Research Article

A Robust and Effective Smart-Card-Based Remote User Authentication Mechanism Using Hash Function

Ashok Kumar Das,1 Vanga Odelu,2 and Adrijit Goswami2

1 Center for Security, Theory and Algorithmic Research, International Institute of Information Technology, Hyderabad 500 032, India
2 Department of Mathematics, Indian Institute of Technology, Kharagpur 721 302, India

Correspondence should be addressed to Ashok Kumar Das; ashok.das@iiit.ac.in

Received 25 August 2013; Accepted 3 February 2014; Published 29 April 2014

Academic Editors: G. Bordogna and W.-J. Hwang

Copyright © 2014 Ashok Kumar Das et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In a remote user authentication scheme, a remote server verifies whether a login user is genuine and trustworthy, and also for mutual authentication purpose a login user validates whether the remote server is genuine and trustworthy. Several remote user authentication schemes using the password, the biometrics, and the smart card have been proposed in the literature. However, most schemes proposed in the literature are either computationally expensive or insecure against several known attacks. In this paper, we aim to propose a new robust and effective password-based remote user authentication scheme using smart card. Our scheme is efficient, because our scheme uses only efficient one-way hash function and bitwise XOR operations. Through the rigorous informal and formal security analysis, we show that our scheme is secure against possible known attacks. We perform the simulation for the formal security analysis using the widely accepted AVISPA (Automated Validation Internet Security Protocols and Applications) tool to ensure that our scheme is secure against passive and active attacks. Furthermore, our scheme supports efficiently the password change phase always locally without contacting the remote server and correctly. In addition, our scheme performs significantly better than other existing schemes in terms of communication, computational overheads, security, and features provided by our scheme.

1. Introduction

In recent years, the remote user authentication using smart cards has become an important research area in computer science. In remote user authentication, communicating parties are verified as to whether they are genuine and trustworthy and the users are authenticated by a remote server before allowing access to services. Several password-based schemes (e.g., [1–3]) or biometric-based schemes (e.g., [4–6]) have been proposed for remote user authentication problem. An idle password-based remote user authentication scheme using smart cards needs to satisfy the following requirements [2]:

(i) not maintaining verification tables;
(ii) user’s ability to freely choose and update password;
(iii) resistance to password disclosure to the server;
(iv) prevention of masquerade attacks;
(v) resistance to replay, modification, parallel session, and stolen-verifier attacks;
(vi) an easy-to-remember password;
(vii) low communication cost and computation complexity;
(viii) achieving mutual authentication between login users and remote servers;
(ix) resistance to guessing attacks even if the smart card is lost or stolen by attackers;
(x) session key agreement;
(xi) resistance to insider attacks;
(xii) prevention of smart card security breach attacks.
The majority of the proposed password-based remote user authentication schemes are either computationally expensive or vulnerable to different known attacks. Some comprehensive surveys on password-based remote user authentication schemes could be found in [7, 8]. Das et al. [9] proposed a dynamic ID and password-based remote user authentication scheme using smart cards, which uses the efficient hash function and bitwise XOR operations. However, Wang et al. [10] showed that Das et al.'s scheme is vulnerable to different attacks and it does not achieve mutual authentication property and does not resist impersonating remote server attack. Wang et al. then proposed an enhancement of their scheme using smart cards. Later, Khan et al. [11] analyzed the security of Wang et al.'s scheme and they showed that Wang et al.'s scheme has several weaknesses, for example, it does not provide anonymity of a user during authentication, the user has no choice in choosing his/her password, it is vulnerable to insider attack, it has no provision for revocation of lost or stolen smart card, and, finally, it does not provide session key agreement. In order to remedy these security weaknesses, Khan et al. also proposed an enhanced password-based remote user authentication scheme using smart cards.

In 2012, Sonwanshi et al. [3] proposed a password-based remote user authentication scheme using the smart card, which uses only the one-way hash function and bitwise XOR operation. However, in 2013, Das et al. [12] showed that their scheme is vulnerable to the offline password guessing attack and stolen smart card attack. In addition, Das et al. showed that their scheme fails to protect strong replay attack. In 2013, Lee and Liu [13] proposed a password-based authentication and key agreement scheme, which uses the public-key cryptosystem and one-way hash function. Lee-Liu’s scheme is expensive in computation as it requires expensive modular exponentiation operations. Lee-Liu’s scheme supports session key security and protects against parallel session attack, password guessing attack, privileged insider attack, replay attack, and man-in-the-middle attack. Their scheme also provides user’s anonymity property. In 2013, Das and Bruhadeshwar [14] showed that Lee-Liu’s scheme has two security weaknesses: (1) it has design flaws in authentication phase and (2) it has design flaws in password change phase. In order to withstand these flaws found in Lee-Liu’s scheme, they proposed an improved and effective password-based remote user authentication scheme. However, Das-Bruhadeshwar’s scheme [14] is also computationally costly as it requires expensive modular exponentiation operations. Recently, in 2013, Jiang et al. [15] proposed a secure password-based remote user authentication scheme without pairings for multiserver architecture. However, their scheme uses ECC (elliptic curve cryptography) cryptosystem and hash function. Due to expensive ECC point addition and scalar multiplication operations, their scheme is also expensive.

In this paper, we propose a new robust and secure password-based remote user authentication scheme using the one-way hash function and bitwise XOR operation only. The rest of this paper is organized as follows. In Section 2, we give a mathematical background on the one-way hash function, which will be helpful for describing and analyzing our scheme. In Section 3, we propose our new robust and secure password-based remote user authentication scheme. In Section 4, we analyze our scheme under different possible attacks using both the informal and formal security analysis. In Section 5, we perform the simulation for the formal security analysis using the widely accepted AVISPA (Automated Validation of Internet Security Protocols and Applications) tool to ensure that our scheme is secure against passive and active attacks. In Section 6, we compare the performance of our scheme with the recently proposed password-based remote user authentication schemes [3, 13–15]. Finally, we conclude the paper in Section 7.

2. Mathematical Preliminaries

In this section, we discuss the properties of one-way hash function for describing and analyzing our scheme. A hash function \( h : \{0, 1\}^* \rightarrow \{0, 1\}^n \) is a one-way function, which takes an arbitrary-length binary string input \( x \in \{0, 1\}^* \) and outputs a fixed-length \( n \)-bit binary string, called the message digest or hash value \( h(x) \in \{0, 1\}^n \). In addition, it has the following important properties [16].

(i) \( h \) can be applied to a data block of all sizes.

(ii) For any given input \( x \), it is relatively easy to compute the hash value \( h(x) \), which enables easy implementation in software and hardware.

(iii) Output length of \( h(x) \) is fixed.

(iv) One-way property: from a given hash value \( y = h(x) \) and the given hash function \( h(\cdot) \), it is computationally infeasible to derive the input \( x \).

(v) Weak-collision resistance property: for any given input \( x \), finding any other input \( y \) with \( y \neq x \), such that \( h(y) = h(x) \) is computationally infeasible.

(vi) Strong-collision resistance property: finding a pair of inputs \( (x, y) \), with \( x \neq y \), such that \( h(x) = h(y) \) is also computationally infeasible.

An example of such a one-way function is SHA-1 [17], which has the above desired properties. At present, the National Institute of Standards and Technology (NIST) does not recommend SHA-1 for top secret documents. In 2011, Manuel [18] showed the collision attacks on SHA-1. Quark [19] is a family of cryptographic hash functions, which is designed for extremely resource-constrained environments like sensor networks and radiofrequency identification tags. Like most one-way hash functions, Quark can be used as a pseudorandom function, a message authentication code, a pseudorandom number generator, a key derivation function, and so forth. Quark performs better than the SHA-1 hash function. Thus, Quark can be used for the one-way function. However, in this paper, as in [14, 20, 21], we can use SHA-2 as the secure one-way hash function in order to achieve top security, whereas we use only 160 bits from the hash digest output of SHA-2 in our scheme and other schemes.
3. The Proposed Scheme

In this section, we propose a new remote user authentication scheme using password, which is based on smart card. For this purpose, we first discuss the threat model used in our scheme. We then discuss the various phases related to our scheme.

3.1. Notations. For describing and analyzing our scheme, we use the notations listed as follows:

- $U_i$: user,
- $S_j$: remote server,
- $ID_i$: identity of user $U_i$,
- $PW_i$: password of user $U_i$,
- $X_i$: permanent secret key only known to the remote server $S_j$,
- $K$: secret number only known to the user $U_i$,
- $T_a$: current system timestamp of an entity $A$,
- $R_a$: random nonce generated by an entity $A$,
- $h(\cdot)$: secure one-way collision-resistant hash function,
- $A \| B$: data $A$ concatenating with data $B$,
- $A \oplus B$: bitwise XOR operation of $A$ and $B$.

3.2. Threat Model. In our scheme, we make use of the Dolev-Yao threat model [22]. In this model, two communicating parties communicate over an insecure channel. Any adversary (attacker or intruder) can thus eavesdrop on the transmitted messages over the public insecure channel and he/she has the ability to modify, delete, or change the contents of the transmitted messages. Usually, the smart card issued to a user is equipped with tamper-resistant device. However, in this paper, we still assume that once a user’s smart card is stolen or lost, the attacker will know all the sensitive information stored in the smart card’s memory by monitoring the power consumption of the smart card [23, 24].

3.3. Motivation. The majority of the proposed password-based remote user authentication schemes are either computationally expensive or vulnerable to different known attacks [7, 8]. Though Sonwanshi et al.’s scheme [3] is very efficient due to usage of one-way hash function and bitwise XOR operations, Das et al. [12] showed that their scheme is vulnerable to the offline password guessing attack and stolen smart card attack. In addition, Das et al. showed that their scheme fails to protect strong replay attack. Lee-Liu’s scheme [13] is expensive in computation as it requires expensive modular exponentiation operations. Further, Das and Bruhadeshwar [14] showed that Lee-Liu’s scheme has security weaknesses. In order to withstand the flaws found in Lee-Liu’s scheme, they proposed an improved and secure password-based remote user authentication scheme. However, Das-Bruhadeshwar’s scheme [14] is also computationally costly as it requires expensive modular exponentiation operations as in Lee-Liu’s scheme [13]. The recently proposed Jiang et al.’s scheme [15] uses ECC cryptosystem and hash function. Due to expensive ECC point addition and scalar multiplication operations, their scheme is also expensive, though their scheme is secure against different attacks. Thus, we feel that there is a great need to propose a new robust and secure password-based remote user authentication scheme which will satisfy the requirements listed in Section I. Our scheme withstands the security flaws found in Sonwanshi et al.’s scheme [3] and it is also very efficient as our scheme relies only on lightweight operations like the one-way hash computations and bitwise XOR operations.

3.4. Different Phases. In this section, we describe the four phases related to our scheme, namely, the registration phase, the login phase, the authentication phase, and the password change phase. In the registration phase, a user $U_i$ needs to register to access services from a remote server $S_j$. After registering, the server $S_j$ will issue a smart card containing important information stored in the smart card’s memory. In the login phase, if the user $U_i$ wants to access services from the server $S_j$, the user $U_i$ needs to login to the system providing his/her identity and password with the help of his/her smart card issued by the registration server. In the authentication phase, the server $S_j$ authenticates the user $U_i$ and the user $U_i$ also authenticates the server $S_j$. After mutual authentication between $U_i$ and $S_j$, both $U_i$ and $S_j$ establish a secret common session key shared between them so that they communicate securely using that established key in future.

3.4.1. Registration Phase. This phase consists of the following steps.

Step R1. The user $U_i$ first selects his/her own secret identity $ID_i$ and chooses a strong (not low-entropy or weak) password $PW_i$.

Step R2. $U_i$ then generates a secret 1024-bit number $K$ randomly, which is kept secret to $U_i$ only.

Step R3. $U_i$ then computes the masked password using $K$, $ID_i$, and $PW_i$ as $RPW_i = h(ID_i \| K \| PW_i)$ and sends the registration request message $\langle ID_i, RPW_i \rangle$ to the registration remote server $S_j$ via a secure channel.

Step R4. After receiving the registration request message in Step R3, the server $S_j$ generates a 1024-bit secret number $X_s$ randomly, which is kept secret to $S_j$ only.

Step R5. $S_j$ then computes $r_i = h(ID_i \| RPW_i) = h(ID_i \| h(ID_i \| K \| PW_i))$ and $e_i = h(ID_i \| X_s) \oplus r_i$. $S_j$ further computes $TD_i = NID_i \oplus h(ID_i \| r_i)$ and $D_i = TD_i$ as in [20]. Here $NID_i$ is a random and temporary identity for the user $U_i$, which is used instead of the permanent identity $ID_i$ to achieve the user anonymity.

Step R6. Finally, $S_j$ issues a smart card $C_i$ containing the information $(r_i, e_i, TD_i, D_i, h(\cdot))$ and sends it to the user $U_i$ via a secure channel.
After receiving the smart card $C_i$ from $S_j$, $U_i$ stores the secret number $K$ into the smart card’s memory. The summary of the registration phase is given in Table 1.

3.4.2. Login Phase. In this phase, the following steps are executed.

**Step L1.** $U_i$ first inserts his/her smart card $C_i$ into a card reader of the specific terminal. $U_i$ then inputs his/her identity $ID_i^1$ and password $PW_i^*$.

**Step L2.** $C_i$ computes the masked password $RPW_i^*$ as $RPW_i^* = h(ID_i^1 \parallel K \parallel PW_i^*)$ using the secret number $K$ stored in its memory. $C_i$ then computes $r_i^* = h(ID_i^1 \parallel RPW_i^*)$ and checks if the condition $r_i^* = r_i$ holds. If this condition holds, $U_i$ passes password verification and the next step is executed. Otherwise, this phase terminates immediately.

**Step L3.** $C_i$ computes $NID_i^1 = h(ID_i^1 \parallel r_i^*) \oplus D_i$ and $M_1 = c_i \oplus r_i^* = h(ID_i^1 \parallel X_i) \oplus r_i \oplus r_i^* = h(ID_i^1 \parallel X_i)$. $C_i$ generates a 160-bit random nonce $R_i$ and then computes $M_2 = M_1 \oplus R_i \oplus T_{\Delta_i} = h(ID_i^1 \parallel X_i) \oplus R_i \oplus T_{\Delta_i}$, where $T_{\Delta_i}$ is the current system timestamp, and $M_3 = h(ID_i^1 \parallel R_i \parallel T_{\Delta_i})$. $C_i$ sends the login request message $(NID_i^1, M_2, M_3, T_{\Delta_i})$ to the server $S_j$ via a public channel.

The summary of the login phase is given in Table 2.

3.4.3. Authentication Phase. After receiving the login request message $(NID_i^1, M_2, M_3, T_{\Delta_i})$ from the user $U_i$, the server $S_j$ checks the format of $NID_i^1$ and then finds the entry $(ID_i^1, NID_i^1)$ in its maintained $ID$ database table. If it is found, $S_j$ performs Case 1; otherwise, $S_j$ proceeds to Case 2.

**Case 1.** Consider the following.

**Step A1.** $S_j$ checks the validity of the timestamp $T_{\Delta_i}$ in the received message by the condition $|T_{\Delta_i} - T_{\Delta_j}| < \Delta T$, where $T_{\Delta_j}$ is the current system timestamp of $S_j$ and $\Delta T$ the expected transmission delay. If this condition is satisfied, $S_j$ computes $M_4 = h(ID_i^1 \parallel X_i)$, using its own secret number $X_j$. After that $S_j$ computes

\[ M_5 = M_2 \oplus M_4 \oplus T_{\Delta_i} = h(ID_i^1 \parallel X_i) \oplus R_i \oplus T_{\Delta_i} = h(ID_i^1 \parallel X_i) \oplus T_{\Delta_i} = R_i, \]

\[ M_6 = h(ID_i^1 \parallel M_5 \parallel T_{\Delta_i}). \]

$S_j$ then verifies the condition whether $M_6 = M_5$ holds. If it does not hold, $S_j$ rejects the login request message and this phase terminates immediately.

In order to protect the man-in-the-middle attacks and the replay attacks, we can adopt the same strategy as in [4, 20].

The server $S_j$ stores the pair $(ID_i, M_2)$, where $M_2 = R_i$, in its database. Suppose the server receives the next login request message $(NID_i, M_2', M_3', T_{\Delta_i}')$ from the user $U_i$ or an attacker $S_j$ first checks the validity of the timestamp $T_{\Delta_i}'$ and if it is valid, it further computes $M_7 = h(ID_i^1 \parallel X_i)$, using its own secret number $X_j$. After that, $S_j$ computes, say, $M_8 = M_4 \oplus M_7 \oplus T_{\Delta_i}' = h(ID_i^1 \parallel X_i) \oplus R_i \oplus T_{\Delta_i}' = h(ID_i^1 \parallel X_i) \oplus T_{\Delta_i}' = R_i$. If $M_8 = M_5$, it ensures that the login request message is a replay one. Otherwise, $S_j$ updates $M_5$ with $M_8$ in its database. Thus, it is noted that the timestamp and random nonces are used together to defend the replay and man-in-the-middle attacks.

**Step A2.** $S_j$ generates a random nonce $R_i$ and then computes $M_7 = M_4 \oplus R_i$; $T_{\Delta_i}$ is the current system timestamp of the server $S_j$, $M_8 = h(R_i \parallel T_{\Delta_i} \parallel M_5 \parallel T_{\Delta_i}) \oplus NID_j^{new}$, where $NID_j^{new}$ is a random and temporary identity generated by $S_j$, and $M_9 = h(ID_i^1 \parallel M_5 \parallel T_{\Delta_i} \parallel R_i \parallel T_{\Delta_i} \parallel NID_j^{new})$. $S_j$ then sends the authentication request message $(M_7, M_8, M_9, T_{\Delta_i})$ to the user $U_i$ via a public channel.

**Step A3.** After receiving the message in Step A2, $C_i$ checks the validity of the timestamp $T_{\Delta_i}$ in the received message with the condition $|T_{\Delta_i} - T_{\Delta_j}| < \Delta T$, where $T_{\Delta_j}$ is the current system timestamp of $C_i$ and $\Delta T$ the expected transmission delay. If this condition does not hold, the phase terminates immediately. Otherwise, $C_i$ computes

\[ M_{10} = M_7 \oplus M_4 \oplus T_{\Delta_i} = h(ID_i^1 \parallel X_i) \oplus R_i \oplus T_{\Delta_i} \oplus h(ID_i^1 \parallel X_i) \oplus T_{\Delta_i} \]

\[ = R_i, \]

\[ M_{11} = h(M_{10} \parallel T_{\Delta_i} \parallel R_i \parallel T_{\Delta_i}), \]

\[ NID_j^{new} = M_8 \oplus M_9. \]  

(2)

$C_i$ further computes $M_{12} = h(ID_i^1 \parallel R_i \oplus 1 \parallel T_{\Delta_i} \oplus 1 \parallel M_{10} \parallel T_{\Delta_i} \parallel NID_j^{new})$ and checks the condition $M_{12} = M_9$. If it does not hold, this phase terminates immediately. Otherwise, on the other hand, $C_i$ updates $T_{\Delta_i}$ and $D_i$ with $D_i$ and $D_i \oplus NID_j^{new} \parallel NID_j^{new}$, respectively, in its memory.

**Step A4.** $C_i$ computes $M_{13} = h(M_{10} + 1 \parallel T_{\Delta_i} \parallel R_i \oplus 1 \parallel T_{\Delta_i} \parallel NID_j^{new} \parallel ID_j)$ and sends the authentication acknowledgment message $(M_{13})$ to the server $S_j$ via a public channel. $C_i$ also computes a secret session key shared between $U_j$ and $S_j$ as $SK_{U_j,S_j} = h(ID_i^1 \parallel R_i \parallel T_{\Delta_i} \parallel M_10 \parallel T_{\Delta_i} \parallel M_4)$.

**Step A5.** After receiving the authentication acknowledgment message $(M_{13})$ from the user $U_i$ in Step A4, $S_j$ computes $M_{14} = h(R_i \oplus 1 \parallel T_{\Delta_i} \parallel M_5 \parallel T_{\Delta_i} \parallel NID_j^{new} \parallel ID_j)$ and verifies whether the condition $M_{14} = M_{13}$ holds. If it holds, $S_j$ authenticates the user $U_j$ and also computes the same secret session key shared with $U_i$ as $SK_{U_i,S_j} = h(ID_i^1 \parallel M_5 \parallel T_{\Delta_i} \parallel R_i \parallel T_{\Delta_i} \parallel M_4)$. Thus, after successful authentication, both $U_i$ and $S_j$ can communicate securely using the established secret session key.
Table 1: Summary of the registration phase of our scheme.

<table>
<thead>
<tr>
<th>User (𝑈𝑖)</th>
<th>Remote server (𝑆𝑗)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selects ID𝑖, PW𝑖.</td>
<td>Generates secret number 𝐾.</td>
</tr>
<tr>
<td>Generates secret number 𝐾.</td>
<td>Computes RP𝑊𝑖 = h(ID𝑖</td>
</tr>
<tr>
<td>Computes RP𝑊𝑖 = h(ID𝑖</td>
<td></td>
</tr>
<tr>
<td>(via a secure channel)</td>
<td>(via a secure channel)</td>
</tr>
</tbody>
</table>

Stores 𝐾 into the smart card’s memory.

Table 2: Summary of the login phase of our scheme.

<table>
<thead>
<tr>
<th>User (𝑈𝑖)/smart card (𝐶𝑖)</th>
<th>Remote server (𝑆𝑗)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs ID𝑖, PW𝑖.</td>
<td>Computes 𝐶𝑖 = h(ID𝑖</td>
</tr>
<tr>
<td>Computes RP𝑊𝑖 = h(ID𝑖</td>
<td></td>
</tr>
<tr>
<td>Checks if 𝑟𝑖 = 𝑟𝑖. If it holds</td>
<td>⟨ID𝑖,RP𝑊𝑖⟩</td>
</tr>
<tr>
<td>computes NID𝑖 = h(ID𝑖</td>
<td></td>
</tr>
<tr>
<td>and M𝑖 = h(ID𝑖</td>
<td></td>
</tr>
<tr>
<td>⟨NID𝑖,𝑀𝑖,𝑟𝑖⟩</td>
<td>(via a public channel)</td>
</tr>
</tbody>
</table>

Case 2. This case remains almost the same as Case 1 except the following in Step A6.

Step A6. NID𝑖 is obtained by computing h(ID𝑖 || 𝑟𝑖) ⊕ 𝑇𝐷𝑖 instead of h(ID𝑖 || 𝑟𝑖) ⊕ 𝐷𝑖 in Step L3 of the login phase. The smart card 𝐶𝑖 of the user 𝑈𝑖 in this case only needs to update 𝐷𝑖 with 𝐷𝑖 ⊕ NID𝑖 ⊕ NID𝑖new without changing 𝑇𝐷𝑖 in Step A3.

The summary of the authentication phase is given in Table 3.

3.4.4. Password Change Phase. To enhance security, a user 𝑈𝑖 needs to change his/her password. Let 𝑈𝑖 want to change his/her password PW𝑖 with a new password PW𝑖new. For this phase, the following steps are executed by the smart card 𝐶𝑖 of the user 𝑈𝑖 without contacting the remote server 𝑆𝑖.

Step P1. 𝑈𝑖 first inserts his/her smart card 𝐶𝑖 into a card reader of the specific terminal and then inputs identity ID𝑖 and provides old password PW𝑖old.

Step P2. 𝐶𝑖 then computes masked password RP𝑊𝑖old = h(ID𝑖 || 𝐾 || PW𝑖old) using the secret number 𝐾 stored in its memory and 𝑘𝑖old = h(ID𝑖 || RP𝑊𝑖old). 𝐶𝑖 checks if the condition 𝑘𝑖old = 𝑟𝑖 holds. If it does not hold, the old password verification fails and this phase terminates immediately.

Otherwise, 𝐶𝑖 asks the user 𝑈𝑖 to input his/her chosen strong (high-entropy) password PW𝑖new, where PW𝑖old ≠ PW𝑖new.

Step P3. 𝐶𝑖 computes

\[
\begin{align*}
    x & = 𝑒𝑖 ⊕ 𝑟𝑖old \\
    & = h(ID𝑖 || X𝑖) ⊕ 𝑟𝑖 ⊕ 𝑟𝑖old \\
    & = h(ID𝑖 || X𝑖), \quad \text{since } 𝑟𝑖old = 𝑟𝑖, \\
    𝑓𝑖new & = h(ID𝑖 || RP𝑊𝑖new) \\
    𝑟𝑖new & = h(ID𝑖 || RP𝑊𝑖new), \\
    𝑒𝑖new & = x ⊕ 𝑟𝑖new.
\end{align*}
\]

C𝑖 further computes 𝑇𝐷𝑖new = 𝑇𝐷𝑖 ⊕ h(ID𝑖 || 𝑟𝑖old) ⊕ h(ID𝑖 || 𝑟𝑖new) = NID𝑖 ⊕ h(ID𝑖 || 𝑟𝑖old || 𝑟𝑖new) and 𝑇𝐷𝑖new = 𝑇𝐷𝑖new.

Step P4. Finally, 𝐶𝑖 updates 𝑟𝑖 with 𝑟𝑖new, 𝑒𝑖 with 𝑒𝑖new, 𝑇𝐷𝑖 with 𝑇𝐷𝑖new, and 𝐷𝑖 with 𝐷𝑖new in its memory.

Thus, it is clear that our scheme provides efficient password change phase in order to change the password of a user 𝑈𝑖 at any time locally and correctly without further contacting the remote server 𝑆𝑖.

4. Security Analysis of the Proposed Scheme

In this section, we first show the correctness of our proposed scheme. We then provide informal and formal security analysis to show that our scheme is secure against various known attacks.

4.1. Correctness. In Theorem I, we provide the correctness of our scheme.

Theorem 1. The proposed scheme always establishes the correct secret session key between the user 𝑈𝑖 and the server 𝑆𝑖 during the authentication phase after the successful mutual authentication between them.
Table 3: Summary of the authentication phase of our scheme.

<table>
<thead>
<tr>
<th>User (𝑈𝑖)/smart card (𝐶𝑖)</th>
<th>Remote server (𝑆𝑗)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks the validity of 𝑇𝑖.</td>
<td>Checks the validity of 𝑇𝑖.</td>
</tr>
<tr>
<td>If it holds, computes</td>
<td>If it holds, computes</td>
</tr>
<tr>
<td>𝑀4 = ℎ(𝐼𝐷𝑖 ⊕ 𝑋𝑖),</td>
<td>𝑀4 = ℎ(𝐼𝐷𝑖 ⊕ 𝑋𝑖),</td>
</tr>
<tr>
<td>𝑀5 = ℎ(𝐼𝐷𝑖 @ 𝑇𝑖),</td>
<td>𝑀5 = ℎ(𝐼𝐷𝑖 @ 𝑇𝑖),</td>
</tr>
<tr>
<td>and 𝑀6 = ℎ(𝐼𝐷𝑖 ⊕ 𝑇𝑖).</td>
<td>and 𝑀6 = ℎ(𝐼𝐷𝑖 ⊕ 𝑇𝑖).</td>
</tr>
<tr>
<td>Checks if 𝑀6 = ℎ(𝐼𝐷𝑖) + 1 ⊕ 𝑇𝑖 + 1</td>
<td></td>
</tr>
</tbody>
</table>

Steps A4 and A5, after the successful mutual authentication between them. Note that, in Step A4, 𝐶𝑖 computes the secret session key shared with 𝑈𝑖 as 𝐾𝑈𝑖,𝐶𝑖 = ℎ(𝐼𝐷𝑖 || 𝑆𝐾𝑖). As a result, 𝐾𝑈𝑖,𝐶𝑖 = ℎ(𝐼𝐷𝑖 || 𝑆𝐾𝑖). Hence, the theorem follows.

4.2. Informal Security Analysis. In this section, through the informal security analysis we show that our scheme has the ability to defend the various known attacks, which are discussed in the following subsections.

4.2.1. Replay Attack. Suppose an attacker intercepts the login request message ⟨𝑁𝐼𝐷𝑖, 𝑀2, 𝑀3, 𝑇𝑖⟩ during the login phase, where 𝑀2 = ℎ(𝐼𝐷𝑖 || 𝑋𝑖) ⊕ 𝑆𝐾𝑖 and 𝑀3 = ℎ(𝐼𝐷𝑖 || 𝑇𝑖), and starts a new session with the message ⟨𝑁𝐼𝐷𝑖, 𝑀2, 𝑀3, 𝑇𝑖⟩ = ⟨𝑁𝐼𝐷𝑖, 𝑀2, 𝑀3, 𝑇𝑖⟩. According to our policy, the server 𝑆𝑗 stores the pair ⟨𝐼𝐷𝑖, 𝑀5⟩, where 𝑀5 = ℎ(𝐼𝐷𝑖 ⊕ 𝑋𝑖), in its database. 𝑆𝑗 first checks the validity of the timestamp 𝑇𝑖, and if it is valid, it further computes 𝑀5 = ℎ(𝐼𝐷𝑖 ⊕ 𝑋𝑖) using its own secret number 𝑋𝑖. After that 𝑆𝑗 computes, say, 𝑀5 = 𝑀1 || 𝑆𝐾𝑖 || ℎ(𝐼𝐷𝑖 ⊕ 𝑋𝑖) || 𝑆𝐾𝑖 || ℎ(𝐼𝐷𝑖 ⊕ 𝑋𝑖). If 𝑀5 = 𝑀5, it ensures that the login request message is a replay one. Since the transmission delay time is short, even if the attacker replays the same login request message during that time, our scheme prevents this as a replay message due to verification of random nonce attached to the message with that in the stored database. As a result, both the timestamp and random nonce together help to defend strongly the replay attack in our scheme.

4.2.2. Man-in-the-Middle Attack. Suppose an attacker intercepts the login request message ⟨𝑁𝐼𝐷𝑖, 𝑀2, 𝑀3, 𝑇𝑖⟩ during the login phase, where 𝑀2 = ℎ(𝐼𝐷𝑖 ⊕ 𝑋𝑖) ⊕ 𝑆𝐾𝑖. In order to make success in the man-in-the-middle attack, the attacker has to change 𝑀2 and 𝑀3 properly so that the server 𝑆𝑗 can authenticate the message successfully. Assume that the attacker uses a timestamp 𝑇′.
and wants to change $M_2$ and $M_3$ to $M' = M_2 \oplus T_c \oplus T'_c = h(ID_i \parallel X_i) \oplus R_c \oplus T'_c$ and $M' = h(ID'_i \parallel R_c \parallel T'_c)$, respectively. However, for $M'_i$, the attacker needs to know both $ID'_i$ and $R_c$ which are unknown to that attacker. As pointed out in [20], the probability of guessing an identity composed of exact $n$ characters is approximately $1/2^{6n}$. Thus, to correctly know $ID'_i$ and $R_c$ from $M'_i$, the attacker has to guess both $ID'_i$ and $R_c$ at the same time using $T_c$ and the probability of guessing both $ID'_i$ composed of exact $n$ characters and $R_c$ composed of $m$ bits ($m = 160$ bits in our scheme) at the same time becomes approximately $1/2^{6n-m}$. If $n = 10$, then this probability is approximately $1/2^{60-160} = 1/2^{220}$, which is very negligible. As a result, the attacker does not have any ability to succeed in this attack and, hence, our scheme is secure against the man-in-the-middle attack.

4.2.3. Impersonation Attack. In this attack, the purpose of an attacker is to impersonate the remote server $S_j$ or a legal user $U_j$ in order to cheat the other party. Suppose an attacker intercepts the login request message $\langle NID_i^j, M_2, M_3, T_j \rangle$ during the login phase and wants to start a new session. In order to start a new session, the attacker has to modify both $M_2$ and $M_3$. However, as discussed in Section 4.2.2, to change $M'_2$ the attacker has to guess/know both $ID'_2$ and $R_c$, which are unknown to the attacker. Thus, the probability of guessing both $ID'_2$ composed of exact $n$ characters and $R_c$ composed of $m$ bits ($m = 160$ bits in our scheme) at the same time becomes approximately $1/2^{6n-m} = 1/2^{60+160}$, which is also very negligible. Hence, our scheme prevents the impersonation attack.

4.2.4. Stolen Smart Card Attack. In this attack, we assume that the smart card $C_i$ of a legal user $U_i$ is lost or stolen by an attacker. Then the attacker can extract all the secret information $(r_i, e_i, T_D^i, D_i, K)$ from the memory of the stolen or lost smart card $C_i$ of the user $U_i$ using the power analysis attacks [23, 24]. Note that $r_i = h(ID_i \parallel RPW_i) = h(ID_i \parallel h(ID_i \parallel K \parallel PW_i))$ and $e_i = h(ID_i \parallel X_i) \oplus r_i$. The attacker can derive $h(ID_i \parallel X_i) = e_i \oplus r_i$. In order to know the secret information $X_i$ of the server $S_i$, the attacker needs to guess both $ID_i$ and $X_i$. The probability of guessing both $ID_i$ composed of exact $n$ characters and $X_i$ composed of $m$ bits ($m = 1024$ bits in our scheme) at the same time becomes approximately $1/2^{6n+1024}$, which is very negligible. Again, to derive the password $PW_i$ composed of $l$ characters, the attacker needs to also guess $ID_i$ using $K$. Thus, the probability of guessing both $ID_i$ composed of exact $n$ characters and $PW_i$ composed of exact $l$ characters at the same time becomes approximately $1/2^{6n+6l}$, which is also negligible. Hence, our scheme prevents the stolen smart card attack.

4.2.5. Password Guessing Attack. In this attack, we consider both offline and online password guessing attacks. As in Section 4.2.4, we assume that the smart card $C_i$ of a legal user $U_i$ is lost or stolen by an attacker and all the secret information $(r_i, e_i, T_D^i, D_i, K)$ stored in the memory of the smart card $C_i$ is known to the attacker. Still then the attacker cannot guess correctly the password $PW_i$ of $U_i$ offline, which is evident from Section 4.2.4.

Suppose the attacker intercepts all the transmitted messages $\langle NID_i^j, M_2, M_3, T_j \rangle$ during the login phase and $\langle M_7, M_8, M_9, T_j \rangle$ and $\langle M_13 \rangle$ during the authentication phase. However, none of these messages involves the password $PW_i$ of the user $U_i$. As a result, these messages will not be helpful to the attacker to obtain $PW_i$ of $U_i$ online. Thus, our scheme is secure against both offline and online password guessing attacks.

4.2.6. Denial-of-Service Attack. Note that, in our scheme, the smart card $C_i$ of a legal user $U_i$ stores $T_D$ and $D_i$ for the previous and the latest random identities, respectively. Thus, the corruption of the message $\langle M_13 \rangle$ during the authentication phase is not possible by an attacker and, hence, our scheme prevents the denial-of-service attack.

4.2.7. User Anonymity. In our scheme, all the transmitted messages include the identity $ID_i$ of a legal user $U_i$ indirectly and it is protected by the one-way secure hash function $h()$. Due to the collision-resistant property of $h()$, it is computationally infeasible for an attacker to derive $ID_i$.

Even if we assume that the smart card $C_i$ of a legal user $U_i$ is lost or stolen by an attacker and all the secret information $(r_i, e_i, T_D^i, D_i, K)$ stored in the memory of the smart card $C_i$ is known to the attacker, from $T_D^i$ and $NID_i^j$ from the intercepted login request message $\langle NID_i^j, M_2, M_3, T_j \rangle$ the attacker can compute $h(ID_i || r_i) = T_D^i \oplus NID_i^j$. Again, $ID_i$ is protected by the one-way secure hash function $h()$. Due to the collision-resistant property of $h()$, it is computationally infeasible for an attacker to derive $ID_i$. Hence, our scheme preserves the user anonymity property.

4.2.8. Mutual Authentication. During the authentication phase, after receiving the authentication request message $\langle M_7, M_8, M_9, T_j \rangle$ from the server $S_j$, the smart card $C_i$ of a legal user $U_i$ computes $M_{12} = h(ID_i \parallel R_c \parallel T_c \parallel T_9 \parallel M_9 \parallel T_i \parallel NID_i^{\text{new}} \parallel ID_j)$ and checks the condition $M_{12} = M_{93}$. If it holds, $U_i$ authenticates the server $S_j$ and then only sends the authentication acknowledgment message $\langle M_{13} \rangle$ to the server $S_j$. After that the server $S_j$ also computes $M_{14} = h(R_c \parallel T_c \parallel M_9 \parallel T_i \parallel M_{12} \parallel NID_i^{\text{new}} \parallel ID_j)$ and verifies whether the condition $M_{14} = M_{93}$ holds. If it holds, $S_j$ authenticates the user $U_i$. Hence, the mutual authentication is always performed in our scheme.

4.2.9. Session Key Security. After mutual authentication, the smart card $C_i$ of a legal user $U_i$ computes the secret session key shared between $U_i$ and $S_j$ as $K_{U_i,S_j} = h(ID_i \parallel R_c \parallel T_c \parallel M_9 \parallel T_j \parallel M_{10} \parallel M_4)$. The server $S_j$ also computes the secret session key shared with the user $U_i$ as $K_{S_j,U_i} = h(ID_i \parallel M_4 \parallel T_9 \parallel R_c \parallel T_i \parallel M_9)$, where $M_4 = h(ID_i \parallel X_i)$ and $M_9 = h(ID_i \parallel X_i)$. It is also evident from Theorem 1 that $K_{U_i,S_j} = SK_{S_j,U_i}$. In order to compute the secret key $SK_{S_j,U_i}$ from all the transmitted messages during the login and authentication phases, an attacker has to guess/derive
correctly composed of exact \( n \) characters, \( X_s \), of \( m = 1024 \) bits, and \( R_c \) and \( R_s \), each composed of 160 bits at the same time, and, thus, the probability of deriving this secret key is approximately \( 1/2^{6m+n+160+160} = 1/2^{6m+1344} \), which is very negligible. As a result, our scheme also provides the session key security.

4.3. Formal Security Analysis. For the formal security analysis, we follow the formal definition of a one-way hash function \( h(\cdot) \) given in Definition 2.

**Definition 2** (one-way hash function \([25, 26]\)). A one-way collision-resistant hash function \( h : \{0, 1\}^* \rightarrow \{0, 1\}^n \) is a deterministic function that takes the input as an arbitrary-length binary string \( x \in \{0, 1\}^* \) and outputs a binary string \( y = h(x) \in \{0, 1\}^n \) of fixed length \( n \). We formalize an adversary \( \mathcal{A} \)'s advantage in finding collision in the following manner:

\[
\text{Adv}^\text{HASH}_\mathcal{A}(t) = \Pr\left[ (x, x') \iff \mathcal{A} : x \neq x' \quad h(x) = h(x') \right],
\]

where \( \Pr[E] \) denotes the probability of an event \( E \) and \( (x, x') \iff \mathcal{A} \) denotes that the pair \( (x, x') \) is selected randomly by \( \mathcal{A} \). The adversary \( \mathcal{A} \) is allowed to be probabilistic and the probability in the advantage is computed over the random choices made by the adversary \( \mathcal{A} \) with the execution time \( t \). The hash function \( h(\cdot) \) is called collision resistant, if \( \text{Adv}^\text{HASH}_\mathcal{A}(t) \leq \epsilon \), for any sufficiently small \( \epsilon > 0 \).

We then define the following random oracle for our formal security analysis.

(i) **Reveal.** This random oracle will unconditionally output the input \( x \) from the corresponding hash value \( y = h(x) \).

In Theorems 3 and 4, we show that our scheme is secure against an adversary for deriving the secret number \( X_s \) of the server and the password \( PW_i \) of a user \( U_i \).

**Theorem 3.** Under the assumption that a one-way hash function \( h(\cdot) \) closely behaves like a random oracle, the proposed scheme is provably secure against an adversary for deriving the secret number \( X_s \) of the server \( S_j \).

**Proof.** We follow the same proof presented in \([14, 27, 28]\). In this proof, we construct an adversary \( \mathcal{A} \) such that he/she can derive the secret number \( X_s \) of the server \( S_j \) correctly. For this purpose, the adversary \( \mathcal{A} \) runs the experiments, \( \text{EXP}^\text{HASH}_\mathcal{A} \), for our robust and effective smart-card-based remote user authentication scheme, say, \( \text{REUS} \) given in Algorithm 1.

We now define the success probability for \( \text{EXP}^\text{HASH}_\mathcal{A} \) as \( \text{succ}^\text{HASH}_\mathcal{A} = \Pr[\text{EXP}^\text{HASH}_\mathcal{A} = 1] - 1 \). Then the advantage of \( \text{EXP}^\text{HASH}_\mathcal{A} \) becomes \( \text{Adv}^\text{HASH}_\mathcal{A} = \max_{\mathcal{A}}[\text{Succ}^\text{HASH}_\mathcal{A}(t_1, q_R) = \max_{\mathcal{A}}[\text{Succ}^\text{HASH}_\mathcal{A}(t_1, q_R) \iff \mathcal{A} : t_1 \quad \text{and the number of queries} q_R \text{ made to the Reveal oracle. We call that our scheme is provably secure against the adversary} \mathcal{A} \text{ for deriving the secret number} X_s \text{ of the server} S_j, \text{if} \text{Adv}^\text{HASH}_\mathcal{A}(t_1, q_R) \leq \epsilon, \text{for any sufficiently small} \epsilon > 0 \).

Consider the experiment provided in Algorithm 1. According to this experiment, if the adversary \( \mathcal{A} \) has the ability to invert the hash function \( h(\cdot) \), then only he/she can derive the secret number \( X_s \) of the server \( S_j \) and win the game. However, according to Definition 2, it is a computationally infeasible (hard) problem for inverting a one-way hash function \( h(\cdot) \). Since \( \text{Adv}^\text{HASH}_\mathcal{A}(t_1, q_R) \leq \epsilon \), for any sufficiently small \( \epsilon > 0 \), we have \( \text{Adv}^\text{HASH}_\mathcal{A}(t_1, q_R) \leq \epsilon \), as it is dependent on the former. As a result, the adversary \( \mathcal{A} \) does not have any ability to derive the secret number \( X_s \) of the server \( S_j \).

**Theorem 4.** Under the assumption that a one-way hash function \( h(\cdot) \) closely behaves like a random oracle, the proposed scheme is provably secure against an adversary for deriving the password \( PW_i \) of a user \( U_i \), even if the smart card \( C_i \) of \( U_i \) is lost or stolen by that adversary.

**Proof.** We need to construct an adversary \( \mathcal{A} \) such that he/she can derive the password \( PW_i \) of the user \( U_i \) correctly after extracting the information stored in the stolen or lost smart card \( C_i \) of \( U_i \). For this purpose, the adversary \( \mathcal{A} \) runs the experiment, \( \text{EXP}^\text{HASH}_\mathcal{A} \), which is provided in Algorithm 2.

Similar to the experiment \( \text{EXP}^\text{HASH}_\mathcal{A} \) given in Algorithm 1, we also define the success probability for \( \text{EXP}^\text{HASH}_\mathcal{A} \) as \( \text{Succ}^\text{HASH}_\mathcal{A} = \Pr[\text{EXP}^\text{HASH}_\mathcal{A} = 1] - 1 \), and the advantage of \( \text{EXP}^\text{HASH}_\mathcal{A} \) as \( \text{Adv}^\text{HASH}_\mathcal{A}(t_2, q_R) = \max_{\mathcal{A}}[\text{Succ}^\text{HASH}_\mathcal{A}(t_2, q_R) \iff \mathcal{A} : t_2 \quad \text{and the number of queries} q_R \text{ made to the Reveal oracle. Our scheme is then provably secure against the adversary} \mathcal{A} \text{ for deriving the password} PW_i \text{ of the user} U_i, \text{if} \text{Adv}^\text{HASH}_\mathcal{A}(t_2, q_R) \leq \epsilon, \text{for any sufficiently small} \epsilon > 0 \).

Now, consider the experiment provided in Algorithm 2. After extracting all the secret information \((r_i, e_i, TD_i, D_i, K)\) from the memory of the stolen or lost smart card \( C_i \) of the user \( U_i \), the adversary \( \mathcal{A} \) can derive the password \( PW_i \) of the user \( U_i \) and win the game, if he/she has the ability to invert the one-way hash function \( h(\cdot) \). Since inverting the one-way hash function \( h(\cdot) \) is computationally infeasible, that is, \( \text{Adv}^\text{HASH}_\mathcal{A}(t_2, q_R) \leq \epsilon \), for any sufficiently small \( \epsilon > 0 \), we have \( \text{Adv}^\text{HASH}_\mathcal{A}(t_2, q_R) \leq \epsilon \), as it is dependent on the former. Hence, our scheme is provably secure against an adversary for deriving the password \( PW_i \) of a user \( U_i \), even if the smart card \( C_i \) of \( U_i \) is lost or stolen by that adversary.

5. Formal Security Verification Using AVISPA Tool

In this section, through the simulation results for the formal security verification using the widely accepted AVISPA tool \([20, 21, 27, 28]\) we show that our scheme is secure against passive and active attacks.
AVISPA (Automated Validation of Internet Security Protocols and Applications) is considered as a push-button tool for the automated validation of Internet security-sensitive protocols and applications [29]. AVISPA has four different back-ends that implement a variety of state-of-the-art automatic analysis techniques. The back-ends are the On-the-Fly Model-Checker (OFMC), Constraint Logic based Attack Searcher (CL-AtSe), SAT-based Model-Checker (SATMC), and Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP). The protocols to be analyzed under the AVISPA tool require specifying them in a language, called HLPSL (High Level Protocols Specification Language), which is a role-oriented language. The specification in HLPSL is first translated into a low-level specification by a translator, which is called the hlpsl2if. hlpsl2if generates a specification in an intermediate format, which is known as the intermediate format (IF). The output format (OF) of AVISPA is generated using one of the four back-ends: OFMC, CL-AtSe, STAMC, and TA4SP. The analysis of the OF is made as follows. The first printed section, called SUMMARY, indicates whether the protocol is safe or unsafe or whether the analysis is inconclusive. DETAILS is the second section, which explains under what condition the protocol is declared safe, what conditions have been used for finding an attack, or finally why the analysis was inconclusive. The remaining sections, called PROTOCOL, GOAL, and BACKEND, represent the name of the protocol, the goal of the analysis, and the name of the back-end used, respectively. Finally, at the end of the analysis, after some possible comments and the statistics, the trace of the attack (if any) is also printed in the usual Alice-Bob format. One can find more details on HLPSL in [29].

5.1. Specifying Our Scheme. We have implemented our scheme for the formal security verification for the registration phase, the login phase, and the authentication phase using the HLPSL language. We have two basic roles: one for Alice, which represents the participant as the user $U_i$, and another for Bob, which represents the remote server $S_j$. The role of the initiator, the user $U_i$, is shown in Algorithm 3. In this role, $U_i$ first receives the start signal, changes its state value from 0 to 1, and then sends the registration request message $\langle ID_i, RPW_i \rangle$ securely to the server $S_j$ using the symmetric key $SK_{uisj}$ shared between $U_i$ and $S_j$ via the $Snd( )$ operation. During the registration phase, the user $U_i$ then receives a smart card containing the information $(r_i, e_i, TD_i, D_i, h(\cdot))$ securely from $S_j$ by the $Rcv( )$ operation. The type declaration channel $(dy)$ in HLPSL specification declares that the channel is for the Dolev-Yao threat model [1]. In this role, agent represents a principal name. The intruder is always assumed to have the special identifier $i$. symmetric_key represents a key for a symmetric-key cryptosystem. text is often used as nonce. This value can be also used for messages. nat type represents the natural numbers in nonmessage contexts, whereas const represents a constant. hash_func represents cryptographic hash functions. function also represents functions on the
Algorithm 3: Continued.

\[
\text{role alice } (U_i, S_j); \text{ agent,}
\]
\[
SK_{u,s}: \text{ symmetric_key,}
\]
\[
\% H \text{ is hash function}
\]
\[
H: \text{ hash_func,}
\]
\[
\text{Snd, Rcv: channel(dy)}
\]
\[
\% U_i \text{ is the user; } S_j \text{ is the server}
\]
played by \( U_i \)
def =

local State: nat,
\[
ID_j, NID_j, PW_j, RPW_j, R_j; \text{ text,}
\]
\[
% K \text{ is a secret number to } U_i
\]
\[
% X_j \text{ is a secret number to } S_j
\]
\[
T_j, R_j, T'_j, K, X_j; \text{ text,}
\]
\[
NID_i, NID_j; \text{ text,}
\]
\[
\text{ADD: hash_func,}
\]
\[
M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13}, M_{14}; \text{ text}
\]
\[
\text{const alice\_bob\_tc, bob\_alice\_ts,}
\]
\[
alice\_bob\_rc, bob\_alice\_rs,
\]
\[
\text{subs1, subs2: protocol\_id}
\]

init State := 0

transition

\% Registration phase

(1) State = 0 \land \text{Rcv(start)} = \rightarrow

\[
\text{State'} := 1 \land \text{RPW}_j := H(ID_j \cdot K \cdot PW_j)
\]
\[
\% \text{Send the registration request message}
\]
\[
\land \text{Snd}([ID_j \cdot \text{RPW}_j], SK_{u,s})
\]
\[
\% \text{Keep } X_j \text{ secret to } S_j \text{ and } PW_j, K \text{ to } U_i
\]
\[
\land \text{secret}([X_j], \text{subs1}, S_j)
\]
\[
\land \text{secret}([PW_j, K], \text{subs2}, U_i)
\]
\[
\% \text{Receive the smart card from the registration server } S_j
\]

(2) State = 1 \land \text{Rcv}([H(ID_j \cdot H(ID_j \cdot K \cdot PW_j))].
\]
\[
\text{xor}([H(ID_j \cdot X_j), H(ID_j \cdot H(ID_j \cdot K \cdot PW_j))].
\]
\[
\text{xor}([NID_j, H(ID_j \cdot H(ID_j \cdot K \cdot PW_j))].
\]
\[
\text{xor}([NID_j, H(ID_j \cdot H(ID_j \cdot H(ID_j \cdot K \cdot PW_j))]).
\]
\[
H]_{SK_{u,s}} = \rightarrow
\]

\% Login phase

\[
\text{State'} := 2 \land
\]
\[
NID'_j := \text{xor}([H(ID_j \cdot H(ID_j \cdot H(ID_j \cdot K \cdot PW_j))].
\]
\[
\text{xor}([NID_j, H(ID_j \cdot H(ID_j \cdot K \cdot PW_j))].
\]
\[
\land M'_1 := \text{xor}([M_1, R'_1])
\]
\[
\land M'_2 := H(ID_j \cdot R'_1 \cdot T'_j)
\]
\[
\% \text{Generate a random nonce}
\]
\[
\land R'_j := \text{new()}
\]
\[
\% T'_j \text{ is the current system timestamp}
\]
\[
\land T'_j := \text{new()}
\]
\[
\land M'_1 := \text{xor}([M'_1, R'_1])
\]
\[
\land M'_2 := H(ID_j \cdot R'_1 \cdot T'_j)
\]
\[
\% \text{Send the login request message}
\]
\[
\land \text{Snd}([NID'_j, M'_1, M'_2, T'_j])
\]
\[
% U_i \text{ has freshly generated the random nonce } R_j \text{ for } S_j
\]
\[
\land \text{witness}([U_i, S_j, alice\_bob\_rc, R'_j])
\]
\[
% U_i \text{ has freshly generated the timestamp } T'_j \text{ for } S_j
\]
\[
\land \text{witness}([U_i, S_j, alice\_bob\_tc, T'_j])
\]
\%

\% Authentication phase
space of messages. In HLPSL, it is assumed that the intruder cannot invert hash functions (in essence, that they are one way). The space of legal messages is defined as the closure of the basic types. For example, given a message Msg and an encryption key Key, \{Msg\}_Key denotes the symmetric/public-key encryption. The associative \"\&\" operator is used for concatenation. The \"played by A\" declaration tells that the agent named in variable A will play a specific role. A knowledge declaration (generally in the top-level Environment role) is used to specify the intruder’s initial knowledge. Immediate reaction transitions have the form $X = I \rightarrow Y$, which relate an event X and an action Y. This means that whenever we take a transition that is labeled in such a way so as to make the event predicate X true, we must immediately (i.e., simultaneously) execute action Y. If a variable V remains permanently secret, it is expressed by the goal secrecy_of V. Thus, if V is ever obtained or derived by the intruder, a security violation will result.

During the login phase of our scheme, the user $U_i$ sends the login request message $\langle NID_i, M_2, M_3, T_j \rangle$ to the server $S_j$. During the authentication phase, after receiving the authentication request message $(M_2, M_3, M_0, T_j)$ from $S_j$, $U_i$ sends the authentication acknowledgment message $(M_{13})$ to $S_j$. In this role, witness (A, B, id, E) for a strong authentication property of A by B on E declares that agent A requests a check of the value E; this goal will be identified by the constant id in the goal section [29]. This formalizes A’s acceptance of the value E as having been generated for him/her by the agent named in B.

The role of the responder, the server $S_j$, is shown in Algorithm 4. During the registration phase, after receiving the login request message $\langle ID_j, RPW_j \rangle$ securely from the user $U_i$, $S_j$ then issues a smart card and sends it containing the information $\{r_i, e, TD_j, D_j, h(\cdot)\}$ securely to $U_i$. During the login phase, after receiving the login request message $\langle NID_i, M_2, M_3, T_j \rangle$, $S_j$ sends the authentication request message $(M_2, M_3, M_0, T_j)$ securely to $U_i$ in the authentication phase. Finally, $S_j$ waits for the authentication acknowledgment message $(M_{13})$ from $U_i$.

Finally, in Algorithms 5 and 6, we have specified the roles for the session and the goal and environment of our scheme.

**Table 4: Comparison of communication overhead between our scheme and other related schemes during the login and authentication phases.**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Total number of messages required</th>
<th>Total number of bits required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lee and Liu [13]</td>
<td>3</td>
<td>1504</td>
</tr>
<tr>
<td>Das and Bruhadeshwar [14]</td>
<td>3</td>
<td>1664</td>
</tr>
<tr>
<td>Sonwanshi et al. [3]</td>
<td>2</td>
<td>704</td>
</tr>
<tr>
<td>Jiang et al. [15]</td>
<td>3</td>
<td>1944</td>
</tr>
<tr>
<td>Ours</td>
<td>3</td>
<td>1184</td>
</tr>
</tbody>
</table>
role bob \((U_i, S_j): \text{agent},\)

\(SK_{U_iS_j}: \text{symmetric key},\)

\% \(H\) is hash function

\(H: \text{hash func},\)

Snd, Rcv: channel(dy))

\% \(U_i\) is the user; \(S_j\) is the server

played by \(S_j\)

def =

local State: nat,

\(ID_i, NID_i, TD_i, D_i, PW_i, RPW_i, R_i, E_i: \text{text},\)

\% \(K\) is a secret number to \(U_i\)

\% \(X_j\) is a secret number to \(S_j\)

\(T_s, R_s, T_i, K, X_s: \text{text},\)

\(NID_{ij}, NID_{ij}: \text{text},\)

ADD: hash_func,

\(M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8, M_9, M_{10}, M_{11}, M_{12}, M_{13}, M_{14}: \text{text}\)

const alice_bob_tc, bob_alice_is, alice_bob_rc, bob_alice_rs,

subsi, subs2: protocol_id

init State := 0

\% Registration phase

\% Receive the registration request message from the user

(1) State = 0 \& Rcv((\(ID_i, H(ID_i \cdot K \cdot PW_i)\), \(SK_{U_iS_j}\)) =⇒>

\% Keep \(X_j\) secret to \(S_j\) and \(PW_i, K\) to \(U_i\)

\% Authentication phase

\% Login phase

(2) State = 1 \& Rcv(xor(H(ID_i \cdot H(ID_i \cdot H(ID_i \cdot K \cdot PW_i)))),

\(\text{xor}(NID_i, H(ID_i \cdot H(ID_i \cdot K \cdot PW_i)))\),

\(\text{xor}(\text{xor}(H(ID_i \cdot X_s), H(ID_i \cdot K \cdot PW_i)), R_i^{'},)

\(H(ID_i \cdot R_i^{'}, T_i^{'},) =⇒>

% Authentication phase

State' := 2 \&

\% generate a random nonce

\(R_i^{'}, := \text{new}()\)

\% \(T_i^{'}, \text{is the current system timestamp}\)

\(T_i^{'}, := \text{new}()\)

\% \(M_i^{'}, := H(ID_i \cdot X_s)\)

\(M_i^{'}, := \text{xor}(\text{xor}(\text{xor}(H(ID_i \cdot X_s),

\(H(ID_i \cdot H(ID_i \cdot K \cdot PW_i))\)),

\(H(ID_i \cdot H(ID_i \cdot K \cdot PW_i)), R_i^{'},)

\(H(ID_i \cdot X_s), T_i^{'})\)

\% \(M_i^{'}, := H(ID_i \cdot M_i^{'}, T_i^{'})\)

\(M_i^{'}, := \text{xor}(M_i^{'}, R_i^{'}, T_i^{'})\)

\% \(NID_{ij}'. := \text{new}()\)

\(M_2^{'}, := \text{xor}(H(R_i^{'}, T_i^{'}, M_i^{'}, T_i^{'},), NID_{ij}')\)

Algorithm 4: Continued.
\[ M'_i = H(ID_i \cdot \text{ADD}(M'_i \cdot 1) \cdot \text{ADD}(T'_i \cdot 1) \cdot R'_i \cdot \text{NID}'_{P_i}) \]

% Send the authentication request message
\[ \wedge \text{Snd}(M'_i \cdot M'_i \cdot M'_i \cdot T'_i) \]
\% S_i has freshly generated the random nonce R_i for U_j
\[ \wedge \text{witness}(S_i, U_j, \text{bob_alice_rs}, R'_i) \]
\% S_i has freshly generated the timestamp T_i for U_j
\[ \wedge \text{witness}(S_i, U_j, \text{bob_alice_ts}, T'_i) \]
\% Receive the authentication acknowledgement message
(3) State = 3 \& \text{Rcv}(H(\text{ADD}(R'_i \cdot 1) \cdot \text{ADD}(T'_i \cdot 1) \cdot \text{ADD}(R'_i \cdot 1)) \cdot \text{NID}'_{P_i} \cdot (ID_i)) = \text{expr}
\% S_i’s acceptance of the value R_i generated for S_j by U_j
State’ = 3 \& \text{request}(U_i, S_j, \text{alice_bob_rc}, R'_i)
\% S_i’s acceptance of the value T_i generated for S_j by U_i
\[ \wedge \text{request}(U_i, S_j, \text{alice_bob_tc}, T'_i) \]
end role

Algorithm 4: Role specification in HLPSL for the server S_j of our scheme.

Table 5: Comparison of computational overhead between our scheme and other schemes during all phases.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>(2t_h)</td>
<td>(4t_h)</td>
<td>(2t_h)</td>
<td>(7t_{ecm} + 6t_{eca} + 8t_h)</td>
<td>(4t_h)</td>
</tr>
<tr>
<td>Login + authentication</td>
<td>(2t_{me} + 10t_h)</td>
<td>(2t_{me} + 14t_h)</td>
<td>(13t_h)</td>
<td>(10t_{ecm} + 3t_{eca} + 10t_h)</td>
<td>(14t_h)</td>
</tr>
<tr>
<td>Password change</td>
<td>(2t_h)</td>
<td>(5t_h)</td>
<td>(4t_h)</td>
<td>(2t_{ecm} + 2t_{eca} + 8t_h)</td>
<td>(6t_h)</td>
</tr>
<tr>
<td>Total</td>
<td>(2t_{me} + 14t_h)</td>
<td>(2t_{me} + 23t_h)</td>
<td>(19t_h)</td>
<td>(19t_{ecm} + 11t_{eca} + 26t_h)</td>
<td>(24t_h)</td>
</tr>
</tbody>
</table>

Note: \(t_{me}\): the time to compute a one-way hash function; \(t_{eca}\): the time to compute a modular exponentiation; \(t_{ecm}\): the time to compute a point multiplication on the elliptic curve group; \(t_{eca}\): the time to compute a point addition on the elliptic curve group.

Algorithm 5: Role specification in HLPSL for the session of our scheme.

In the session segment, all the basic roles, alice and bob, are instanced with concrete arguments. The top-level role (called the environment) is always defined in the specification of HLPSL language, which has the global constants and a composition of one or more sessions, where the intruder may play some roles as legitimate users. The intruder (i) participates in the execution of protocol as a concrete session during the simulation. Goals are given in their own section, which generally comes at the end of a HLPSL specification. We have two secrecy goals and four authentication processes in the specification of HLPSL in our scheme.

(i) secrecy_of_subs1: it represents that \(X_i\) is kept secret to the server \(S_j\) only.

(ii) secrecy_of_subs2: it represents that \(PW_i\) and \(K\) are kept secret to the user \(U_j\) only.

(iii) authentication_on_alice_bob_tc: \(U_j\) (the smart card) generates a timestamp \(T_i\). When the server \(S_j\) receives \(T_i\) in the messages from \(U_i, S_j\) authenticates \(U_j\).

(iv) authentication_on_alice_bob_rc: \(U_j\) (the smart card) generates a random nonce \(R_i\), where \(R_i\) is only known to the user \(U_j\). When the server \(S_j\) receives \(R_i\) in the messages from \(U_i, S_j\) authenticates \(U_j\).

(v) authentication_on_bob_alice_ts: \(S_j\) generates a timestamp \(T_i\). When \(U_j\) receives \(T_i\) in the messages from \(S_j, U_j\) authenticates \(S_j\).

(vi) authentication_on_bob_alice_rs: \(S_j\) generates a random nonce \(R_i\), where \(R_i\) is only known to \(S_j\). When the user \(U_j\) receives \(R_i\) in the messages from \(S_j, U_j\) authenticates \(S_j\).

5.2. Analysis of Results. The simulation results of our scheme using the AVISPA web tool [30] for the widely accepted OFMC back-end [31] are shown in Table 7. It is evident from the summary of the results under OFMC back-end that our scheme is safe. Thus, our scheme is secure against the passive attacks and the active attacks.
6. Performance Comparison with Related Schemes

In this section, we compare the performance of our scheme with the related recently proposed password-based remote user authentication schemes: Lee and Liu [13], Das and Bruhadeshwar [14], Sonwanshi et al. [3], and Jiang et al. [15].

For communication cost comparison, we assume that the identity of a user/server is 160 bits, the random nonce is 160 bits, the timestamp is 32 bits, and the hash value is 160 bits. Since the security of 163-bit ECC (elliptic curve cryptography) is the same as that for 1024-bit RSA cryptosystem, we take the elliptic curve over a 163-bit prime field and the modulus in RSA as 1024 bits. Thus, each elliptic curve point addition and that of multiplication take \((163 + 163) = 326\) bits as these are again a point in the elliptic curve, whereas the ciphertext in RSA is 1024 bits.

In our scheme, during the login phase, the login request message \((NID', M_2, M_3, T_j)\) requires \((160 + 160 + 160 + 32) = 512\) bits. During the authentication phase of our scheme, the authentication request message \((M_7, M_8, M_9, T_j)\) requires \((160 + 160 + 160 + 32) = 512\) bits and, finally, the authentication acknowledgment message \((M_{13})\) requires 160 bits. Summing all these, the total communication cost of our scheme during the login and authentication phases becomes \((512 + 512 + 160) = 1184\) bits. In Table 4, we have compared the communication cost of our scheme with other related recent password-based schemes [3,13–15] for the login and authentication phases. It is noted that Sonwanshi et al.’s scheme [3] requires less communication cost as compared to our scheme and other schemes. However, Sonwanshi et al.’s scheme [3] is shown to be insecure against offline password guessing attack and stolen smart card attack, and it also

Table 6: Functionality comparison between our scheme and other schemes.

<table>
<thead>
<tr>
<th>Functionality</th>
<th>[13]</th>
<th>[14]</th>
<th>[3]</th>
<th>[15]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F3</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F5</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F6</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>F7</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>F8</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F9</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F10</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F11</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>F12</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>F13</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>F14</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F15</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>F16</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes: \(F_1:\) whether it protects against strong replay attacks or not; \(F_2:\) whether it protects against man-in-the-middle attacks or not; \(F_3:\) whether it protects against privileged insider attacks or not; \(F_4:\) whether it protects against impersonation attacks or not; \(F_5:\) whether it protects against password guessing attacks or not; \(F_6:\) whether it protects against denial-of-service attacks or not; \(F_7:\) whether it provides mutual authentication or not; \(F_8:\) whether it provides password changing freely and correctly or not; \(F_9:\) whether it provides formal security proof or not; \(F_{10}:\) whether it establishes a secret session key between \(U_i\) and \(S_j\) after successful authentication or not; \(F_{11}:\) whether it provides formal security verification or not; \(F_{12}:\) whether it establishes a secret session key or not; \(F_{13}:\) whether it provides local password verification or not; \(F_{14}:\) whether it provides password changing freely and correctly or not; \(F_{15}:\) whether it requires any password verification or not; \(F_{16}:\) whether it provides local password verification or not.

Table 7: The result of the analysis using OFMC of our scheme.

<table>
<thead>
<tr>
<th>% OFMC</th>
<th>% Version of 2006/02/13</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFE</td>
<td></td>
</tr>
</tbody>
</table>

DETAILS
BOUND_NUMEROFSSESSIONS
PROTOCOL
/home/avispa/web--interface--computation/.
tempdir/workfiletnHXXFr.if

GOAL
as_specified
BACKEND
OFMC
COMMENTS
STATISTICS
parseTime: 0.00 s
searchTime: 0.30 s
visitedNodes: 13 nodes
depth: 4 plies

scheme [3] is shown to be insecure against offline password guessing attack and stolen smart card attack, and it also
suffers to protect strong replay attacks. On the other hand, our scheme requires less communication cost as compared to [13–15].

In Table 5, we have compared the computation cost of our scheme with other schemes [3, 13–15] for all the phases. In our scheme, the registration phase requires only 4 hash computations. We ignore the cost of the bitwise XOR operation as it is negligible. The login and authentication phases require 14 hash computations, whereas the password change phase requires 6 hash computations. Thus, a total of 24 hash computations are required for all the phases in our scheme. It is noted that the time taken for a hash computation is significantly less as compared to that for modular exponentiation in RSA encryption/decryption and elliptic curve point addition/multiplication [32]. Thus, our scheme performs significantly better in terms of computational costs than Lee-Liu’s scheme [13], Das-Bruhadeshwar’s scheme [14], and Jiang et al’s scheme [15]. Though Sonwanshi et al’s scheme [3] requires less computational cost than our scheme, Sonwanshi et al’s scheme is insecure.

Finally, we have compared the functionality provided by our scheme with those for other schemes [3, 13–15] in Table 6. From this table, it is clear that our scheme performs better than Lee-Liu’s scheme [13] and Sonwanshi et al’s scheme [3]. Further, our scheme is also comparable to Das-Bruhadeshwar’s scheme [14] and Jiang et al’s scheme [15]. However, Lee-Liu’s scheme [13] has several security weaknesses as shown in [14], and Das-Bruhadeshwar’s scheme [14] and Jiang et al’s scheme [15] require more communication and computational costs as compared to our scheme. Further, Sonwanshi et al’s scheme [3] is insecure against different attacks. Thus, our scheme performs better in terms of various functionalities as compared to Sonwanshi et al’s scheme [3].

7. Conclusion

In this paper, we have proposed a new robust and secure three-factor remote user authentication scheme, which uses the user’s identity, the user’s password, and the smart card. Our scheme avoids the expensive operations like modular exponentiation and ECC point addition/multiplication operations as used in [13–15]. Our scheme uses the efficient bitwise XOR operations and one-way hash computations. Due to this, our scheme requires significantly less communication and computational overheads as compared to those for other existing schemes. Our scheme supports several extra features as compared to other schemes. Further, through the rigorous informal and formal security analysis, we have shown that our scheme is secure against possible known attacks. In addition, we have performed the simulation for the formal security analysis to check whether our scheme is secure against passive and active attacks. The simulation results stated in this paper clearly show that our scheme is secure against passive and active attacks. Our scheme also supports efficiently the password change phase always locally without contacting the remote server and correctly. As a result, high security and low communication and computational costs make our scheme more suitable for practical applications.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to acknowledge the many helpful suggestions of the anonymous reviewers and the editors of this journal.

References


Submit your manuscripts at
http://www.hindawi.com