Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.
Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Robust Mutual Authentication with a Key Agreement Scheme for Session Initiation Protocol

Chien-Ming Chen[1], Bin Xiang[1], King-Hang Wang[2], Yong Zhang[3], Tsu-Yang Wu[4]
[1]Harbin Institute of Technology (Shenzhen), Shenzhen, China
[2]Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong
[3]Shenzhen University, Shenzhen, China
[4]College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China

Corresponding author: Chien-Ming Chen (e-mail: chienming@hit.edu.cn).

This paragraph of the first footnote will contain support information, including sponsor and financial support acknowledgment. For example, "This work was supported in part by the U.S. Department of Commerce under Grant BS123456."

ABSTRACT The session initiation protocol (SIP) is the most widely used application layer's control protocol for creating, modifying, and terminating the session process. With the aim to provide secure communication, many authentication schemes have been proposed for SIP. Very recently, a new authentication and key agreement scheme for SIP has been proposed and claimed that it could resist various attacks. However, in this paper, we show that that scheme is vulnerable to an offline password guessing attack and a stolen memory device attack. Furthermore, we show that it lacks verification mechanism for a wrong password, and password updating process is not efficient. In order to mitigate the flaws and inefficiencies, we design a new robust mutual authentication with a key agreement scheme for SIP. The security analysis reveals that our proposed scheme is robust to severe kinds of attacks. Besides, the proposed scheme was simulated by the automatic cryptographic protocol tool Proverif. The performance analysis shows that our proposed scheme is superior to the other related schemes.

INDEX TERMS Key Agreement, Mutual Authentication, Proverif, SIP.

I. INTRODUCTION

The session initiation protocol (SIP) is an application layer's control protocol proposed and studied by the Internet Engineering Task Force (IETF) on the Internet Protocol (IP) network for multimedia communication. The SIP is used to create, modify, and terminate one or more participants' session processes. It supports five aspects in establishing and maintaining the termination of a multimedia session: user location, user effectiveness, user ability, session establishment, and session management. An important feature of SIP is that it does not define the type of a session to establish but only defines how to manage a session. Due to such flexibility, SIP can be used in many applications and services, including interactive games, music and video on demand, and voice, video and Web conferences. The SIP reuses a Multipurpose Internet Mail Extensions (MIME) type description as an e-mail client, so that the conversational related applications can be automatically activated. Moreover, SIP reuses several existing mature Internet services and protocols, such as Domain Name System (DNS), Real-time Transport Protocol (RTP), Resource Reservation Protocol (RSVP), and so on. Therefore, there is no need to introduce new services to support SIP infrastructure since many parts of the infrastructure are in place or ready for use.

When users enjoy the services provided by the SIP, the security has emerged as a major issue because these transmitted data usually contain people's sensitive and private information. To guarantee a secure communication in the SIP, a secure authentication with a key agreement scheme should be executed before the communication begins. For this reason, many related schemes for SIP have been proposed [1-19] in the past few years.

In 2014, Zhang *et al.* [6] proposed a flexible smart card based authentication scheme for session initiation protocol and claimed that it has strong security. However, Irshad *et al.* [7] pointed out that Zhang *et al.*'s scheme is vulnerable to a DOS attack, but it can become more secure by adding a few modifications. They then proposed an improved SIP scheme [7]. Unfortunately, Arshad *et al.* [8] later found that the scheme of Irshad *et al.*'s cannot resist a user impersonation attack. To overcome this weakness, based on ECC, Arshad *et al.* proposed a new efficient and secure scheme [8]. Very recently, Lin *et al.* [10] demonstrated that Arshad *et al.*'s scheme is vulnerable to a server spoofing attack, a DOS attack, a privilege insider attack, and cannot achieve user anonymity. To mitigate these weaknesses, they proposed a new scheme for SIP using the ECC [10].

In this paper, we analyze the security of Lin *et al.*'s anonymous authentication and key agreement SIP scheme. We show that their scheme cannot withstand an offline password guessing attack and a stolen memory device attack. Furthermore, Lin *et al.*'s scheme lacks verification mechanism for a wrong password and password updating process is not efficient. In order to overcome these flaws and inefficiencies, we propose a robust mutual authentication with a key agreement scheme.

The paper is organized as follows. In Section 2, the review of Lin *et al.*'s scheme is presented. In Section 3, the flaws and inefficiencies of the mentioned scheme are described. In Section 4, a SIP scheme is introduced and described in detail. The security analysis of the proposed scheme is given in Section 5. In Section 6, an automatic cryptographic protocol tool Proverif is used to simulate the proposed scheme. The performance analysis is given in Section 7. Lastly, conclusions and our finding are listed in Section 8.

II. Preliminaries

In this section, we introduce the elliptic curve cryptosystem [23], and the model of attacker [24-26].

A. ELLIPTIC CURVE CRYPTOSYSTEM

An elliptic curve denote by E is defined by the form of $y^2 = x^3 + ax + b (mod\ p)$ over a finite field F, where $a, b \in F$ and $4a^2 + 27b^2 \neq 0$. Given a point $P \in E$ and an integer $t \in F$, the point multiplication $t \cdot P = \underbrace{P + P + P \ldots + P}_{t}$($t$ times).

Elliptic Curve Discrete Logarithm Problem (ECDLP): With the given two points $P, tP \in E$, it is computational impossible to obtain the value of t, where $t \in F$

Elliptic Curve Computational Diffie-Hellman Problem (ECCDHP): With the given three points $P, t \cdot P, s \cdot P \in E$, it is hard to compute $ts \cdot P \in E$, $t, s \in F$.

B. MODEL OF ATTACKER

Here, we illustrate the attacker model under the three-factor authentication scheme. An attacker $\mathcal{A}$ has the following capabilities.

- $\mathcal{A}$ has the full control of the public channel, but not the secure channel. That means $\mathcal{A}$ can obtain all the transmitted data in the login and authentication phase.
- $\mathcal{A}$ can alter, delete or replay the data that he captured from the public channel.
- $\mathcal{A}$ has the ability to read or extract the secret data from the stolen smart card issued to user.
- $\mathcal{A}$ can guess either the user's identity or the password, but not both at a time.
- $\mathcal{A}$ knows the authentication scheme since he can be an outsider user or a legal user.

III. REVIEW OF LIN *et al.*'s SCHEME

This section presents Lin *et al.*'s scheme that includes two phases: registration phase, and login and authentication phase. For convenience, the notations used in the rest of the paper are listed in **Table 1**.

TABLE I
THE NOTATIONS AND THEIR DESCRIPTIONS

Notation	Description
ID_i	Client's identity
PW_i	Client's password
P	Base point on ECC
k_s	Server's secret key
K_s	Server's public key ($K_s = k_s \cdot P$)
$h(\cdot)$	Secure hash function
$\parallel$	Concatenation operation
$\oplus$	Exclusive-or operation
$E_s(\cdot)$	Symmetric key encryption under the key s
$D_s(\cdot)$	Symmetric key decryption under the key s

A. REGISTRATION PHASE

A client registers on a remote server via a secure channel following the listed steps.

Step 1. Client selects an identity ID_i, a password PW_i, a random number N_c, and computes $V_i = h(ID_i \parallel PW_i \parallel N_c)$. Then, he submits a registration message $\{ID_i, V_i\}$ to the server.

Step 2. When receives the registration message, the server first checks the validity of ID_i. Then, it computes $A_i = h(ID_i \parallel k_s), B_i = h(A_i \parallel k_s)$ and $C_i = E_{B_i}(V_i)$. After that, it stores $\{A_i, C_i, E_s(\cdot), D_s(\cdot)\}$ into the memory device and issues it to the client

Step 3. On receiving the memory device, the client stores N_c into it

B. LOGIN AND AUTHENTICATION PHASE

A legal client can login to the server by either **Case-1** or **Case-2**. When a client does not want to update his password, he uses **Case-1**; otherwise, he uses **Case-2**. The steps of these two cases are described detailly in the following, and the corresponding procedures are illustrated in **Figure 1** and **Figure 2**.

Client		Server
Input ID_i, PW_i		
$V_i = h(ID_i \parallel PW_i \parallel N_c)$		
Generates nonce r_c, timestamp T_1		
$R_c = r_c \cdot P$		
$F_i = h(V_i \parallel A_i \parallel C_i \parallel T_1)$		
$H_i = E_{r_cK_s}(realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1)$		
	$\xrightarrow{\{H_i, R_c\}}$	$D_{k_sR_c}(H_i) = realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1$
		Verifies the validity of T_1
		$B_i' = h(A_i \parallel k_s)$, $D_{B_i'}(C_i) = V_i'$
		$F_i' = h(V_i' \parallel A_i \parallel C_i \parallel T_1)$
		Checks $F_i' ?= F_i$
		Generates nonce r_s, timestamp T_2
		$R_s = r_s \cdot P$
		$J_i = h(V_i' \oplus F_i' \oplus r_sR_c \oplus T_2)$
		$SK = h(V_i' \oplus r_sR_c \oplus T_2)$
		$L_i = E_{k_sR_c}(R_s \parallel J_i \parallel T_2 \parallel realm)$
$D_{r_cK_s}(L_i) = R_s \parallel J_i \parallel T_2 \parallel realm$	$\xleftarrow{\{L_i\}}$	
Verifies the validity of T_2		
$J_i' = h(V_i \oplus F_i \oplus r_cR_s \oplus T_2)$		
Checks $J_i' ?= J_i$		
$SK = h(V_i \oplus r_cR_s \oplus T_2)$		

FIGURE 1. Login and authentication phase without password updating

1) CASE-1: LOGIN AND AUTHENTICATION PHASE WITHOUT PASSWORD UPDATING

Step 1. Client inserts his memory device and inputs ID_i, PW_i. Then, he computes $V_i = h(ID_i \parallel PW_i \parallel N_c)$. After that, he generates a random integer r_c, current timestamp T_1, and computes $R_c = r_c \cdot P, F_i = h(V_i \parallel A_i \parallel C_i \parallel T_1), k1 = r_cK_s$, $H_i = E_{k1}(realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1)$. Lastly, he sends the *REQUEST* message $\{H_i, R_c\}$ to the server.

Step 2. When receives the *REQUEST* message, the server obtains the data $realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1$ by decrypting H_i with $k2 = k_sR_c$. Then, it verifies the validity of T_1. If T_1 is valid, he computes $B_i' = h(A_i \parallel k_s)$ and obtains V_i' by decrypting C_i. Next, it computes $F_i' = h(V_i' \parallel A_i \parallel C_i \parallel T_1)$ and checks whether $F_i' = F_i$ holds or not. If it holds, the server executes *Step 3*; otherwise, the authentication process is stopped.

Step 3. Server generates a random integer r_s, timestamp T_2, and computes $R_s = r_s \cdot P, J_i = h(V_i' \oplus F_i' \oplus r_sR_c \oplus T_2)$, session key $SK = h(V_i' \oplus r_sR_c \oplus T_2)$, and $L_i = E_{k2}(R_s \parallel J_i \parallel T_2 \parallel realm)$. Finally, server sends the *ACCEPT* message $\{L_i\}$ to the client.

Step 4. On receiving the *ACCEPT* message, the client obtains $R_s \parallel J_i \parallel T_2 \parallel realm$ by decrypting L_i with $k1$. Then, the client verifies the validity of T_2, and if it is valid, he computes $J_i' = h(V_i \oplus F_i \oplus r_cR_s \oplus T_2)$ and checks whether $J_i' = J_i$ holds or not. If it holds, he computes the session key $SK = h(V_i \oplus r_cR_s \oplus T_2)$; otherwise, he stops the authentication process.

Client		Server
Input ID_i, PW_i		
$V_i = h(ID_i \parallel PW_i \parallel N_c)$		
Generates nonce r_c, timestamp T_1		
$R_c = r_c \cdot P$		
$F_i = h(V_i \parallel A_i \parallel C_i \parallel T_1)$		
Selects new password PW_i^{new}		
$V_i^{new} = h(ID_i \parallel PW_i^{new} \parallel N_c)$		
$G_i = h(V_i^{new} \oplus V_i \oplus T_1 \oplus R_c)$		
$H_i = E_{r_cK_s}(realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1 \parallel V_i^{new} \parallel G_i \parallel CHANGEPW)$		
	$\xrightarrow{\{H_i, R_c\}}$	$D_{k_sR_c}(H_i) = realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1 \parallel V_i^{new} \parallel G_i \parallel CHANGEPW$
		Verifies the validity of T_1
		$B_i' = h(A_i \parallel k_s)$, $D_{B_i'}(C_i) = V_i'$
		$F_i' = h(V_i' \parallel A_i \parallel C_i \parallel T_1)$
		$G_i' = h(V_i^{new} \oplus V_i' \oplus T_1 \oplus R_c)$
		Checks $F_i' ?= F_i$, $G_i' = G_i$
		Generates nonce r_s, timestamp T_2
		$R_s = r_s \cdot P$
		$J_i = h(V_i' \oplus F_i' \oplus r_sR_c \oplus T_2)$
		$C_i^{new} = E_{B_i'}(V_i^{new})$
		$SK = h(V_i' \oplus r_sR_c \oplus T_2)$
		$L_i = E_{k_sR_c}(R_s \parallel J_i \parallel T_2 \parallel C_i^{new} \parallel realm)$
$D_{r_cK_s}(L_i) = R_s \parallel J_i \parallel T_2 \parallel C_i^{new} \parallel realm$	$\xleftarrow{\{L_i\}}$	
Verifies the validity of T_2		
$J_i' = h(V_i \oplus F_i \oplus r_cR_s \oplus T_2)$		
Checks $J_i' ?= J_i$		
$SK = h(V_i \oplus r_cR_s \oplus T_2)$		
Replaces C_i with C_i^{new}		

FIGURE 2. Login and authentication phase with password updating

2) CASE-2: LOGIN AND AUTHENTICATION PHASE WITH PASSWORD UPDATING

Step 1. Client inserts his memory device and inputs ID_i, PW_i. Then, he computes $V_i = h(ID_i \parallel PW_i \parallel N_c)$. After that, client generates a random integer r_c and current timestamp T_1, and computes $R_c = r_c \cdot P, F_i = h(V_i \parallel A_i \parallel C_i \parallel T_1)$.

Step 2. Client selects new password PW_i^{new}, and computes $V_i^{new} = h(ID_i \parallel PW_i^{new} \parallel N_c), G_i = h(V_i^{new} \oplus V_i \oplus T_1 \oplus R_c), k1 = r_cK_s$, and $H_i = E_{k1}(realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1 \parallel V_i^{new} \parallel G_i \parallel CHANGEPW)$. Lastly, he sends the *REQUEST* message $\{H_i, R_c\}$ to the server.

Step 3. When receives the *REQUEST* message, server obtains the data $realm \parallel F_i \parallel A_i \parallel C_i \parallel T_1 \parallel V_i^{new} \parallel G_i \parallel CHANGEPW$ by decrypting H_i with $k2 = k_sR_c$. Then, the server verifies the validity of T_1, and if it is valid, he computes $B_i' = h(A_i \parallel k_s)$ and obtains V_i' by decrypting C_i. Next, the server computes $F_i' = h(V_i' \parallel A_i \parallel C_i \parallel T_1), G_i' = h(V_i^{new} \oplus V_i' \oplus T_1 \oplus R_c)$ and checks whether $F_i' = F_i$ and $G_i' = G_i$ hold or not, if they do, server executes *Step 4*; otherwise, server stops the authentication process.

Step 4. Server generates a random integer r_s, timestamp T_2, and computes $R_s = r_s \cdot P, J_i = h(V_i' \oplus F_i' \oplus r_sR_c \oplus T_2), C_i^{new} = E_{B_i'}(V_i^{new})$, $SK = h(V_i' \oplus r_sR_c \oplus T_2)$ and $L_i = E_{k2}(R_s \parallel J_i \parallel T_2 \parallel C_i^{new} \parallel realm)$. Finally, server sends the *ACCEPT* message $\{L_i\}$ to the client

Step 5. On receiving the *ACCEPT* message, the client obtains $R_s \parallel J_i \parallel T_2 \parallel C_i^{new} \parallel realm$ by decrypting L_i with 1 . Then, the client verifies the validity of T_2. If T_2 is valid, he computes $J_i' = h(V_i \oplus F_i \oplus r_cR_s \oplus T_2)$ and checks whether $J_i' = J_i$ holds or not. If it holds, he computes $SK = h(V_i \oplus r_cR_s \oplus T_2)$

and replaces C_i with C_i^{new} ; otherwise, he stops the authentication process.

IV. FLAWS AND INEFFICIENCIES OF LIN *et al.*'s SCHEME

Although Lin *et al.* claimed that their scheme could resist various types of attacks, we have found that their scheme cannot withstand an offline password guessing attack and a stolen memory device attack. Furthermore, the scheme lacks verification mechanism for a wrong password, and password updating process is not efficient. In this section, we describe our findings in detail.

A. OFFLINE PASSWORD GUESSING ATTACK

Lin *et al.* claimed in their work that even when attacker $\mathcal{A}$ extracts secret data $\{A_i, C_i, N_i\}$ stored in the memory device and has the capability to guess the client's identity and password at the same time, he still cannot obtain a true password. However, that is not true in reality. The following steps show that $\mathcal{A}$ can successfully launch an offline password guessing attack to obtain the client's password.

Step 1. $\mathcal{A}$ extracts the secret data $\{A_i, C_i, N_i\}$ stored in the memory device.

Step 2. $\mathcal{A}$ selects an identity $ID_a = A_i$, a password PW_a, and a random number N_a, and computes $V_i = h(ID_a \parallel PW_a \parallel N_a)$. Then, he submits the registration message $\{ID_a, V_a\}$ to the server.

Step 3. When receives the registration message from $\mathcal{A}$, server checks ID_a. Then, it computes $A_a = h(ID_a \parallel k_s) = h(A_i \parallel k_s) = B_i, B_a = h(A_a \parallel k_s)$ and $C_a = E_{B_a}(V_a)$. After that, server stores $\{A_a, C_a, E_s(\cdot), D_s(\cdot)\}$ into the memory device and issues it to the client.

Step 4. On receiving the memory device, $\mathcal{A}$ obtains V_i by decrypting C_i with the key $A_a(\, A_a = B_i)$.

Step 5. $\mathcal{A}$ guesses client's identity ID_i^* and password PW_i^*, and computes $V_i^* = h(ID_i^* \parallel PW_i^* \parallel N_c)$.

Step 6. $\mathcal{A}$ compares V_i^* with V_i. If these two values are equal, then he believes that PW_i^* is a true password and return it; otherwise, he repeats *Step 5.*

B. STOLEN MEMORY DEVICE ATTACK

Stolen memory device attack means that when an attacker $\mathcal{A}$ stoles one certain user's memory device and extracts the data stored in it, then he can impersonate the user to login in the system.

Form the aforementioned analysis we can conclude that when a memory device is lost or stolen, and the secret data stored in it are extracted, it is easy for $\mathcal{A}$ to obtain the client's registered value V_i. In the following, it will be shown that Lin *et al.*'s scheme cannot withstand a stolen memory device attack since $\mathcal{A}$ can impersonate a certain client with V_i. We take **Case-1** of Lin *et al.*'s login and authentication phase as an example.

Step 1. $\mathcal{A}$ extracts secret data $\{A_i, C_i, N_i\}$ stored in a memory device.

Step 2. $\mathcal{A}$ obtains V_i with the assistance of memory device using the *Step 1* to *Step 4* presented in subsection 3.1.

Step 3. $\mathcal{A}$ generates a random integer r_a and timestamp T_1, and then computes $R_a = r_a \cdot P, F_a = h(V_i \parallel A_i \parallel C_i \parallel T_1), k1 = r_a K_s$, $H_a = E_{k1}(realm \parallel F_a \parallel A_i \parallel C_i \parallel T_1)$. Lastly, he sends the *REQUEST* message $\{H_a, R_a\}$ to the server

Step 4. When receives the *REQUEST* message from $\mathcal{A}$, server obtains data $realm \parallel F_a \parallel A_i \parallel C_i \parallel T_1$ by decrypting H_a with $k2 = k_s R_a$. Then, server verifies T_1, which is valid, computes $B_a' = h(A_i \parallel k_s)$, and obtains V_i' by decrypting C_i. Next, server computes $F_a' = h(V_i' \parallel A_i \parallel C_i \parallel T_1)$ and checks F_a'. The same as T_1, the value passes the verification.

Step 5. The server generates r_s and timestamp T_2, and then computes $k2 = k_s R_a$, $R_s = r_s \cdot P$, $J_a = h(V_i' \oplus F_a' \oplus r_s R_a \oplus T_2)$, $SK = h(V_i' \oplus r_s R_a \oplus T_2)$, and $L_a = E_{k2}(R_s \parallel J_a \parallel T_2 \parallel realm)$. Finally, it sends the *ACCEPT* message $\{L_a\}$ to $\mathcal{A}$.

Step 6. On receiving the *ACCEPT* message, $\mathcal{A}$ obtains $R_s \parallel J_a \parallel T_2 \parallel realm$ by decrypting L_a with $k1$. Then, he computes the shared session key $SK = h(V_i \oplus r_a R_s \oplus T_2)$. Up till now, $\mathcal{A}$ is seen as a legal client and establish a session key SK with server. That means, $\mathcal{A}$ can pretend a legal user to login in and obtain his personal information.

C. ABSENCE OF VERIFICATION MECHANISM FOR WRONG PASSWORD

As it is stated in [11-12], in real life, people need to manage a large number of accounts for different applications, so it easily happens that someone inputs a wrong password. The verification mechanism for a wrong password at a device is an ideal feature for the authentication protocol, which not only can reduce needless communication, but also save calculation costs. However, this valuable mechanism is absent in Lin *et al.*'s scheme. The consequence of this shortcoming is that session initiated by a wrong password will be continued until the server finds some errors, and the client will not realize an error of a password until the request is out of time. In this way, much communication and computational resources are wasted, and an authentication process is made ineffective.

D. INEFFICIENCY OF PASSWORD UPDAING

By analyzing some related memory based authentication schemes [11-20], we found that a trend in password updating operations is to carry out this operation without a help from a server. However, in Lin *et al.*'s scheme, when a client wants to update his password, he must login in and establish a session key with a server even when the client does not want to access any of the server's services. Although this is not wrong, it is absolutely not efficient.

V. OUR PROPOSED SCHEME

To mitigate the flaws and inefficiencies we mentioned above, we propose a robust mutual authentication with a key agreement scheme for SIP. Our proposed scheme contains four phases: initialization phase, registration phase, login and authentication phase, and password change phase.

A. INITIALIZATION PHASE

In the initialization phase of our proposed scheme, server initials some parameters: it selects an elliptic curve equation $E_p(a,b)$, a base point $P \in E_p(a,b)$, a secure one-way hash function $h(\cdot)$, and symmetric key encryption/decryption functions $E_s(\cdot)/\ D_s(\cdot)$; it selects a high entropy integer k_s as its secret key and computes $K_s = k_s \cdot P$.

B. REGISTRATION PHASE

When a client desires to access any service provided by a remote server, he must first register on that server. The steps of registration phase are illustrated in **Figure 3** and described in the following.

Step 1. Client selects an identity ID_i and a password PW_i, and generates a random integer b. Then, he computes $HPW_i = h(PW_i \parallel b)$ and sends the registration message $\{ID_i, HPW_i\}$ to the server.

Step 2. On receiving the registration message, server generates a random integer $m(2^4 < m < 2^8)$ and then computes $A_i = h(ID_i \parallel k_s)$, $B_i = h(h(ID_i \parallel HPW_i)\ mod\ m)$, and $C_i = A_i \oplus HPW_i \oplus B_i$. After that, server issues the data $\{B_i, C_i, m, K_s, E_s(\cdot), D_s(\cdot), h\ (\cdot)\}$ into a memory device and sends it to the client

Step 3. When receives the memory device from the server, the client stores b into it. Finally, the memory device contains $\{B_i, C_i, m, b, K_s, E_s(\cdot), D_s(\cdot), h\ (\cdot)\}$.

Client		*Server*
Selects ID_i, PW_i Generates a random integer b $HPW_i = h(PW_i \parallel b)$		
	$\xrightarrow{\{ID_i, HPW_i\}}$	Generates a random integer m $A_i = h(ID_i \parallel k_s)$ $B_i = h(h(ID_i \parallel HPW_i)\ mod\ m)$ $C_i = A_i \oplus HPW_i \oplus B_i$ Stores $\{B_i, C_i, m, K_s, E_s(\cdot), D_s(\cdot), h\ (\cdot)\}$ into a memory device
Stores b into memory device	$\xleftarrow{MD}$	

FIGURE 3. **Registration phase of our proposed scheme**

C. LOGIN AND AUTNENTICATION PHASE

A legal client can submit a login request message to a remote server and obtain various services after being authenticated. The steps of login and authentication are shown in **Figure 4** and explained in the following.

Step 1. Client inserts his memory device and inputs ID_i, PW_i. Then, client computes $HPW_i = h(PW_i \parallel b)$, $B_i^* = h(h(ID_i \parallel HPW_i)\ mod\ m)$. After that, he compares B_i^* with B_i. If they are equal, he executes *Step 2*; otherwise, he stops the process.

Step 2. The client generates a random integer r_c, the current timestamp T_1, and then computes $A_i = C_i \oplus HPW_i \oplus B_i$, $R_c = r_c A_i \cdot P$, $k1 = r_c A_i \cdot K_s$, and $H_i = E_{k1}(realm \parallel ID_i \parallel A_i \parallel T_1)$. Lastly, he sends the *REQUEST* message $\{H_i, R_c\}$ to the server.

Step 3. When receives the *REQUEST* message, the server obtains the data $realm \parallel ID_i \parallel A_i \parallel T_1$ by decrypting H_i with $k2 = k_s \cdot R_c$. Then, the server verifies the validity of T_1. If T_1 is valid, he computes $A_i^* = h(ID_i \parallel k_s)$ and checks whether $A_i^* = A_i$ holds or not. If it holds, the server executes *Step 4*; otherwise, it stops the process.

Step 4. Server generates a random integer r_s and timestamp T_2, and computes $R_s = r_s \cdot P, J_i = r_s \cdot R_c, SK = h(J_i \parallel T_1 \parallel T_2)$, and $L_i = E_{k2}(ID_i \parallel R_s \parallel J_i \parallel T_2 \parallel realm)$. Finally, server sends the *ACCEPT* message $\{L_i\}$ to the client.

Step 5. On receiving the *ACCEPT* message, the client obtains $ID_i \parallel R_s \parallel J_i \parallel T_2 \parallel realm$ by decrypting L_i with $k1$. Then, the client verifies the validity of T_2, and if it is valid, he computes $J_i' = r_c A_i \cdot R_s$ and checks whether $J_i' = J_i$ holds or not. If it holds, he computes the session key $SK = h(J_i' \parallel T_1 \parallel T_2)$; otherwise, the client stops the process.

Client		*Server*
Inputs ID_i, PW_i $HPW_i = h(PW_i \parallel b)$ $B_i^* = h(h(ID_i \parallel HPW_i)\ mod\ m)$ Checks $B_i^*\ ?= B_i$ Generates nonce r_c, timestamp T_1 $A_i = C_i \oplus HPW_i \oplus B_i$ $R_c = r_c A_i \cdot P$ $k1 = r_c A_i \cdot K_s$ $H_i = E_{k1}(realm \parallel ID_i \parallel A_i \parallel T_1)$		
	$\xrightarrow{\{H_i, R_c\}}$	$k2 = k_s \cdot R_c$ $D_{k2}(H_i) = realm \parallel ID_i \parallel A_i \parallel T_1$ Verifies the validity of T_1 $A_i^* = h(ID_i \parallel k_s)$ Checks $A_i^*\ ?= A_i$ Generates nonce r_s, timestamp T_2 $R_s = r_s \cdot P$ $J_i = r_s \cdot R_c$ $SK = h(J_i \parallel T_1 \parallel T_2)$ $L_i = E_{k2}(ID_i \parallel R_s \parallel J_i \parallel T_2 \parallel realm)$
$D_{k1}(L_i) = ID_i \parallel R_s \parallel J_i \parallel T_2 \parallel realm$ Verifies the validity of T_2 $J_i' = r_c A_i \cdot R_s$ Checks $J_i'\ ?= J_i$ $SK = h(J_i' \parallel T_1 \parallel T_2)$	$\xleftarrow{\{L_i\}}$	

FIGURE 4. **Login and authentication phase of our proposed scheme**

D. PASSWORD CHANGE PHASE

When a client wants to change his password, he has to perform the following steps without any help of a remote server.

Step 1. Client inserts his memory device and inputs ID_i and PW_i. He is authenticated before executing *Step 2*.

Step 2. Client inputs a new PW_i^{new} and computes $HPW_i^{new} = h(PW_i^{new} \parallel b)$, $B_i^{new} = h(h(ID_i \parallel HPW_i^{new})\ mod\ m)$, and $C_i^{new} = C_i \oplus HPW_i \oplus B_i \oplus HPW_i^{new} \oplus B_i^{new}$.

Step 3. Client uses B_i^{new} and C_i^{new} to replace B_i and C_i in the memory device.

VI. SECURITY ANALYSIS

This section presents security performance of our proposed scheme, which reveals that our scheme is resistant to severe kinds of attacks, such as an offline password guessing attack, a stolen memory device attack, privilege insider attack, etc.

A. UER ANONYMITY

In our scheme, even an attacker $\mathcal{A}$ has obtained the message $\{H_i, R_c, L_i\}$ transmitted via the public channel, he cannot obtain the true identity ID_i because H_i, R_c and L_i are protected by random integer r_c and server's secret key k_s, which are unknown to $\mathcal{A}$. Therefore, our proposed scheme provides user anonymity.

B. UNTRACEABILITY

In our scheme, the *REQUEST* message $\{H_i, R_c\}$ submitted in the login and authentication phase is different in each communication due to randomly selected integer r_c. Similarly, the back *ACCEPT* message $\{L_i\}$ is also different. Therefore, an attacker $\mathcal{A}$ cannot link any two messages and trace the client. In this way, untraceability is achieved.

C. OFFLINE PASSWORD GUESSING ATTACK

Suppose an attacker $\mathcal{A}$ has obtained client's memory device and extracted secret data $\{B_i, C_i, m, b, K_s\}$ stored in it. Then, $\mathcal{A}$ can guess the possible pair ID_i^*, PW_i^*, and compute $B_i^* = h\big(h(\ ID_i^* \parallel h(PW_i^* \parallel b)\big)\ mod\ m\big)$. However, in our scheme, $\mathcal{A}$ cannot definitely find out the correct pair ID_i^*, PW_i^* by checking $B_i^* = B_i$ since B_i is a "fuzzy verifier" [21-22]. Therefore, our proposed scheme resists offline password guessing attacks.

D. STOLEN MEMORY DEVICE ATTACK

Stolen memory device attacks happen when an attacker $\mathcal{A}$ stoles a memory device, extracts the data stored in it, and logins to the server as a legal client. In our proposed scheme, with the data $\{B_i, C_i, m, b, K_s\}$ stored in the memory devices, to login in the server, $\mathcal{A}$ has to construct a legal *REQUEST* message $\{H_i', R_c'\}$. However, without the client's true identity ID_i and server's private key k_s, he cannot recreate $A_i = h(ID_i \parallel k_s)$, which is essential in H_i and R_c. Therefore, our proposed scheme resists stolen memory device attacks.

E. USER IMPERSONATION ATTACK

Assume that an attacker $\mathcal{A}$ has obtained the *REQUEST* message $\{H_i, R_c\}$ and extracted the data $\{B_i, C_i, m, b, K_s\}$ stored in the client's memory device. When he intends to impersonate the user, he needs to construct H_i' and R_c'. However, as we mentioned before, without the client's true identity ID_i and server's secret key k_s, an attacker cannot recreate A_i. Therefore, our proposed scheme resists user impersonation attacks.

F. SERVER SPOOFING ATTACK

Assume that an attacker $\mathcal{A}$ has obtained all the transmitted messages $\{H_i, R_c, L_i\}$ and intends to masquerade as a server to deceive the client. In this case, he needs to construct a legal *ACCEPT* message $\{L_i'\}$. Unfortunately, without the server's secret key k_s, he cannot decrypt H_i to acquire ID_i, which is used in L_i'. Therefore, our proposed scheme resists server spoofing attacks.

G. PRIVILEGE INSIDER ATTACK

In our proposed scheme, we assume that a privileged insider has obtained $ID_i, HPW_i = h(PW_i \parallel b)$ of a certain legal client in the registration phase. However, without knowing b, he can guess, but he cannot obtain the right password PW_i. Therefore, our proposed scheme resists privilege insider attacks.

H. REPLAY ATTACK

In our proposed scheme, if an attacker $\mathcal{A}$ intercepts the *REQUEST* message $\{H_i, R_c\}$ and replays it later, the server will detect it by checking the timestamp T_1. On the other hand, if $\mathcal{A}$ replays the *ACCEPT* message $\{L_i\}$ from the server, the client can recognize it by checking T_2. Therefore, our proposed scheme resists replay attacks.

I. STOLEN-VERIFIER ATTACK

Stolen-verifier attacks mean that an attacker $\mathcal{A}$ gets some precious information that is stored on a server's end. In our scheme, only information about ID_i is stored in a database. However, H_i and L_i are enciphered, and R_c is protected by k_s; thus, $\mathcal{A}$ cannot utilize ID_i to obtain other values. Therefore, our proposed scheme resists stolen-verifier attacks.

J. FORWARD SECRECY

Forward secrecy means that all the past session keys remain secure even though a server's master key is compromised by an attacker. In our proposed scheme, $SK = h(J_i \parallel T_1 \parallel T_2) = h(r_s A_i R_c \parallel T_1 \parallel T_2) = h(r_c A_i R_s \parallel T_1 \parallel T_2)$. When $\mathcal{A}$ obtains the server's secret key k_s, he can compute $k_s R_c$. With this value, he can decrypt H_i to get T_1,ID_i and decrypt L_i to get T_2, R_s. Furthermore, he can compute $A_i = h(ID_i \parallel k_s)$. However, the values of r_s and r_c are out of his range. Therefore, even $\mathcal{A}$ knows the server's master key, he cannot know any past session keys. Therefore, our proposed scheme provides forward secrecy.

K. KNOWN KEY SECURITY

Our proposed scheme can provide known key security, which means that when authentication and key agreement protocol is executed, both client and server generate a unique session key. In other words, the disclosure of some session keys has no effect on the security of the others. In our proposed scheme, $SK = h(J_i \parallel T_1 \parallel T_2) = h(r_c r_s A_i P \parallel T_1 \parallel T_2)$, where timestamps and random integers are exploited in the computation. Therefore, even $\mathcal{A}$ has known some session keys, without knowing the timestamps and random integers generated in a certain communication, he cannot obtain the needed session key. Hence, our proposed scheme provides known key security.

L. PERFECT FORWARD SECRECY

Perfect forward secrecy means that using the secret keys of server and client, an attacker $\mathcal{A}$ still cannot obtain the previous session keys. In our proposed scheme, the secret key of server is k_s, and that of the client is the data $\{B_i, C_i, m, b, K_s\}$ stored in the memory device. With the above assumption, $\mathcal{A}$ can obtain T_1, T_2, A_i, R_c, and R_s. However, when $\mathcal{A}$ intends to compute the shared session key $SK = h(J_i \parallel T_1 \parallel T_2) = h(r_s A_i R_c \parallel T_1 \parallel T_2) = h(r_c A_i R_s \parallel T_1 \parallel T_2)$, he faces the difficulties of extracting r_c from R_c or r_s from R_s. Therefore, he cannot obtain the previous session keys; thus, our proposed scheme provides perfect forward secrecy.

VII. FORMAL VERIFICATION

We used an automatic cryptographic protocol tool Proverif to show that our proposed scheme is secure. We used Proverif because it can implement the one-way hash function, symmetric and asymmetric encryption, digital signatures, etc. Moreover, various attacks can be reconstructed by Proverif. The code of the scheme is illustrated in the following.

```
(* channel *)
free ch:channel.                                   (* public channel *)
free sch:channel [private].     (* secure channel, used for registering *)

(* shared keys *)
free SKu:bitstring [private].
free SKs:bitstring [private].                           (* constants *)
free k:bitstring [private].                  (* the server's secret key *)
const P:bitstring.                           (* the base point on ecc *)
const realm:bitstring.

(* functions & reductions & equations *)
fun h(bitstring):bitstring.                             (* hash function*)
fun mult(bitstring,bitstring):bitstring.   (* scalar multiplication operation *)
fun mod(bitstring,bitstring):bitstring.              (* modulus operation *)
fun con(bitstring,bitstring):bitstring.        (* concatenation operation *)
reduc forall m:bitstring, n:bitstring; getfirst(con(m,n)) = m.
reduc forall m:bitstring, n:bitstring; getsecond(con(m,n)) = n.
fun senc(bitstring,bitstring):bitstring. (* symmetric encryption *)
reduc forall m:bitstring, key:bitstring; sdec(senc(m,key),key)=m.
fun xor(bitstring,bitstring):bitstring.                  (* XOR operation*)
equation forall m:bitstring, n:bitstring; xor(xor(m,n),n)=m.

(* queries *)
query attacker(SKu).
query attacker(SKs).
query id:bitstring; inj-event(UserAuthed(id)) ==> inj-event(UserStarted(id)).

(* event *)
event UserStarted(bitstring).
event UserAuthed(bitstring).
```

FIGURE 5. The declarations of variables, functions, keys, and other related parameters

There were two types of channels, the private channel for transmitting sensitive messages and public channel for transmitting general messages. The declarations of variables, functions, keys, and other related parameters are shown in **Figure 5**. The processes performed by client and server are presented in **Figure 6** and **Figure 7,** respectively. The main process is shown in **Figure 8**.

```
(* ----- the client's process ----- *)
let ProcessUser =
    new IDi:bitstring;                        (* the client's identity *)
    new PWi:bitstring;                        (* the client's password *)
    new b:bitstring;
    let HPWi = h(con(PWi,b)) in
    out(sch,(IDi,HPWi));
    in(sch,(xBi:bitstring,xCi:bitstring,xK:bitstring,xm:bitstring));
    !
    (
    let HPWi' = h(con(PWi,b)) in
    let Bi' = h(mod(h(con(IDi,HPWi')),xm)) in
    if Bi' = xBi then
    new r1:bitstring;
    new T1:bitstring;
    let Ai = xor(xor(xCi,xBi),HPWi') in
    let Rc = mult(r1,mult(Ai,P)) in
    let key = mult(r1,mult(Ai,xK)) in
    let content = con(con(con(realm,IDi),Ai),T1) in
    let Hi = senc(content,key) in
    out(ch,(Hi,Rc));
    in(ch,xLi:bitstring);
    let substance = sdec(xLi,key) in
    let T2 = getsecond(getfirst(substance)) in
    let Ji = getsecond(getfirst(getfirst(substance))) in
    let Rs = getsecond(getfirst(getfirst(getfirst(substance)))) in
    let Ji' = mult(r1,mult(Ai,Rs)) in
    if Ji' = Ji then
    let SK = h(con(con(Ji',T1),T2)) in
    event UserAuthed(IDi);
    out(ch,senc(SKu,SK));
    0
    ).
```

FIGURE 6. The client's process

```
(* ----- the server's process ----- *)
let UserReg =
    in(sch,(xIDi:bitstring,xHPWi:bitstring));
    new m:bitstring;
    let Ai = h(con(xIDi,k)) in
    let Bi = h(mod(h(con(xIDi,xHPWi)),m)) in
    let Ci = xor(xor(Ai,xHPWi),Bi) in
    let K = mult(k,P) in
    out(sch,(Bi,Ci,K,m));
    0.

let ServerAuth =
    in(ch,(xHi:bitstring,xRc:bitstring));
    let key = mult(k,xRc) in
    let content = sdec(xHi,key) in
    let T1 = getsecond(content) in
    let Ai = getsecond(getfirst(content)) in
    let IDi = getsecond(getfirst(getfirst(content))) in
    let Ai' = h(con(IDi,k)) in
    if Ai' = Ai then
    event UserStarted(IDi);
    new r2:bitstring;
    new T2:bitstring;
    let Rs = mult(r2,P) in
    let Ji = mult(r2,xRc) in
    let SK = h(con(con(Ji,T1),T2)) in
    let substance = con(con(con(con(IDi,Rs),Ji),T2),realm) in
    let Li = senc(substance,key) in
    out(ch,Li);
    out(ch,senc(SKs,SK));
    0.

let ProcessServer = UserReg | ServerAuth.
```

FIGURE 7. The server's process

```
(* ----- Main ----- *)
process (!ProcessServer | !ProcessUser)
```

FIGURE 8. The main process

The results of our proposed scheme are presented in **Figure 9**. From the presented results it can be concluded that the session key is out of an attacker's reach.

```
-- Query inj-event(UserAuthed(id)) ==> inj-event(UserStarted(id))
Completing...
200 rules inserted. The rule base contains 192 rules. 0 rules in the queue.
Starting query inj-event(UserAuthed(id)) ==> inj-event(UserStarted(id))
RESULT inj-event(UserAuthed(id)) ==> inj-event(UserStarted(id)) is true.

-- Query not attacker(SKs[])
Completing...
Starting query not attacker(SKs[])
RESULT not attacker(SKs[]) is true.

-- Query not attacker(SKu[])
Completing...
Starting query not attacker(SKu[])
RESULT not attacker(SKu[]) is true.
```

FIGURE 9. **The declarations of variables, functions, keys, and other related parameters**

VIII. PERFORMANCE ANALYSIS

In this section, the security features and communication cost of the proposed scheme and other related schemes [6-8,10] are compared. The comparison results are presented in **Table 2**, from which it is clear that our proposed scheme performs better in terms of security features. The computation cost of all schemes is listed in **Table 3**, but it only relates to the authentication and key agreement phase since only this phase is frequently utilized.

TABLE 2
COMPARISON OF SECURITY FEATURES

SF	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
[6]	N	Y	N	N	Y	Y	Y	Y	Y	Y
[7]	N	Y	Y	N	Y	Y	Y	Y	Y	Y
[8]	N	Y	Y	Y	N	N	Y	Y	Y	Y
[10]	Y	Y	N	N	Y	Y	Y	N	Y	Y
Our's	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

C1: Provide user anonymity;
C2: Withstand replay attacks;
C3: Withstand offline password guessing attacks;
C4: Withstand user impersonation attacks;
C5: Withstand server spoofing attacks;
C6: Withstand privilege insider attacks;
C7: Withstand stolen-verifier attacks;
C8: Withstand stolen memory device attacks;
C9: Provide known key security
C10: Provide perfect forward secrecy

TABLE 3
COMPARISON OF COMPUTATION COST

Schemes	Computation cost
[6]	$10T_h + 8T_d + 2T_a + 2T_m$
[7]	$8T_h + 6T_d + 4T_{ed} + 4T_m + T_n$
[8]	$8T_h + 4T_d + T_m + T_n$
[10]	$11T_h + 6T_d + 6T_{ed}$
Our's	$6T_h + 6T_d + 4T_{ed} + T_n$

$\boldsymbol{T_h}$: Time for executing a one-way hash function.
$\boldsymbol{T_a}$: Time for executing a point addition operation of an elliptic curve.
$\boldsymbol{T_d}$: Time for executing a scalar multiplication operation of an elliptic curve.
$\boldsymbol{T_m}$: Time for executing modular multiplication operation.
$\boldsymbol{T_{ed}}$: Time for executing encryption or decryption.
$\boldsymbol{T_n}$: Time for executing modular inversion operation

IX. CONCLUSION

In this paper, we first analyze an anonymous and secure authentication scheme for session initiation protocol proposed by other authors. Although the authors claimed that their scheme could resist various attacks, we still found out that it is not robust to an offline password guessing attack and a stolen memory device attack. Moreover, it lacks verification mechanism for wrong password insertion, and password updating is not efficient. To mitigate the flaws and inefficiencies in the investigated scheme and enhance the security, we design a new robust mutual authentication with a key agreement scheme. The results of security analysis and performance analysis show that our proposed scheme is superior to the other related schemes.

APPENDIX

Appendixes, if needed, appear before the acknowledgment.

ACKNOWLEDGMENT

The preferred spelling of the word "acknowledgment" in American English is without an "e" after the "g." Use the singular heading even if you have many acknowledgments. Avoid expressions such as "One of us (S.B.A.) would like to thank ... ." Instead, write "F. A. Author thanks ... ." In most cases, sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page, not here.
REFERENCES AND FOOTNOTES

REFERENCES

[1] C.-C. Yang, R.-C. Wang, and W.-T. Liu, "Secure authentication scheme for session initiation protocol," *Computers & Security*, vol. 24, no. 5, pp. 381-386, Aug. 2005.
[2] L. F. Wu, Y. Q. Zhang, and F. J. Wang, "A new provably secure authentication and key agreement protocol for SIP using ECC," *Computer Standards & Interfaces*, vol. 31, no. 2, pp. 286-291, Feb. 2009.
[3] E.-J. Yoon *et al.*, "Robust mutual authentication with a key agreement scheme for the session initiation protocol," *IETE Technical Review*, vol. 27, no. 3, pp. 203-213, Sep. 2010
[4] D. B. He, J. H. Chen, and R. Zhang, "A more secure authentication scheme for telecare medicine information systems," *Journal of Medical Systems*, vol. 36, no. 3, Jun. 2012.
[5] R. Arshad, and N. Ikram, "Elliptic curve cryptography based mutual authentication scheme for session initiation protocol," *Multimedia tools and applications*, vol. 66, no. 2, pp. 165-178, Sep. 2013.
[6] L. P. Zhang, S. Y. Tang, and Z. H. Cai, "Efficient and flexible password authenticated key agreement for voice over internet protocol session initiation protocol using smart card," *International Journal of communication systems*, vol. 27, no. 11, pp. 2691-2702, Jan. 2013.
[7] A. Irshad *et al.*, "A single round-trip sip authentication scheme for voice over internet protocol using smart card," *Multimedia Tools and Applications*, vol. 74, no. 11, pp. 3967-3984, Jun. 2015.
[8] H. Arshad, and N. Morteza, "An efficient and secure authentication and key agreement scheme for session

initiation protocol using ECC," *Multimedia Tools and Applications*, vol. 75, no. 1, pp. 181-197, Jan. 2016.
[9] S. Kumari *et al.*, "An improved smart card based authentication scheme for session initiation protocol," *Peer-to-Peer Networking and Applications*, vol. 10, no. 1, pp. 92-105, Jan. 2017.
[10] H. Lin, F. T. Wen, and C. X. Du, "An anonymous and secure authentication and key agreement scheme for session initiation protocol," *Multimedia Tools and Applications*, vol. 76, no. 2, pp. 2315-2329, Jan. 2017
[11] X. Li *et al.*, "A robust biometrics based three-factor authentication scheme for global mobility networks in smart city," *Future Generation Computer Systems*, vol. 83, pp. 607-618, Jun. 2018.
[12] N, Doshi *et al.*, "A password based authentication scheme for wireless multimedia systems," *Multimedia Tools and Applications*, vol. 76, no. 24, pp. 25893-25918, Dec. 2017.
[13] Q. Feng *et al.*, "Anonymous biometrics-based authentication scheme with key distribution for mobile multi-server environment," *Future Generation Computer Systems*, vol. 84, pp. 239-251, Jul. 2017.
[14] Q. Jiang *et al.*, "Lightweight three-factor authentication and key agreement protocol for internet-integrated wireless sensor networks," *IEEE Access*, vol. 5, pp. 3376-3392, Mar. 2017.
[15] J. Srinivas, S. Mukhopadhyay, and D. Mishra, "A self-verifiable password based authentication scheme for multi-server architecture using smart card," *Wireless Personal Communications*, vol. 96, no. 4, pp. 6273-6297, Qct. 2017.
[16] D. Kang *et al.*, "Security analysis and enhanced user authentication in proxy mobile IPv6 networks," *PloS one*, vol. 12, no. 7, pp. e0181031, Jul. 2017.
[17] M Luo *et al.*, "A secure and efficient identity‐based mutual authentication scheme with smart card using elliptic curve cryptography," *International Journal of Communication Systems*, vol. 30, no. 16, May 2017.
[18] T. Limbasiya, M. Soni, and S. K. Mishra, "Advanced formal authentication protocol using smart cards for network applicants," *Computers & Electrical Engineering*, vol. 66, pp. 50-63, Feb. 2018.
[19] X. Li *et al.*, "A three-factor anonymous authentication scheme for wireless sensor networks in internet of things environments," *Journal of Network and Computer Applications*, vol.103, pp. 194-204, Feb. 2018.
[20] S. Kumari *et al.*, "A provably secure biometrics-based authenticated key agreement scheme for multi-server environments." *Multimedia Tools and Applications*, vol. 77, no. 2, pp. 2359-2389, Jan. 2018.
[21] Q. Jiang *et al.*, "Efficient end-to-end authentication protocol for wearable health monitoring systems," Computers & *Electrical Engineering*, vol. 63, pp. 182-195, Qct. 2017.
[22] Z. J. Fu *et al.*, "Toward efficient multi-keyword fuzzy search over encrypted outsourced data with accuracy improvement," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 12, pp. 2706-2716, Jul. 2016.
[23] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203-209, Mat. 1987.
[24] V. Sureshkumar, R. Amin *et al.*, "An enhanced bilinear pairing based authenticated key agreement protocol for multiserver environment." *International Journal of Communication Systems*, vol. 30, no. 17, Jun. 2017.
[25] S. Kumari, M.K. Khan *et al.*, "More secure smart card‐based remote user password authentication scheme with user anonymity," *Security and Communication Networks*, vol. 7, no. 11, pp. 2039-2053, Nov. 2013.
[26] S. Kumari, X. Li, F. Wu *et al.*, "A user friendly mutual authentication and key agreement scheme for wireless sensor networks using chaotic maps," *Future Generation Computer Systems*, vol. 63, pp. 56-75, Oct. 2016.