Pr[\text{Succ}_3] - \Pr[\text{Succ}_2] \leq \frac{q_h + q_a + q_s}{2^n} \tag{5}$$

(v) Game $G_4$: In this game, we assume that all oracle queries are executed correctly in the previous games. However only if $E$ guesses the correct private key $A_j/A_j$, then the game will be terminated. If $E$ guesses the correct $A_j = h(ID_{U_i} \parallel s)$, $E$ can impersonate $U_i$ by selecting a secret value $K_E$ and computing $m_1$. In the same way, $E$ can impersonate $S_j$ if $E$ correctly guesses $A_j = h(ID_{S_j} \parallel s)$. At the same time, if $E$ guesses the correct private key $A_j/A_j$, $E$ can distinguish the session key from the random value, whereas $E$ will obtain the correct hash values by guessing the hash one at a time. If $E$ can win this game, then we can get the following.

$$\Pr[\text{Succ}_4] - \Pr[\text{Succ}_3] \leq \frac{q_i}{|D_{RL_i}|} \tag{6}$$

(vi) Game $G_5$: We embed the ECDH problem into the simulation process: namely, An ECDH instance means that given the two points $a \ast P$ and $b \ast P$, we need to compute $a \ast b \ast P$ without knowing the secret values $a$ and $b$. We can first select a target session to challenge, and $E$ will conduct the corresponding oracle queries. We embed the ECDH instance of $a \ast P$ and $b \ast P$ to take the place of $K_{U_i} \ast P$ and $K_{S_j} \ast P$. Suppose that $E$ can decrypt the MAC algorithm in the time range of $t$.

Under these circumstances, we cannot distinguish the game $G_5$ from $G_4$ unless $E$ can intercept or guess the correct secret values $K_{U_i}$ and $K_{S_j}$ and break the security of the underlying MAC algorithm. Thus $E$ must compute $SK = ECDH(K_{U_i},K_{S_j})$.

$$\Pr[\text{Succ}_5] - \Pr[\text{Succ}_4] \leq Adv_{KD}^{ECDH}(t) + \frac{(q_a + q_s)\text{Adv}_{MAC}(t)}{2} \tag{7}$$

From the previous games, it is clear that there are no collisions on the authentication process and queries. At the same time, the authentication values and the hash value cannot be guessed correctly by the adversary $E$. Therefore, the previous games are independent of each other. The only way that $E$ can win the game is to guess the value of $b$. Thus we know the following.

$$\Pr[\text{Succ}_5] = \frac{1}{2} \tag{8}$$

4.3. Analysis of Security Requirements. Single Registration: In this proposed protocol, the user only needs to register at registration center. When user logs in to any server in the system, the login information is sent to RC and verified by RC. Therefore, the protocol can provide single registration.

Mutual Authentication: In Section 3.3, $RC, U_i$ and $S_j$ will authenticate each other.

(i) $RC$ authenticates $U_i$ and $S_j$. Upon receiving message $m_1$, $RC$ decrypts it and $m_1$ and gets $\{A_j, ID_{S_j}, K_{S_j}, A_i, ID_{U_i}, K_{U_i}, ID_{S_j}^{'}\}$. Due to not guessing $s$, there are only $RC$ and $S_j$ computing or knowing $h(ID_{S_j} \parallel s)$ even if $E$ gets $S_j$’s identity $ID_{S_j}$. Thus, RC computes $h(ID_{S_j}^{'} \parallel s)$ and compares it with $A_j$. If they are equal, it can be said that $RC$ authenticates $S_j$. Afterwards the reason why $RC$ authenticates $U_i$ is the same as that of why it authenticates $S_j$. Thus it can be said that $RC$ authenticates $U_i$.

(ii) $U_i$ and $S_j$ authenticate $RC$. Upon receiving message $[c_1,Auth_{RC}]$, $S_j$ decrypts $c_1$ and gets $K_{U_i}$. Then it verifies the validity of $Auth_{RC}$ with the value of $K_{U_i}$. If it is valid, it can be said that $S_j$ authenticates $RC$. $U_i$ decrypts $c_2$ and compares $A_j$ with $h(ID_{U_i} \parallel s)$. If they are equal, it can be said that $U_i$ authenticates RC.

(iii) $U_i$ and $S_j$ authenticate each other. $U_i$ computes session key and verifies $Auth_{S_j}$ on receiving message $[c_2,Auth_{RC}]$. If it is valid, it can be said that $U_i$ authenticates $S_j$. $S_j$ authenticates $U_i$ by verifying $Auth_{U_i}$ with the same argument. If it is valid, it can be said that $S_j$ authenticates $U_i$.

Therefore, the protocol can achieve mutual authentication.

User Anonymity: During the execution of protocol, $U_i$’s identity $ID_{U_i}$ is only contained in messages $m_1$ and $c_1$. While the messages are encrypted by $P_{pub}$ and $K_{S_j}$, respectively, that is, $m_1 = E_{P_{pub}}(A_j,ID_{U_i},K_{U_i})$ and $c_1 =$
Even though $E$ intercepts the messages, as long as he cannot get RC's private key $s$ or server $S_j$'s secret value $K_{S_j}$, $E$ cannot get $ID_{U_i}$.

**Untraceability:** During the execution of protocol, the user generates a random secret value $K_U$ to compute message $m_1 = E_{p_h}(A_i, ID_{U_i}, K_U)$. According to the protocol, we know that user will randomly generate secret values at each execution. Thus the adversary cannot find any relation from the messages sent by $U_i$ and also cannot trace users. Therefore, the protocol can provide untraceability.

**Secure Session Key Agreement:** $U_i$ and $S_j$ compute the session key $SK = ECDH(K_{U_i}, K_S)$ independently. $U_i$ checks the validity of $SK$ by verifying $Auth_{U_i}$, where $Auth_{U_i}$ is encrypted by $U_i$'s private key $K_U$. $S_j$ checks the validity of $SK$ by verifying $Auth_{S_j}$, where $Auth_{S_j}$ is encrypted by $S_j$'s private key $K_{S_j}$. The valid $SK$ can be used in the future communication. Therefore, the proposed protocol is able to provide secure session key agreement.

**Backward and Forward Secrecy:** Assume that $E$ knows the current $SK$. Owing to the session key $SK = ECDH(K_{U_i}, K_S)$, where the secret values $K_U$ and $K_S$ are randomly selected by $U_i$ and $S_j$. Different $K_U$ and $K_S$ are selected with the execution of protocol, and each session key is independent; thus, even if the current session key is known by $E$, he cannot know the previous and coming session key.

**Resistance of Various Attacks:** We will show that the protocol can withstand insider attack, off-line password guessing attack, user impersonation attack, server spoofing attack, modification attack, stolen card attack, and man-in-the-middle attack. The details are shown as follows.

(i) **Insider Attack:** In registration phase, $U_i$ sends $(ID_{U_i}, h(PW_{U_i} \parallel r))$ to RC. Suppose that an insider attacker can get user's information; however, obviously the password $PW_{U_i}$ is not in plain text. Thus the adversary cannot get correct password from the messages. In addition, RC stores $(ID_{U_i}, h(ID_{U_i} \parallel s))$ in a list, and the insider attacker cannot get password from the list. Therefore, the protocol can resist the insider attack.

(ii) **Off-line Password Guessing Attack:** Assume that the adversary steals user's smart card and can extract the information $(ID_{U_i}, B_i, r)$ by the side channel attack, where $B_i = h(ID_{U_i} \parallel s) \oplus h(PW_{U_i} \parallel r)$, $s$ is RC's private key. The adversary can guess the password $PW_{U_i}^*$. However, $E$ cannot verify the correctness from $B_i$ because the password is protected by the secure hash function and RC's private key $s$. Therefore, the protocol can resist off-line password guessing attack.

(iii) **User Impersonation Attack:** $E$ wants to impersonate $U_i$; he should send message $m_E$ to $S_j$ and be verified by RC. $E$ selects two random numbers as his identity $ID_E$ and his secret value $K_E$. Then $E$ computes $A_E = h(ID_E \parallel s^*)$, where $s^*$ is guessed by $E$ as RC's private key. $E$ computes $m_E = E_{p_h}(A_E, ID_E, K_E)$ and sends $m_E$ to $S_j$. Upon receiving the message $m_E$, RC decrypts $m_E$ and computes $h(ID_E \parallel s)$ where $s$ is RC's real private key and compares it with $A_E$. However, $E$ does not know RC's private key $s$. Thus RC cannot authenticate the impersonated user and the protocol cannot resist user impersonation attack.

(iv) **Server Spoofing Attack:** $E$ wants to impersonate $S_j$; he should send legal message $m_E$ to RC and be verified by RC. $E$ selects two random numbers as his identity $ID_E$ and his secret value $K_E$. Then $E$ computes $A_E = h(ID_E \parallel s^*)$, where $s^*$ is guessed as RC's private key. $E$ computes $m_E = E_{p_h}(A_E, ID_E, K_E, m_1)$ and sends $m_E$ to RC. Upon receiving the message $m_E$, RC decrypts $m_E$ and $m_1$, then computes $h(ID_E \parallel s)$, and compares it with $A_E$. However, $E$ does not know RC's private key $s$. Thus RC cannot authenticate the server and the protocol can resist server spoofing attack.

(v) **Stolen Card Attack:** Assume that the adversary steals user's smart card and can extract the information $(ID_{U_i}, B_i, r)$ by the side channel attack, where $B_i = h(ID_{U_i} \parallel s) \oplus h(PW_{U_i} \parallel r)$, $s$ is RC's private key. The adversary can guess the password $PW_{U_i}^*$. However, $E$ cannot verify the correctness from $B_i$ because the password is protected by the secure hash function and RC's private key $s$. Therefore, the protocol can resist stolen cards attack.

(vi) **Modification Attack:** According to the authentication phase, we know that authentication messages $(m_1, m_2)$ are encrypted by public keys. RC can find any modification by checking the equations $A_j = h(ID_{S_j} \parallel s)$ and $A_i = h(ID_{U_i} \parallel s)$. $(c_1, Auth_{RC}, Auth_{S_j})$ are encrypted by $S_j$'s secret value, and $S_j$ cannot decrypt the messages once the messages are modified. $(c_2, Auth_{S_j})$ are encrypted by $U_i$'s secret value, and $U_i$ cannot decrypt the messages once they are modified. Therefore, the protocol can resist modification attack.

(vii) **Man-in-the-Middle Attack:** Based on the above analysis, we conclude that the proposed protocol can provide mutual authentication. Thus the protocol can resist man-in-the-middle attack.

5. **Performance Analysis**

In this section, we analyze the computation complexity, the resistance of various attacks, and communication cost of our architecture for multiserver authentication key agreement. And then we make the comparison with the protocols previously mentioned in the related work; they are the protocol of Sood et al. [28], the protocol of Lee et al. [27], and the protocol of Li et al. [29].

Table 2 is a comparative summary of the computation costs and analysis of various attacks for the four protocols, in which $T_E$ denotes the time complexity of completing a typical hash function, $T_{Im}$ denotes the time complexity of completing an elliptic curve point multiplication operation, $T_{MAC}$ denotes the time complexity of completing a MAC algorithm, and $T_{ED}$ is the time complexity of executing an
of ECC cryptosystem is 160 bits. Thus, we can know that the length of the encrypted data is 160 bits and the output length of the MAC is equal to the input data. Our proposed protocol’s communication cost is, therefore, 2080 bits.

### 6. Conclusion

In this paper, we proposed a general architecture for multiserver authentication key agreement protocol. With it, the user can subscribe many kinds of services just needing one pair of identity and passwords. We prove the architecture is secure and can provide resistance to various attacks under the random oracle model. In addition, our proposed protocol has lower computation and communication costs and good computational efficiency in terms of storage and operation cost.

### Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

### Conflicts of Interest

The authors declare that they have no conflicts of interest.

### Acknowledgments

The work was supported in part by the National Natural Science Foundation of China (Nos. 61572370, 61572379, 61501333) and the fund of the Jiangsu Key Laboratory of Big Data Security & Intelligent Processing (No. BDSIP1807).

### References


