Article

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

Chien-Ming Chen ^{1,2} , Bin Xiang ², King-Hang Wang ³ , Kuo-Hui Yeh ⁴
and Tsu-Yang Wu ^{5,6,*}

¹ College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China; chienming.taiwan@gmail.com

² Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China; jiabinxiang@126.com

³ Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China; kevinw@cse.ust.hk

⁴ Department of Information Management, National Dong Hwa University, Hualien 97401, Taiwan; khyeh@gms.ndhu.edu.tw

⁵ National Demonstration Center for Experimental Electronic Information and Electrical Technology Education, Fujian University of Technology, Fuzhou 350118, China

⁶ Fujian Provincial Key Laboratory of Big Data Mining and Applications, Fujian University of Technology, Fuzhou 350118, China

* Correspondence: wutsuyang@gmail.com

Received: 14 September 2018; Accepted: 20 September 2018; Published: 1 October 2018

Abstract: Session initiation protocol (SIP) is the most widely used application layer control protocol for creating, modifying, and terminating session processes. Many authentication schemes have been proposed for SIP aimed at providing secure communication. Recently, a new authentication and key agreement scheme for SIP has been proposed, and it was claimed that it could resist a variety of attacks. However, in this paper, we show that this scheme is vulnerable to an offline password guessing attack and a stolen memory device attack. Furthermore, we show that it lacks the verification mechanism for a wrong password, and that the password updating process is not efficient. To mitigate the flaws and inefficiencies of this scheme, we design a new robust mutual authentication with a key agreement scheme for SIP. A security analysis revealed that our proposed scheme was robust to several kinds of attacks. In addition, the proposed scheme was simulated by the automatic cryptographic protocol tool ProVerif. A performance analysis showed that our proposed scheme was superior to other related schemes.

Keywords: key agreement; mutual authentication; SIP; cryptanalysis

1. Introduction

Session initiation protocol (SIP) is an application layer control protocol proposed and studied by the Internet Engineering Task Force (IETF) on the Internet Protocol (IP) network for multimedia communication. 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. Because of this flexibility, SIP can be used in many applications and services including interactive games; music and video on demand; and voice, video, and Web conferences. 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, since many parts of the infrastructure are in place or ready for use, there is no need to introduce new services to support the SIP infrastructure.

Although users enjoy the services provided by SIP, security has emerged as a major issue because the transmitted data usually contains people's sensitive and private information. To guarantee a secure communication in 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 [1–11] in the past few years.

In 2014, Zhang et al. [1] proposed a flexible smart card based authentication scheme for SIP and claimed that it has strong security. However, Irshad et al. [2] pointed out that Zhang et al.'s scheme is vulnerable to a DOS attack, and that it could become more secure by adding a few modifications. They then proposed an improved SIP scheme [2]. Unfortunately, Arshad et al. [3] later found that the scheme of Irshad et al. cannot resist a user impersonation attack. To overcome this weakness, Arshad et al. proposed a new efficient and secure scheme based on ECC [3]. Very recently, Lin et al. [4] demonstrated that the scheme of Arshad et al. is vulnerable to a server spoofing attack, a DOS attack, a privilege insider attack, and that it cannot achieve user anonymity. To mitigate these weaknesses, they proposed a new scheme for SIP using ECC.

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 nor a stolen memory device attack. Furthermore, Lin et al.'s scheme lacks a verification mechanism for a wrong password and the password updating process is not efficient. To overcome these flaws and inefficiencies, we propose a robust mutual authentication with a key agreement scheme.

This paper is organized as follows: In Section 2, a review of Lin et al.'s scheme is presented. In Section 3, the flaws and inefficiencies of their scheme are described. In Section 4, a SIP scheme is introduced and described in detail. A 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. A performance analysis is given in Section 7. Lastly, conclusions and our findings are given in Section 8.

2. Review of Lin et al.'s Scheme

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

Table 1. Notations used in this paper.

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 ($K_s = k_s P$)
$ $	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

2.1. Registration Phase

A client registers on a remote server via a secure channel by following the steps listed below:

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

Step 2: When the registration message is received, the server first checks the validity of ID_i . Then, it computes $A_i = h(ID_i || k_s)$, $B_i = h(A_i || k_s)$, and $C_i = E_{B_i}(V_i)$. Next, 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.

2.2. Login and Authentication Phase

A legal client can log in to the server by either Case-1 or Case-2. When a client does not want to update his password, Case-1 is used; otherwise, Case-2 is used. The steps of these two cases are described in the following section, and the corresponding procedures are illustrated in Figures 1 and 2.

Figure 1. Login and authentication phase without password updating.

Figure 2. Login and authentication phase with password updating.

2.2.1. CASE-1: Login and Authentication Phase without Password Updating

- Step 1:** The client inserts the memory device and inputs ID_i and PW_i . Then, $V_i = h(ID_i || PW_i || N_c)$ is computed. Next, a random integer r_c and current timestamp T_1 are generated, and $R_c = r_c \cdot P$, $F_i = h(V_i || A_i || C_i || T_1)$, $k_1 = r_c K_s$, $H_i = E_{k_1}(realm || F_i || A_i || C_i || T_1)$ is computed. Lastly, the REQUEST message $\{H_i, R_c\}$ is sent to the server.
- Step 2:** When the server receives the REQUEST message, it obtains the data $\{realm || F_i || A_i || C_i || T_1\}$ by decrypting H_i with $k_2 = k_s R_c$. Then, it verifies the validity of T_1 . If T_1 is valid, $B'_i = h(A_i || k_s)$ is computed and V'_i is obtained by decrypting C_i . Next, it computes $F'_i = h(V'_i || A_i || C_i || T_1)$ and checks whether $F'_i = F_i$ holds or not. If it holds, then the server executes Step 3; otherwise, the authentication process is stopped.
- Step 3:** The server generates a random integer r_s , timestamp T_2 , and computes $R_s = r_s \cdot P$, $J_i = h(V'_i \oplus F'_i \oplus r_s R_c \oplus T_2)$, session key $SK = h(V'_i \oplus r_s R_c \oplus T_2)$, and $L_i = E_{k_2}(R_s || J_i || T_2 || realm)$. Finally, the server sends the ACCEPT message $\{L_i\}$ to the client.
- Step 4:** On receiving the ACCEPT message, the client obtains $\{R_s || J_i || T_2 || realm\}$ by decrypting L_i with k_1 . Then, the client verifies the validity of T_2 , and if it is valid, $J'_i = h(V_i \oplus F_i \oplus r_c R_s \oplus T_2)$ is computed and checks whether $J'_i = J_i$ holds or not. If it holds, then the session key $SK = h(V_i \oplus r_c R_s \oplus T_2)$ is computed; otherwise, the authentication process is stopped.

2.2.2. CASE-2: Login and Authentication Phase with Password Updating

- Step 1:** The client inserts the memory device and inputs ID_i and PW_i . Then, $V_i = h(ID_i || PW_i || N_c)$ is computed. Next, the client generates a random integer r_c and current timestamp T_1 , and computes $R_c = r_c \cdot P$, $F_i = h(V_i || A_i || C_i || T_1)$.
- Step 2:** The client selects new password PW_i^{new} , and computes $V_i^{new} = h(ID_i || PW_i^{new} || N_c)$, $G_i = h(V_i^{new} \oplus V_i \oplus T_1 \oplus R_c)$, $k_1 = r_c K_s$, and $H_i = E_{k_1}(realm || F_i || A_i || C_i || T_1 || V_i^{new} || G_i || CHANGE PW)$. Lastly, the client sends the REQUEST message $\{H_i, R_c\}$ to the server.
- Step 3:** When the server receives the REQUEST message, it obtains the data $\{realm || F_i || A_i || C_i || T_1 || V_i^{new} || G_i || CHANGE PW\}$ by decrypting H_i with $k_2 = k_s R_c$. Then, the server verifies the validity of T_1 , and if it is valid, $B'_i = h(A_i || k_s)$ is computed. V'_i is obtained by decrypting C_i . Next, the server computes $F'_i = h(V'_i || A_i || C_i || T_1)$, $G'_i = h(V_i^{new} \oplus V'_i \oplus T_1 \oplus R_c)$ and checks whether $F'_i = F_i$ and $G'_i = G_i$ hold or not. If they do, then the server executes Step 4; otherwise, the server stops the authentication process.
- Step 4:** The server generates a random integer r_s , timestamp T_2 , and computes $R_s = r_s \cdot P$, $J_i = h(V'_i \oplus F'_i \oplus r_s R_c \oplus T_2)$, $C_i^{new} = E_{B'_i}(V_i^{new})$, $SK = h(V'_i \oplus r_s R_c \oplus T_2)$, and $L_i = E_{k_2}(R_s || J_i || T_2 || C_i^{new} || realm)$. Finally, the server sends the ACCEPT message $\{L_i\}$ to the client.
- Step 5:** On receiving the ACCEPT message, the client obtains $\{R_s || J_i || T_2 || C_i^{new} || realm\}$ by decrypting L_i with k_2 . Then, the client verifies the validity of T_2 . If T_2 is valid, $J'_i = h(V_i \oplus F_i \oplus r_c R_s \oplus T_2)$ is computed and checks whether $J'_i = J_i$ holds or not. If it holds, $SK = h(V_i \oplus r_c R_s \oplus T_2)$ is computed and C_i is replaced with C_i^{new} ; otherwise, the authentication process is stopped.

3. 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 nor a stolen memory device attack. Furthermore, their scheme lacks a verification mechanism for a wrong password, and the password updating process is not efficient. In this section, we describe our findings in detail.

We first illustrate the attacker model under a three-factor authentication scheme [12–14]. Assume that an attacker A has the following capabilities.

1. A has the full control of the public channel, but not the secure channel. That means A can obtain all the transmitted data in the login and authentication phase.
2. A can alter, delete, or replay the data that was captured from the public channel.

3. A has the ability to read or extract the secret data from the stolen smart card issued to a user.
4. A knows the authentication scheme since A can be an outsider user or a legal user.

3.1. Offline Password Guessing Attack

Lin et al. claimed in their work that even when attacker 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, a true password still cannot be obtained. However, that is not true in reality. The following steps show that A can successfully launch an offline password guessing attack to obtain the client's password.

- Step 1:** A extracts the secret data $\{A_i, C_i, N_i\}$ stored in the memory device.
- Step 2:** A selects an identity $ID_a = A_i$, a password PW_a , and a random number N_a , and then computes $V_i = h(ID_a || PW_a || N_a)$. Then, A submits the registration message $\{ID_a, V_a\}$ to the server.
- Step 3:** When the server receives the registration message from A , it checks ID_a . Then, the server computes $A_a = h(ID_a || k_s) = h(A_i || k_s) = B_i$, $B_a = h(A_a || k_s)$, and $C_a = E_{B_a}(V_a)$. After that, the 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, A obtains V_i by decrypting C_i with the key A_a ($A_a = B_i$).
- Step 5:** A guesses the client's identity ID_i^* and password PW_i^* , and then computes $V_i^* = h(ID_i^* || PW_i^* || N_c)$.
- Step 6:** A compares V_i^* with V_i . If these two values are equal, then A believes that PW_i^* is a true password and returns it; otherwise, A repeats Step 5.

3.2. Stolen Memory Device Attack

A stolen memory device attack means that an attacker A steals a certain user's memory device and extracts the data stored in it, and then A can impersonate the user to log in to the system.

From the above-stated 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 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 A can impersonate a certain client with V_i . **Case-1** of Lin et al.'s login and authentication phase is taken as an example.

- Step 1:** A extracts the secret data $\{A_i, C_i, N_i\}$ stored in a memory device.
- Step 2:** A obtains V_i with the assistance of memory device using Steps 1 to 4 presented in Section 3.1.
- Step 3:** A generates a random integer r_a and timestamp T_1 , and then computes $R_a = r_a \cdot P$, $F_a = h(V_i || A_i || C_i || T_1)$, $k_1 = r_a K_s$, $H_a = E_{k_1}(realm || F_a || A_i || C_i || T_1)$. Lastly, the REQUEST message $\{H_a, R_a\}$ is sent to the server.
- Step 4:** When the server receives the REQUEST message from A , it obtains data $\{realm || F_a || A_i || C_i || T_1\}$ by decrypting H_a with $k_2 = k_s R_a$. Then, the server verifies T_1 , which is valid, computes $B'_a = h(A_i || k_s)$, and obtains V'_i by decrypting C_i . Next, the server computes $F'_a = h(V'_i || A_i || C_i || T_1)$ 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 $k_2 = k_s R_a$, $R_s = r_s \cdot P$, $J_a = h(V'_i \oplus F'_a \oplus r_s R_a \oplus T_2)$, $SK = h(V'_i \oplus r_s R_a \oplus T_2)$, and $L_a = E_{k_2}(R_s || J_a || T_2 || realm)$. Finally, it sends the ACCEPT message $\{L_a\}$ to A .
- Step 6:** On receiving the ACCEPT message, A obtains $\{R_s || J_a || T_2 || realm\}$ by decrypting L_a with k_1 . Then, the shared session key $SK = h(V_i \oplus r_a R_s \oplus T_2)$ is computed. Until now, A was seen as a legal client and established a session key SK with server. This means that A can pretend to be a legal user to log in and obtain the user's personal information.

3.3. Absence of a Verification Mechanism for a Wrong Password

As stated in [15,16], in real life, people need to manage a large number of accounts for different applications, so it can easily happen 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 can not only 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 a session initiated by a wrong password will be continued until the server finds some errors, and the client will not realize there is a password error until the request is out of time. In this way, much communication and computational resources are wasted, and the authentication process is made ineffective.

3.4. Inefficiency of Password Updating

By analyzing some of the related memory-based authentication schemes [15–18], we found that a trend in password updating operations was to carry out this operation without help from a server. However, in Lin et al.'s scheme, when a client wants to update his password, he must log in and establish a session key with the server even when the client does not want to access any of the server's services. Although this is not necessarily wrong, it is absolutely not efficient.

4. The Proposed Scheme

To mitigate the flaws and inefficiencies mentioned above, we propose a robust mutual authentication with a key agreement scheme for SIP. Compared with Lin et al.'s scheme, the proposed scheme has the following advantages:

1. If a user desires to update his password, he does not need help from the server.
2. We add a verification mechanism for a wrong password. This can reduce needless communication and computation cost.
3. To prevent an offline password guessing attack or a stolen memory device attack, we redesign the equations used in the login and authentication phase.

Our proposed scheme contains four phases: Initialization phase, registration phase, login and authentication phase, and password change phase.

4.1. Initialization Phase

In the initialization phase of our proposed scheme, the server initializes 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 $E_s(\cdot)/D_s(\cdot)$. In addition, it selects a high entropy integer k_s as its secret key and computes $K_s = k_s \cdot P$.

4.2. Registration Phase

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

- Step 1:** The client selects an identity ID_i , a password PW_i , and a random number b . Then, the client computes $HPW_i = h(PW_i || b)$ and sends the registration message $\{ID_i, HPW_i\}$ to the server.
- Step 2:** On receiving the registration message, the server generates a random integer m and then computes $A_i = h(ID_i || k_s)$, $B_i = h(h(ID_i || HPW_i) \bmod m)$, and $C_i = A_i \oplus HPW_i \oplus B_i$. After that, the 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 the memory device is received from the server, the client stores b into it. Finally, the memory device contains $\{B_i, C_i, m, b, K_s, E_s(\cdot), D_s(\cdot), h(\cdot)\}$.

Figure 3. Registration phase of our proposed scheme.

4.3. Login and Authentication 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 below:

Figure 4. Login and authentication phase of our proposed scheme.

- Step 1:** The client inserts the memory device and inputs ID_i and PW_i . Then, the client computes $HPW_i = h(PW_i \parallel b)$, $B_i^* = h(h(ID_i \parallel HPW_i) \bmod m)$. After that, B_i^* is compared with B_i . If they are equal, the client executes Step 2; otherwise, the process is stopped.
- Step 2:** The client generates a random integer r_c , the current timestamp T_1 , and then computes $A_i = C_i \oplus HPW_i \oplus B_i$, $R_c = r_c A_i P$, $k_1 = r_c A_i K_s$, and $H_i = E_{k_1}(realm \parallel ID_i \parallel A_i \parallel T_1)$. Lastly, the client sends the REQUEST message $\{H_i, R_c\}$ to the server.
- Step 3:** When the REQUEST message is received, the server obtains the data $\{realm \parallel ID_i \parallel A_i \parallel T_1\}$ by decrypting H_i with $k_2(k_s R_c)$. Then, the server verifies the validity of T_1 . If T_1 is valid, $A_i^* = h(ID_i \parallel k_s)$ is computed and $A_i^* = A_i$ checks whether the validity holds or not. If it holds, then the server executes Step 4; otherwise, it stops the process.

- Step 4:** The server generates a random integer r_s and timestamp T_2 , and then computes $R_s = r_s \cdot P$, $J_i = r_s R_c$, $SK = h(J_i || T_1 || T_2)$, and $L_i = E_{k_2}(ID_i || R_s || J_i || T_2 || realm)$. Finally, the server sends the ACCEPT message $\{L_i\}$ to the client.
- Step 5:** On receiving the ACCEPT message, the client obtains $\{ID_i || R_s || J_i || T_2 || realm\}$ by decrypting L_i with k_1 . Then, the client verifies the validity of T_2 , and if it is valid, the client computes $J'_i = r_c A_i R_s$ and checks whether $J'_i = J_i$ holds or not. If it holds, the client computes the session key $SK = h(J'_i || T_1 || T_2)$; otherwise, the client stops the process.

4.4. Password Change Phase

When a client wants to make a password change, then the following steps must be performed without any help from a remote server.

- Step 1:** The client inserts the memory device and inputs ID_i and PW_i . The client is authenticated before executing Step 2.
- Step 2:** The client inputs a new PW_i^{new} and computes $HPW_i^{new} = h(PW_i^{new} || b)$, $B_i^{new} = h(h(ID_i || HPW_i^{new}) \bmod m)$, and $C_i^{new} = C_i \oplus HPW_i \oplus B_i \oplus HPW_i^{new} \oplus B_i^{new}$.
- Step 3:** The client uses B_i^{new} and C_i^{new} to replace B_i and C_i in the memory device.

5. Security Analysis

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

5.1. User Anonymity

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

5.2. 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 A cannot link any two messages nor trace the client. In this way, untraceability is achieved.

5.3. Offline Password Guessing Attack

Suppose an attacker A has obtained client's memory device and extracted secret data $\{B_i, C_i, m, b, K_s\}$ stored in it. A can then start to guess the possible pair ID_i^* , PW_i^* , and compute $B_i^* = h(h(ID_i^* || h(PW_i^* || b)) \bmod m)$. However, even if B_i^* is equal to B_i , then A still cannot be sure if the guessed ID_i^* and PW_i^* are correct. This is because there are various pairs of (ID_i^*, PW_i^*) which can obtain the same B_i^* . Therefore, the proposed scheme resists offline password guessing attacks.

5.4. Stolen Memory Device Attack

Stolen memory device attacks happen when an attacker A steals a memory device, extracts the data stored in it, and logs in to the server as a legal client [19,20]. In our proposed scheme, with the data $\{B_i, C_i, m, b, K_s\}$ stored in the memory devices, to log in to the server, A has to construct a legal REQUEST message $\{H'_i, R'_c\}$. However, without the client's true identity ID_i and the server's private key k_s , A cannot recreate $A_i = h(ID_i || k_s)$, which is essential in H_i and R_c . Therefore, our proposed scheme resists stolen memory device attacks.

5.5. User Impersonation Attack

Assume that an attacker 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 A intends to impersonate the user, H'_i and R'_c need to be constructed. However, as we mentioned before, without the client's true identity ID_i and server's secret key k_s , A cannot recreate A_i . Therefore, our proposed scheme resists user impersonation attacks.

5.6. Server Spoofing Attack

Assume that an attacker 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, A needs to construct a legal ACCEPT message $\{L'_i\}$. Unfortunately, without the server's secret key k_s , A cannot decrypt H_i to acquire ID_i , which is used in L'_i . Therefore, our proposed scheme resists server spoofing attacks.

5.7. Privileged Insider Attack

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

5.8. Replay Attack

In our proposed scheme, if an attacker 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 A replays the ACCEPT message $\{L_i\}$ from the server, then the client can recognize it by checking T_2 . Therefore, our proposed scheme resists replay attacks.

5.9. Stolen Verifier Attack

Stolen verifier attacks [21,22] occur when an attacker A obtains some precious information that is stored on a server's end. In our scheme, the only information about ID_i is stored in a database. H_i and L_i are enciphered, and R_c is protected by k_s . Thus, A cannot utilize ID_i to obtain other values. Therefore, our proposed scheme resists stolen verifier attacks.

5.10. Forward Secrecy

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

5.11. Known Key Security

Our proposed scheme can provide known key security, which means that when the authentication and key agreement protocol is executed, both the 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, $SK = h(J_i || T_1 || T_2) = h(r_c r_s A_i P || T_1 || T_2)$, where timestamps and random integers are exploited in the computation. Therefore, even if A knows some session keys, without knowing the timestamps and random integers generated in a certain communication, A cannot obtain the needed session key. Hence, our proposed scheme provides known key security.

5.12. Perfect Forward Secrecy

Perfect forward secrecy means that when using the secret keys of the server and client, an attacker A still cannot obtain the previous session keys. In our proposed scheme, the secret key of the 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, A can obtain T_1, T_2, A_i, R_c , and R_s . However, when A intends to compute the shared session key $SK = h(J_i || T_1 || T_2) = h(r_s A_i R_c || T_1 || T_2) = h(r_c A_i R_s || T_1 || T_2)$, A faces difficulties in extracting r_c from R_c or r_s from R_s . Therefore, A cannot obtain the previous session keys. Thus, our proposed scheme provides perfect forward secrecy.

6. Formal Verification

ProVerif is an automatic cryptographic protocol verifier, which is widely used to specify and analyze the security of authenticated key agreement protocols [23–27].

In this section, we used the automatic cryptographic protocol tool ProVerif to show that our proposed scheme is secure. ProVerif was used because it can implement 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 below.

There were two types of channels, the private channel for transmitting sensitive messages and the 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 the client and server are presented in Figures 6 and 7, respectively. The main process is shown in Figure 8.

```

(* 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.

```

(* ----- 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. It can be concluded from these results 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 results.

7. Comparison

In this section, the security features and communication cost of the proposed scheme and other related schemes are compared.

7.1. The Security Features

The proposed scheme was compared with other related schemes [1–4]. The comparison results are presented in Table 2, from which it is clear that our proposed scheme performed better in terms of security features.

Table 2. Comparison of security features.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
[1]	N	Y	N	N	Y	Y	Y	Y	Y	Y
[2]	N	Y	Y	N	Y	Y	Y	Y	Y	Y
[3]	N	Y	Y	Y	N	N	Y	Y	Y	Y
[4]	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.

7.2. Performance Discussion

In this subsection, the estimated time of the proposed scheme was compared with other related protocols [1–4]. First, a personal smart device (iPhone 6s with ARM(armv8-a) CPU, 2GB RAM, and iOS 10.1.1 operating system) was used to calculate the execution time of SHA-256, symmetric encryption/decryption operation (AES-GCM), scalar multiplication on ECC (256bits), and modular multiplication. Our experiment was based on Pairing-Based Cryptography (PBC) library. The PBC library is a free C library that performs the mathematical operations underlying pairing-based cryptosystems.

Each operation was executed 1000 times and the average running times were computed. The experimental result are shown in Table 3. Because the computation of the XOR operation, point addition operation, and modular inversion operation are very trivial, they can be ignored.

The computation costs of all the schemes are listed in Table 4, but it is the authentication and key agreement phase which is frequently utilized.

Table 3. Computation of cryptographic operations.

Symbol	Description	Estimated Time
T_h	One invocation of SHA-256 hash	0.03 ms
T_d	Scalar multiplication on ECC	20.23 ms
T_m	Modular multiplication	4.72 ms
T_{ed}	Symmetric encryption/Decryption operation	0.12 ms

Table 4. Comparison of computation costs.

Schemes	Computation Cost	Estimated Time
[1]	$10T_h + 8T_d + 2T_a + 2T_m$	171.58 ms
[2]	$8T_h + 6T_d + 4T_{ed} + 4T_m + T_n$	140.98 ms
[3]	$8T_h + 4T_d + T_m + T_n$	85.55 ms
[4]	$11T_h + 6T_d + 6T_{ed}$	122.43 ms
Our's	$6T_h + 6T_d + 4T_{ed} + T_n$	122.04 ms

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

8. Conclusions

In this paper, we first analyzed an anonymous and secure authentication scheme for SIP proposed by other authors. Although the authors claimed that their scheme could resist various attacks, we found that it is not robust to an offline password guessing attack nor a stolen memory device attack. Moreover, it lacks a verification mechanism for wrong password insertion, and its password updating is not efficient. To mitigate these flaws and inefficiencies and enhance security, we designed a new robust mutual authentication with a key agreement scheme. The results of security and performance analyses showed that our proposed scheme is superior to the other related schemes.

Author Contributions: C.-M.C. and T.-Y.W. wrote the main concepts of the manuscript; K.-H.Y. and B.X. designed and implemented the experiments; K.-H.W. checked the English writing and organization of the manuscript.

Funding: This research was funded by Science and Technology Development Center, Ministry of Education, China [grant number 2017A13025] and funded by Natural Science Foundation of Fujian Province [grant number 2018J01636].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Zhang, L.; Tang, S.; Cai, Z. Efficient and flexible password authenticated key agreement for voice over internet protocol session initiation protocol using smart card. *Int. J. Commun. Syst.* **2014**, *27*, 2691–2702.
2. Irshad, A.; Sher, M.; Rehman, E.; Ch, S.A.; Hassan, M.U.; Ghani, A. A single round-trip sip authentication scheme for voice over internet protocol using smart card. *Multimedia Tools Appl.* **2015**, *74*, 3967–3984. [[CrossRef](#)]
3. Arshad, H.; Nikooghadam, M. An efficient and secure authentication and key agreement scheme for session initiation protocol using ECC. *Multimedia Tools Appl.* **2016**, *75*, 181–197. [[CrossRef](#)]

4. Lin, H.; Wen, F.; Du, C. An anonymous and secure authentication and key agreement scheme for session initiation protocol. *Multimedia Tools Appl.* **2017**, *76*, 2315–2329. [[CrossRef](#)]
5. Yang, C.C.; Wang, R.C.; Liu, W.T. Secure authentication scheme for session initiation protocol. *Comput. Secur.* **2005**, *24*, 381–386. [[CrossRef](#)]
6. Wu, L.; Zhang, Y.; Wang, F. A new provably secure authentication and key agreement protocol for SIP using ECC. *Comput. Standards Interfaces* **2009**, *31*, 286–291. [[CrossRef](#)]
7. Yoon, E.J.; Shin, Y.N.; Jeon, I.S.; Yoo, K.Y. Robust mutual authentication with a key agreement scheme for the session initiation protocol. *IETE Tech. Rev.* **2010**, *27*, 203–213. [[CrossRef](#)]
8. Arshad, R.; Ikram, N. Elliptic curve cryptography based mutual authentication scheme for session initiation protocol. *Multimedia Tools Appl.* **2013**, *66*, 165–178. [[CrossRef](#)]
9. Kumari, S.; Chaudhry, S.A.; Wu, F.; Li, X.; Farash, M.S.; Khan, M.K. An improved smart card based authentication scheme for session initiation protocol. *Peer-to-Peer Netw. Appl.* **2017**, *10*, 92–105. [[CrossRef](#)]
10. Poltronieri, F.; Fronteddu, R.; Stefanelli, C.; Suri, N.; Tortonesi, M.; Paulini, M.; Milligan, J. A secure group communication approach for tactical network environments. In Proceedings of the 2018 International Conference on Military Communications and Information Systems (ICMCIS), Warsaw, Poland, 22–23 May 2018.
11. Castiglione, A.; D’Arco, P.; De Santis, A.; Russo, R. Secure group communication schemes for dynamic heterogeneous distributed computing. *Future Gener. Comput. Syst.* **2017**, *74*, 313–324. [[CrossRef](#)]
12. Sureshkumar, V.; Amin, R.; Anitha, R. An enhanced bilinear pairing based authenticated key agreement protocol for multiserver environment. *Int. J. Commun. Syst.* **2017**, *30*, e3358. [[CrossRef](#)]
13. Kumari, S.; Khan, M.K. More secure smart card-based remote user password authentication scheme with user anonymity. *Secur. Commun. Netw.* **2014**, *7*, 2039–2053. [[CrossRef](#)]
14. Kumari, S.; Li, X.; Wu, F.; Das, A.K.; Arshad, H.; Khan, M.K. A user friendly mutual authentication and key agreement scheme for wireless sensor networks using chaotic maps. *Future Gener. Comput. Syst.* **2016**, *63*, 56–75. [[CrossRef](#)]
15. Li, X.; Niu, J.; Kumari, S.; Wu, F.; Choo, K.K.R. A robust biometrics based three-factor authentication scheme for global mobility networks in smart city. *Future Gener. Comput. Syst.* **2018**, *83*, 607–618. [[CrossRef](#)]
16. Doshi, N.; Kumari, S.; Mishra, D.; Li, X.; Choo, K.K.R.; Sangaiah, A.K. A password based authentication scheme for wireless multimedia systems. *Multimedia Tools Appl.* **2017**, *76*, 25893–25918. [[CrossRef](#)]
17. Feng, Q.; He, D.; Zeadally, S.; Wang, H. Anonymous biometrics-based authentication scheme with key distribution for mobile multi-server environment. *Future Gener. Comput. Syst.* **2018**, *84*, 239–251. [[CrossRef](#)]
18. Srinivas, J.; Mukhopadhyay, S.; Mishra, D. A self-verifiable password based authentication scheme for multi-server architecture using smart card. *Wirel. Pers. Commun.* **2017**, *96*, 6273–6297. [[CrossRef](#)]
19. Chen, C.M.; Li, C.T.; Liu, S.; Wu, T.Y.; Pan, J.S. A Provable Secure Private Data Delegation Scheme for Mountaineering Events in Emergency System. *IEEE Access* **2017**, *5*, 3410–3422. [[CrossRef](#)]
20. Chen, C.M.; Xu, L.; Wu, T.Y.; Li, C.R. On the security of a chaotic maps-based three-party authenticated key agreement protocol. *J. Netw. Intell.* **2016**, *1*, 61–66.
21. Chen, C.M.; Ku, W.C. Stolen-verifier attack on two new strong-password authentication protocols. *IEICE Trans. Commun.* **2002**, *85*, 2519–2521.
22. Wang, K.H.; Chen, C.M.; Fang, W.; Wu, T.Y. A secure authentication scheme for Internet of Things. *Pervasive Mob. Comput.* **2017**, *42*, 15–26. [[CrossRef](#)]
23. Chen, C.M.; Xiang, B.; Wu, T.Y.; Wang, K.H. An Anonymous Mutual Authenticated Key Agreement Scheme for Wearable Sensors in Wireless Body Area Networks. *Appl. Sci.* **2018**, *8*, 1074. [[CrossRef](#)]
24. Chaudhry, S.A.; Naqvi, H.; Sher, M.; Farash, M.S.; Hassan, M.U. An improved and provably secure privacy preserving authentication protocol for SIP. *Peer-to-Peer Netw. Appl.* **2017**, *10*, 1–15. [[CrossRef](#)]
25. Wu, F.; Xu, L.; Kumari, S.; Li, X.; Shen, J.; Choo, K.K.R.; Wazid, M.; Das, A.K. An efficient authentication and key agreement scheme for multi-gateway wireless sensor networks in IoT deployment. *J. Netw. Comput. Appl.* **2017**, *89*, 72–85. [[CrossRef](#)]

26. Abbasinezhad-Mood, D.; Nikooghadam, M. Efficient anonymous password-authenticated key exchange protocol to read isolated smart meters by utilization of extended chebyshev chaotic maps. *IEEE Trans. Ind. Inform.* **2018**. [[CrossRef](#)]
27. Abbasinezhad-Mood, D.; Nikooghadam, M. Design and hardware implementation of a security-enhanced elliptic curve cryptography based lightweight authentication scheme for smart grid communications. *Future Gener. Comput. Syst.* **2018**, *84*, 47–57. [[CrossRef](#)]

© 2018 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).