Received January 24, 2022, accepted January 30, 2022, date of publication February 7, 2022, date of current version February 11, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3149315

Improved Secure and Efficient Chebyshev Chaotic Map-Based User Authentication Scheme

JIHYEON RYU[1], DONGWOO KANG[2], AND DONGHO WON[2]

[1]Department of Software, Sungkyunkwan University, Suwon, Gyeonggi-do 16419, South Korea

[2]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Gyeonggi-do 16419, South Korea

Corresponding author: Dongho Won (dhwon@security.re.kr)

This work was supported by the Institute of Information & Communications Technology Planning Evaluation (IITP) Grant funded by the Korea Government (MSIT) (Development of National Statistical Analysis System using Homomorphic Encryption Technology) under Grant 2021-0-00558.

ABSTRACT A Chebyshev chaotic map is a method for implementing efficient authentication. Recently, Chatterjee *et al.* proposed an authentication scheme for multi-server/multi-client environments using a Chebyshev chaotic map. However, we found four critical vulnerabilities in the proposed authentication method. To begin with, the method is open to user as well as server impersonation attacks. User revocation is also infeasible. Finally, there is a possibility of critical information leakage. We propose a new scheme that overcomes the four weaknesses listed previously using a symmetric Chebyshev chaotic map. We tested our proposed scheme using ProVerif and AVISPA, and compared its performance with that of the conventional authentication method. Results indicate that our scheme is 44.025 times more efficient than the conventional method.

INDEX TERMS Authentication Chebyshev chaotic map.

I. INTRODUCTION

The Chebyshev chaotic map is a promising cryptographic primitive that can be used for encryption because it is efficient and provides a reasonable level of security and randomness owing to its chaotic property. However, in 2005, Bergamo *et al.* [1] claimed that the Chebyshev chaotic map is unsuitable for encryption purposes due to its vulnerability to key recovery attacks. Consequently, many researchers [2]–[4] have proposed alternate techniques, including the discrete Chebyshev chaotic map [4] and the Chebyshev chaotic map with symmetric encryption [3]. Unsurprisingly, however, these methods also introduce additional operations such as modular multiplication and symmetric encryption, which result in longer authentication times, thus reducing the Chebyshev chaotic map's efficiency.

This motivated us to design a new data encapsulation method that incorporates the Chebyshev chaotic map. Recently, Chatterjee *et al.* [2] proposed a Chebyshev chaotic map-based two-factor multi-server authentication scheme. We found that their scheme is susceptible to user and server impersonation attacks, critical information leakage, user traceability, and infeasibility of user revocation. To mitigate these issues, we developed an efficient authentication scheme using the symmetric Chebyshev chaotic map. To demonstrate the feasibility of our proposed scheme, we compared it to more conventional methods and found that our implementation is approximately 44.025 times more efficient than the average of the compared schemes.

A. OUR CONTRIBUTION

The key contributions of this paper are as follows:

1) We propose a new data encapsulation method, dubbed the "symmetric Chebyshev chaotic map," and mathematically prove that it is theoretically secure under the computational model.
2) We present a multi-server/multi-client authentication scheme using the symmetric Chebyshev chaotic map. The proposed authentication scheme is resistant to well-known attacks, including impersonation and stolen smart card attacks.
3) By performing automated simulations, we prove that our new scheme is secure against identity/password guessing attacks, man-in-the-middle attacks, etc.
4) Our proposed authentication scheme is more cost-efficient than existing authentication schemes. When we run a comparison test, it is observed that our

The associate editor coordinating the review of this manuscript and approving it for publication was Diana Gratiela Berbecaru.

proposed scheme is approximately 44.025 times faster than the average of the compared schemes. Furthermore, our scheme exhibits the best performance compared to all existing methods we tested.

All codes concerning the formal proof and performance test have been uploaded as https://doi.org/10.6084/m9.figshare.12 198834 [5].

B. ORGANIZATION OF THE PAPER

The remainder of this paper is organized as follows. In Section II, we present an overview of related works and the history of this field. In Section III, we discuss some preliminaries, such as the Chebyshev polynomial and Bergamo attacks, which were used in this study. Chatterjee *et al.*'s scheme is discussed in detail in Section IV. In Section V, we detail several security issues present in the scheme of Chatterjee *et al.* In Section VI, we propose a symmetric Chebyshev chaotic map and authentication protocol, and describe them in detail. In Section VII, we mathematically prove the accuracy and time complexity of the symmetric Chebyshev chaotic map. Furthermore, we verify the secureness of our proposed authentication scheme by simulating it using the ProVerif and Automated Validation of Internet Security Protocols and Application (AVISPA) tools and by assessing its resistance to several common types of attacks. In Section VIII, we present the results of a performance test and compare them with those of other existing protocols. Finally, in Section IX, we provide further discussion and conclude this paper.

II. RELATED WORK

Ever since Lamport first invented the two-party (client/server) authentication scheme in 1981 [6], many researchers have proposed authentication schemes for a single-server environment [7], [8]. Unfortunately, these authentication schemes typically do not provide strong security because they are based on passwords or are prone to side-channel attacks [9]. In 2009, Das introduced the first two-factor authentication scheme [10], and since then, many researchers have started using two- and multi-factor authentication schemes to ensure a strong level of security.

The recent development of Internet-of-Things technologies and sensor networks has increased the need for a lightweight multi-server scheme. There are several ways to develop such a scheme. One such approach uses hash-based authentication. In 2006, Wong *et al.* [11] first proposed a lightweight user authentication protocol that uses only cryptographic hash functions and exclusive-OR (XOR) operations. Despite the efficiency of Wong *et al.*'s scheme, Das [10] found several vulnerabilities (same login-id threat and stolen-verifier attack) and proposed an enhanced version of the protocol. However, other researchers found further weaknesses (do not provide key agreement) in Das's scheme. Since then, many researchers have repeatedly improved upon the hash-based scheme by repeating break-fix procedures [12]–[15].

In addition to hash-based authentication, the RSA and Elgamal authentication schemes have also been studied by multiple researchers [16], [17]. These studies found that to improve the efficiency of computing systems, authentication protocols require a large key space (1024 bits and more) to guarantee security. To address the weakness of the RSA and Elgamal authentication schemes, researchers have proposed an elliptic curve cryptography (ECC)-based protocol [18], [19]. This scheme has been widely used in recent years because of its small key space.

In 2018, Chatterjee *et al.* introduced a Chebyshev chaotic map-based two-factor multi-server authentication scheme [2]. However, we observed that this scheme is vulnerable to several classes of attacks, such as user and server impersonation attacks, as well as critical information leakage. Moreover, users can be traced (user traceability), and a user revocation phase cannot be realized (infeasibility of user revocation).

III. PRELIMINARIES

A. CRYPTOGRAPHIC HASH FUNCTION

A hash function is used to convert an arbitrary-length message into a fixed-length message and check the integrity of the original message. Applications of hash functions include hash-based message authentication code and checksum. However, in cryptography, the hash function (often referred to as a cryptographic hash function) is used to verify the validity of a message by analyzing its properties. To be a cryptographic hash function, the function must satisfy the following properties:

Definition (Cryptographic Hash Function): Given a function $H : \{0, 1\}^n \rightarrow \{0, 1\}^x$, *where* x *is fixed and* $\forall m \in \{0, 1\}^n$, *the hashed message* $H(m) = h \in \{0, 1\}^x$ *must satisfy the following properties:*

- *Pre-image resistance (one-wayness): For a given value of* h, *it is infeasible to determine* m *such that* $H(m) = h$.
- *Second pre-image resistance (weak collision resistance): For a given value of* m_1, *it is infeasible to determine* m_2 *such that* $m_1 \neq m_2$ *and* $H(m_1) = H(m_2)$.
- *Collision resistance (strong collision resistance): It is infeasible to determine any* (m_1, m_2) *pairs such that* $m_1 \neq m_2$ *and* $H(m_1) = H(m_2)$.

B. CHEBYSHEV POLYNOMIAL

The Chebyshev polynomial $T_n(x)$ is a general solution of the Chebyshev differential equation when n is an integer. Due to its simplicity, it is one of the most commonly used chaotic maps in authentication schemes [20]–[22]. The Chebyshev polynomial can be defined in two ways:

Definition (Trigonometric Definition): A function $T_n(x)$: $(Z^+, [-1, 1]) \rightarrow [-1, 1]$ *is defined as in Eq 1:*

$$T_n(x) = cos(n\ arccos(x)), \tag{1}$$

where $n \in Z^+$ *and* $|x| \leq 1$. *If* $x = cos(\theta)$, *where* $\theta \in [0, \pi]$, *then* $T_n(x) = cos(n\theta)$.

TABLE 1. Summarize of related work.

Author	Weakness
Two party schemes [6]–[8]	side-channel attacks
Wong et al. [11]	same login-id threat, stolen-verifier attack
Das [10]	does not provide key agreement
elliptic curve cryptography (ECC)-based authentication scheme [16], [17]	need a large key
Chatterjee et al. [2]	impersonation attacks, traceability, infeasibility of user revocation

Definition (Recurrence Relation): A function $T_n(x)$: $(Z^+, [-1, 1]) \rightarrow [-1, 1]$ is defined as in Eq 2:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_n(x) &= 2xT_{n-1}(x) - T_{n-2}(x), \quad \text{where} \quad n \geq 2. \end{aligned} \tag{2}$$

Here are some examples of the Chebyshev polynomial (Eq 3):

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_2(x) &= 2x^2 - 1 \\ T_3(x) &= 4x^3 - 3x \\ T_4(x) &= 8x^4 - 8x^2 + 1 \end{aligned} \tag{3}$$

The Chebyshev polynomial has several interesting properties. Among them, we introduce some crucial features that make this polynomial desirable as a cryptographic function.

Property (Semi-Group Property): With $n \in Z^+$ and $m \in Z^+$, the following property holds:

$$T_n(T_m(x)) = T_m(T_n(x)) = T_{nm}(x)$$

Property (Chaotic Property): The Lyapunov exponent (λ) is one of the most frequently used measurements for determining unpredictability in a chaotic system. When the Lyapunov exponent is positive, a system exhibits chaotic behavior. The Chebyshev chaotic map has a chaotic property when $n > 2$ because its Lyapunov exponent ($T_n(x)$: $[-1, 1] \rightarrow [-1, 1]$) is $\lambda = ln(n) > 0$ when $n > 2$. Therefore, in general (for $n > 2$), the Chebyshev map exhibits chaotic characteristics [23], [24].

C. BERGAMO ATTACKS ON THE CHEBYSHEV POLYNOMIAL

In 2005, Bergamo *et al.* introduced an attack on the Chebyshev chaotic map-based cryptography method [1]. On the basis of the public key, an attacker could easily find collision keys, for which the result of the Chebyshev polynomial is the same as that for the corresponding private key. A precise explanation is given below.

Definition (Chebyshev Chaotic Map Diffie–Hellman (DH) Problem): With the Chebyshev chaotic map, we can define the DH problem as follows:

$$\text{given } x,\ T_r(x),\ T_s(x),\ \text{it is infeasible to calculate } T_{rs}(x).$$

where $T_n(x)$ is a Chebyshev polynomial ($T_n(x)$: $(Z^+, [-1, 1]) \rightarrow [-1, 1]$), and $r, s \in Z^+$.

Definition (Chebyshev Public Key Cryptosystem): A public key cryptosystem can be implemented using the Chebyshev chaotic map:

1) *Alice selects a large random $s \in Z^+$ and a random $x \in [-1, 1]$.*
2) *She computes $T_s(x)$ and declares $(x, T_s(x))$ and s as the public and private keys, respectively.*
3) *Bob obtains Alice's public key $(x, T_s(x))$.*
4) *He selects a large random $r \in Z^+$.*
5) *He computes $T_{rs}(x) = T_r(T_s(x))$ and $X = M \cdot T_{rs}(x)$.*
6) *Furthermore, he sends the ciphertext $C = (T_r(x), X)$ to Alice.*
7) *Alice computes $T_{rs}(x) = T_s(T_r(x))$*
8) *Finally, she recovers $M = X/T_{rs}(x)$.*

Theorem (Bergamo et al.'s Attack): If both $T_s(x)$ and x are known, then one can determine s' such that $T_s(x) = T_{s'}(x)$. More precisely,

$$s' = \frac{arccos(T_s(x)) + 2k\pi}{arccos(x)} \quad fork \in Z^+. \tag{4}$$

In addition, let $b = \frac{2\pi}{arccos(x)}$ and $b' = (bmod1) \cdot B^L$(where L is the maximum digit). Then, the number of collision keys s' is $gcd(b', B^L)$.

Definition (Discrete Chebyshev Chaotic Map): One of the alleviations of the Bergamo attack is the discrete Chebyshev chaotic map. This map has properties similar to those of the Chebyshev chaotic map (e.g., semi-group and chaotic properties). Moreover, a discrete Chebyshev chaotic map problem in Z_{p^d} has proven to be computationally equivalent to the discrete logarithm problem in Z_{p^d}. A DH problem along with the discrete Chebyshev chaotic map is described as follows:

$$\begin{aligned} &\text{With } x \ (mod\ p^d),\ T_r(x)\ (mod\ p^d) \text{ and } T_s(x)\ (mod\ p^d), \\ &\qquad \text{it is still infeasible to calculate } T_{rs}(x)\ (mod\ p^d). \end{aligned}$$

where, p is an odd prime, $d \in Z^+$, $T_n(x)$ is a Chebyshev polynomial on Z_{p^d} ($T_n(x)(modp^d)$: $(Z^+, Z_{p^d}) \rightarrow Z_{p^d}$), and $r, s \in Z^+$.

D. AUTHENTICATION PROCESS OF MULTI-SERVER/MULTI-CLIENT ENVIRONMENT

Much like the scheme presented by Chatterjee *et al.*, our proposed scheme work in a multi-server/multi-client environment consisting of a user, server, and registration center. After the user and server are registered in the registration center, the user can request a login to all servers logged by the registration center. This environment provides the advantage

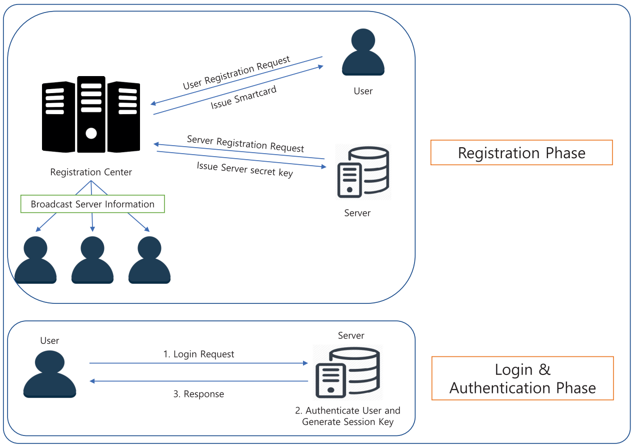

FIGURE 1. **Authentication process of multi-server/multi-client environment.**

that the user does not have to register separately for each server. The process comprises three major phases.

1) Registration phase: The server and user register themselves in the registration center, the server is issued public information, and the user is issued a smart card.
2) Login phase: The user requests login to the server that provides the service user wants.
3) Authentication phase: The server verifies the user's identity through the user table received from the Registration Center and generates a session key to be used in future communication with the user.

The authentication process of a multi-server/multi-client environment is shown in Fig. 1.

E. THREAT MODEL AND AUTHENTICATION GOAL

In this study, we describe the threat model that we constructed under a set of generally applicable assumptions, including the abilities of an attacker in a multi-server/multi-client environment. An attacker is a Dolev-Yao intruder under the Dolev-Yao formal model in [25]. In addition, the attacker of a typical user authentication scheme is capable of the following [26]–[28].

- All devices are not tamper-resistant, so the attacker can extract sensitive information from devices by implementing side-channel attacks [29].
- Valid users can also be attackers.
- An attacker can eavesdrop all login and authentication messages transmitted through insecure public channels.
- An attacker can modify or resend the eavesdropped messages.
- An attacker can easily guess low-entropy passwords and identities off-line in polynomial time [30].

Under this threat model, the user authentication scheme must satisfy the following conditions and establish the session key.

1) Sensitive information pertaining to users and servers, such as identity, password, and secret key, should not be leaked to or derivable by an attacker.
2) Valid login and authentication messages should only be generated for registered users and servers.
3) The session key, an encryption key used for communication between the user and server, should not be leaked to or derivable by an attacker.

IV. REVIEW OF CHATTERJEE *et al.*'s SCHEME

In 2018, Chatterjee *et al.* proposed a multi-server/multi-client environment authentication scheme using the Chebyshev chaotic map, cryptographic hash function, and symmetric key cryptosystem [2]. By implementing this authentication scheme, an efficient and secure multi-server environment can be established. The scheme consists of five phases: registration, login and authentication, password and biometric change, dynamic server addition, and user revocation and re-registration. Table 2 presents the notation used in Chatterjee *et al.*'s scheme.

TABLE 2. **Notations used in this paper.**

Notation	Description
U_i	An i-th user
S_j	A j-th server
RC	The registration center
ID_i	i-th user's identity
SID_j	j-th server's identity
x_j	j-th server's secret key
$T_x(y)$	A Chebyshev polynomial
K_s	A secret parameter for all servers
K_u	A secret parameter for all users
K_{u_i}	A secret parameter for a user U_i
$H(x)$	A one-way cryptographic hash function
$BH(x)$	A secure biohashing function
$E_k(x)/D_k(x)$	A symmetric encryption/decryption
n	The number of users
m	The number of servers
m'	The number of added servers ($m' << m$)
$\|\| / \oplus$	A concatenation/bitwise XOR operation

A. REGISTRATION PHASE

In the registration phase, personal information cannot be leaked because all operations in this phase are performed offline. Therefore, Chatterjee *et al.* assumed that all communications are transmitted through a secure channel. The registration phase is split into two sub-phases: user registration and server registration.

1) SERVER REGISTRATION PHASE

When a server registers the multi-server environment, the following procedures are executed via secure channel in the following order:

1) First, a registration center (RC) selects random values K_s and K_u in the interval $(-\infty, \infty)$. K_s and K_u are secret parameters of the chaotic maps for all servers and all clients, respectively.
2) A server S_j selects its identity SID_j and transmits SID_j to RC through a secure channel.

3) RC selects a server S_j's secret key x_j and uses it to compute the Chebyshev polynomials $T_{x_j}(K_s)$ and $T_{x_j}(K_u)$.
4) RC transmits $\{SID_j, T_{x_j}(K_s), T_{x_j}(K_u), x_j\}$ to S_j.

2) USER REGISTRATION PHASE

When a client registers the multi-server environment, the following procedures are performed via a secure channel:

1) A user U_i chooses their identity ID_i and password PW_i, and imprints their biometric information B_i using a sensor or a mobile device. Then, U_i generates a random secret number R_i and computes the parameters $HID_i = H(ID_i \parallel R_i \parallel T_i)$, $b_i = BH(B_i)$, $RPW_i = H(HID_i \parallel PW_i \parallel b_i \parallel R_i)$, $K_i = H(b_i \parallel R_i \parallel HID_i)$, and $C_i = R_i \oplus H(b_i \parallel ID_i \parallel PW_i)$ using the current timestamp T_i (which will be used as the registration timestamp). Consequently, U_i sends $\{HID_i, T_i, K_i, C_i, RPW_i\}$ to RC via a secure channel.
2) After receiving the parameters from U_i, RC picks a random master key x_i and Chebyshev parameter K_{u_i}. Furthermore, RC calculates the Chebyshev polynomials $T_{x_i}(K_{u_i})$ and $T_{x_i}(K_u)$, and computes $MSK_i = K_i \oplus x_i$, $P = K_s \oplus H(x_i \parallel K_i)$, and $A_i = H(ID_i \parallel RPW_i \parallel T_i \parallel T_{x_i}(K_{u_i}) \parallel x_i \parallel P)$.
3) RC saves $\{HID_i, T_i, A_i, T_{x_i}(K_{u_i}), T_{x_i}(K_u), C_i, MSK_i, P, (SID_j, T_{x_j}(K_s) \mid 1 \leq j \leq m+m')\}$ into U_i's smart card, and sends the smart card to U_i securely.
4) When the registration of U_i is complete, RC selects a unique random U_{r_i}, calculates $Uh_i = H(T_{x_i}(K_{u_i}) \parallel Ur_i)$, and transmits the result to every server. Each server stores $\{Uh_i, Ur_i,$ and $HID_i\}$ for U_i.
5) Finally, RC maintains only $\{HID_i, T_i, Ur_i\}$ for U_i.

B. LOGIN AND AUTHENTICATION PHASE

In this phase, a user U_i attempts to login to a server S_j, and S_j and U_i try to authenticate each other. Chatterjee *et al.* assumed that an anonymous user can eavesdrop on those messages because all transmissions between them are sent through a public channel.

1) A user U_i enters their identification ID_i and password PW_i, and imprints their biometric information B_i. Using its stored parameters, U_i's smart card computes $b_i = BH(B_i)$, $R_i' = C_i \oplus H(ID_i \parallel PW_i \parallel b_i)$, $HID_i' = H(ID_i \parallel R_i' \parallel T_i)$, $K_i' = H(b_i \parallel R_i' \parallel HID_i')$, $RPW_i' = H(HID_i' \parallel PW_i \parallel b_i \parallel R_i')$, $x_i' = K_i' \oplus MSK_i$, and $A_i' = H(ID_i \parallel RPW_i' \parallel T_i \parallel T_{x_i}(K_{u_i}) \parallel x_i' \parallel P)$.
2) The smart card checks whether A_i' matches the stored value A_i. If the login is successful, U_i may select a specific server S_j to connect to. Furthermore, U_i's smart card computes $K_s = P \oplus H(x_i \parallel K_i)$ and $SK_1 = H(T_{x_j}(K_s) \parallel HID_i \parallel SID_j \parallel TS_i)$, where TS_i is the current timestamp, and then calculates the Chebyshev polynomial $T_{K_1} = T_{x_i}(T_{x_j}(K_s))$ and $T_{x_i}(K_s)$. Finally, the smart card selects a random nonce RN_i and transmits $\{HID_i, SID_j, TS_i, H(K_i \parallel TS_i \parallel HID_i \parallel SID_j \parallel RN_i \parallel T_{x_i}(K_u) \parallel T_{K_1})$ and $E_{SK_1}(HID_i \parallel SID_j \parallel T_{K_1} \parallel T_{x_i}(K_s) \parallel T_{x_i}(K_u) \parallel T_{x_i}(K_{u_i}) \parallel RN_i \parallel K_i)\}$ to S_j.
3) After receiving the message from U_i, S_j checks the freshness of the message by testing whether $|TS_i - TS_i^*| < \Delta TS_i$, where ΔTS_i is the maximum transmission delay and TS_i^* is the current timestamp of S_j. If the freshness of the message is accepted, S_j computes $SK_1' = H(T_{x_j}(K_s) \parallel HID_i \parallel SID_j \parallel TS_i)$ and decrypts $E_{SK_1}(HID_i \parallel SID_j \parallel T_{K_1} \parallel T_{x_i}(K_s) \parallel T_{x_i}(K_u) \parallel T_{x_i}(K_{u_i}) \parallel RN_i \parallel K_i)$.
4) S_j searches for $\{Uh_i, Ur_i, HID_i\}$ using HID_i and computes $Uh_i' = H(T_{x_i}(K_{u_i}) \parallel Ur_i)$ using $T_{x_i}(K_{u_i})$ from the decrypted ciphertext. Furthermore, S_j verifies whether $Uh_i = Uh_i'$. If the test returns true, S_j computes $H(K_i \parallel TS_i \parallel HID_i \parallel SID_j \parallel RN_i \parallel T_{x_i}(K_u) \parallel T_{K_1})$ and compares it to the received hash value.
5) Upon completion of the authentication process, S_j computes $T_{K_2} = T_{x_j}(T_{x_i}(K_{u_i}))$, $Y = K_i \oplus T_{K_2}$, $SK_2 = H(T_{x_i}(K_{u_i}) \parallel SID_j \parallel HID_i \parallel TS_i \parallel TS_j \parallel RN_i \parallel T_{K_1}')$, and $T_{K_3} = T_{x_j}(T_{x_i}(K_u))$, where TS_j is the current timestamp. Subsequently, S_j transmits $\{HID_i, SID_j, H(RN_j \parallel TS_j \parallel Y \parallel T_{K_3} \parallel T_{x_j}(K_u)), E_{SK_2}(HID_i \parallel SID_j \parallel Y \parallel T_{x_j}(K_u) \parallel RN_j \parallel T_{K_3})\}$ to U_i.
6) After receiving the message from S_j, U_i checks the freshness of the message by confirming whether $|TS_j - TS_j^*| < \Delta TS_j$, where TS_j^* is the current timestamp of U_i. If the test is successful, U_i computes $SK_2' = H(T_{x_i}(K_{u_i}) \parallel SID_j \parallel HID_i \parallel TS_i \parallel TS_j \parallel RN_i)$, decrypts $E_{SK_2}(HID_i \parallel SID_j \parallel Y \parallel T_{x_j}(K_u) \parallel RN_j \parallel T_{K_3})$. Furthermore, U_i computes $T_{K_2}' = H(b_i \parallel R_i \parallel HID_i) \oplus Y$ and $T_{K_3}' = T_{x_i}(T_{x_j}(K_u))$, and checks whether $T_{K_3}' = T_{K_3}$.
7) When the final verification is completed, U_i generates $SK_{ij} = H(HID_i \parallel SID_j \parallel TS_i \parallel TS_j \parallel RN_i \parallel RN_j \parallel T_{K_1}' \parallel T_{K_2} \parallel T_{K_3})$ and S_j generates $SK_{ij} = H(HID_i \parallel SID_j \parallel TS_i \parallel TS_j \parallel RN_i \parallel RN_j \parallel T_{K_1}' \parallel T_{K_2} \parallel T_{K_3})$, where SK_{ij} will be used as the session key between U_i and S_j.

C. PASSWORD AND BIOMETRIC CHANGE PHASE

In this phase, a user U_i can change their password PW_i and biometric imprint B_i without any involvement from the registration center RC. To update their information, U_i must perform the following steps:

1) To authenticate themselves, U_i inputs their identification ID_i, original password PW_i^{old}, and biometric imprint B_i^{old}. Furthermore, with a secure biohashing function $BH(\cdot)$, U_i's smart card computes $b_i' = BH(B_i^{old})$, $R_i' = C_i \oplus H(ID_i \parallel PW_i^{old} \parallel b_i')$, $HID_i' = H(ID_i \parallel R_i' \parallel T_i)$, $RPW_i' = H(HID_i' \parallel PW_i^{old} \parallel b_i' \parallel R_i')$, $x_i' = H(b_i' \parallel R_i' \parallel T_i) \oplus MSK_i$, and $A_i' = H(ID_i \parallel RPW_i' \parallel T_i \parallel T_{x_i}(K_{u_i}) \parallel x_i' \parallel P)$.
2) U_i's smart card compares A_i' with the stored parameter A_i. If the two parameters do not match, the procedure is terminated. Otherwise, the smart card requests the

user to enter their new password PW_i^{new} and biometric information B_i^{new}.

3) When U_i enters a new password PW_i^{new} and biometric information B_i^{new}, the smart card computes the following equations:
$b_i^{new} = BH(B_i^{new})$,
$C_i^{new} = R_i' \oplus H(b_i^{new} \parallel ID_i \parallel PW_i^{new})$,
$RPW_i^{new} = H(HID_i' \parallel PW_i^{new} \parallel b_i^{new} \parallel R_i')$,
$A_i^{new} = H(ID_i \parallel RPW_i^{new} \parallel T_i \parallel T_{x_i}(K_{u_i}) \parallel x_i' \parallel P)$.
4) After the above computation is completed, U_i's smart card replaces A_i with A_i^{new} and C_i with C_i^{new}.

D. DYNAMIC SERVER ADDITION PHASE

In this phase, a new server S_j requests to join an existing network. The following processes are carried out to add S_j to the network:

1) The registration center RC assigns the unique identifier SID_j and S_j's secret key x_j to S_j, and calculates the Chebyshev polynomial $T_{x_j}(K_s)$. Furthermore, RC assigns the unique identifier SID_j and S_j's secret key x_j to S_j, and calculates the Chebyshev polynomial $T_{x_j}(K_s)$. Subsequently, RC transfers $\{SID_j, T_{x_j}(K_s), T_{x_j}(K_u)$, and $x_j\}$ to S_j through a secure channel, and S_j securely stores the received parameters.
2) After S_j successfully joins the existing network, RC broadcasts the registration of S_j to all registered users.

E. USER REVOCATION AND RE-REGISTRATION PHASE

Chatterjee *et al.* also assumed that a user U_i could lose their smart card. In this case, the following procedures are performed to revoke U_i's identity, and U_i can register again without changing their identity.

1) When U_i loses their smart card, U_i informs RC, which transmits a revocation message about U_i with $\{HID_i, Uh_i, Ur_i\}$ to every server.
2) After receiving the revocation message from RC, all servers simply put a revocation flag on the corresponding data of U_i in their database. In addition, they reject all authentication messages relating to U_i with the revocation flag.
3) If an authentic user U_i tries to register again using the same revoked identity ID_i, RC first verifies U_i by checking their authorized documents, and then finds and reactivates U_i's hashed identity HID_i.

V. SECURITY ANALYSIS OF CHATTERJEE *et al.*'s SCHEME

In this section, we discuss several security weaknesses of Chatterjee *et al.*'s scheme. Overall, there are four major vulnerabilities: user impersonation attacks, server impersonation attacks, critical information leakage, and infeasibility of user revocation. The basic assumption regarding these attacks is that an attacker is a valid user within the existing network and can extract parameters from their own smart card using side-channel attacks. These attacks are described in detail in the following subsections.

A. USER IMPERSONATION ATTACK

Within Chatterjee *et al.*'s scheme, an attacker A can forge a session key SK_{ij} after intercepting authentication messages M_1 and M_2. Furthermore, A could forge a fake login message M_1' using the intercepted parameters between U_i and S_j. A detailed description of the process is as follows (the overall attack trace is described in Fig. 2):

1) An attacker A, who is a valid user, extracts the server-key-plus-ID $(SID_j, T_{x_j}(K_s))$ from their own smart card.
2) Then, A chooses a victim user U_i and waits for U_i to send a login message.
3) Upon intercepting the login message M_1, A produces a symmetric key $SK_1 = H(T_{x_j}(K_s) \parallel HID_i \parallel SID_j \parallel TS_i)$, decrypts $E_{SK_1}(HID_i \parallel SID_j \parallel T_{K_1} \parallel T_{x_i}(K_s) \parallel T_{x_i}(K_u) \parallel T_{x_i}(K_{u_i}) \parallel RN_i \parallel K_i)$, and obtains the secret parameters T_{K_1}, $T_{x_i}(K_s)$, $T_{x_i}(K_u)$, $T_{x_i}(K_{u_i})$, and K_i.
4) When a server S_j sends an authentication message M_2 to U_i, A wiretaps M_2 and calculates the symmetric key $SK_2 = H(T_{x_i}(K_{u_i}) \parallel SID_j \parallel HID_i \parallel TS_i \parallel TS_j \parallel RN_i)$.
5) Finally, A generates a session key SK_{ij} between U_i and S_j. Whenever U_i or S_j transmits a secret message, A can use the forged session key SK_{ij} to decrypt it.
6) Furthermore, A can fabricate a fake login message M_1' with the intercepted parameters from M_1 and M_2. This fake login message can successfully force S_j into acting as if it is communicating with U_i.

B. SERVER IMPERSONATION ATTACK

The scheme presented by Chatterjee *et al.* is also susceptible to server impersonation attacks. Similar to user impersonation attacks, after eavesdropping on M_1 and M_2, an attacker intercepts a new authentication message M_1', and deceives U_i into thinking that S_j communicating with U_i. This is explained in detail below (the overall attack trace is described in Fig. 3):

1) Like in a user impersonation attack, an attacker A intercepts the messages M_1 and M_2 between user U_i and server S_j.
2) A then creates a symmetric key SK_1 using an extracted server-key-plus-ID $(SID_j, T_{x_j}(K_s))$, and decrypts $E_{SK_1}(HID_i \parallel SID_j \parallel T_{K_1} \parallel T_{x_i}(K_s) \parallel T_{x_i}(K_u) \parallel T_{x_i}(K_{u_i}) \parallel RN_i \parallel K_i)$ in M_1.
3) Using the collected parameters, A generates a symmetric key SK_2 and decrypts $E_{SK_2}(HID_i \parallel SID_j \parallel Y \parallel T_{x_j}(K_u) \parallel RN_j \parallel T_{K_3})$.
4) A waits until U_i re-authenticates S_j. When U_i sends the re-authentication message M_1' to S_j, A intercepts M_1' and computes the symmetric key SK_2' using TS_i', RN_i' in M_1', and previously collected parameters.
5) Consequently, A can successfully create a fake server message, M_2', and transmit it to U_i. Because A has access to all private parameters between U_i and S_j, U_i can be easily fooled into thinking that they are communicating with S_j.

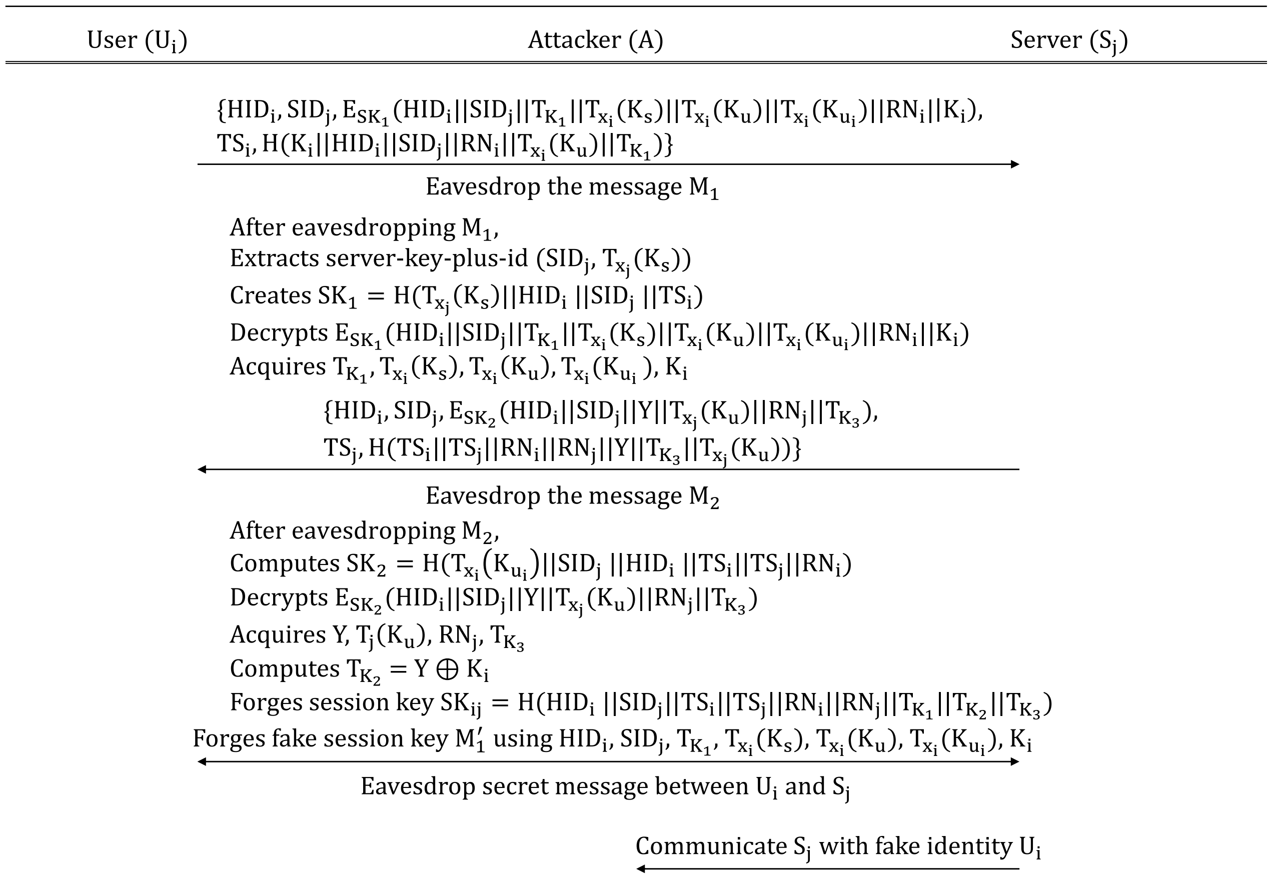

FIGURE 2. **User impersonation attack.**

C. INFEASIBILITY OF USER REVOCATION

Although Chatterjee *et al.*'s scheme does feature a user revocation phase, it is unattainable. To revoke a user U_i's identity, RC should find $< Uh_i, Ur_i,$ and $HID_i >$ in its database. However, U_i cannot be expected to remember the random values of Ur_i and R_i. Considering that U_i can only provide the value of ID_i, RC cannot calculate either $Uh_i = H(T_{x_i}(K_{u_i}) \parallel Ur_i)$ or $HID_i = H(ID_i \parallel R_i \parallel T_i)$; thus, $< Uh_i, Ur_i, HID_i >$ cannot be calculated, and a revocation message cannot be sent to every server.

D. CRITICAL INFORMATION LEAKAGE

Finally, Chatterjee *et al.*'s scheme may leak important information about users and servers. These crucial parameters could cause a user anonymity breach, as well as the leakage of private keys and hashed user identities.

1) PARAMETER x_i

A user U_i's secret key x_i can also be leaked in following ways. If A obtains U_i's smart card, they can acquire the stored HID_i and MSK_i by implementing a side-channel attack. With HID_i, A can find U_i's login message history and recover $K_i = H(b_i \parallel R_i \parallel HID_i)$, (like in a user impersonation attack). Finally, A can acquire $x_i = MSK_i \oplus K_i$,

2) PARAMETER HID_i

HID_i is a hashed identity whose associated user cannot be determined by attacker A directly; however, because HID_i remains the same in every authentication phase, A can determine whether user U_i' matches U_i or not using HID_i. As a result, users become traceable.

VI. PROPOSED SCHEME

In this section, we introduce a new scheme to overcome the vulnerabilities of the scheme proposed by Chatterjee *et al.* To do so, we present a new type of data encapsulation: a symmetric Chebyshev chaotic map. This map is used to alter the user's pseudo-identity for each login request (in our scheme, HID_i) to eliminate the user traceability present in the existing scheme.

A. SYMMETRIC CHEBYSHEV CHAOTIC MAP

The Chebyshev chaotic map is inappropriate for asymmetric encryption because an integer r can be easily derived from the Chebyshev polynomial $T_r(x)$ using the Bergamo approach. However, thanks to Bergamo's approach, we define the symmetric Chebyshev chaotic map.

Definition (Symmetric Chebyshev Chaotic Map Cryptosystem): Let $x \in [-1, 1]$ be the shared key, and $n \in [2, 2^L]$ be

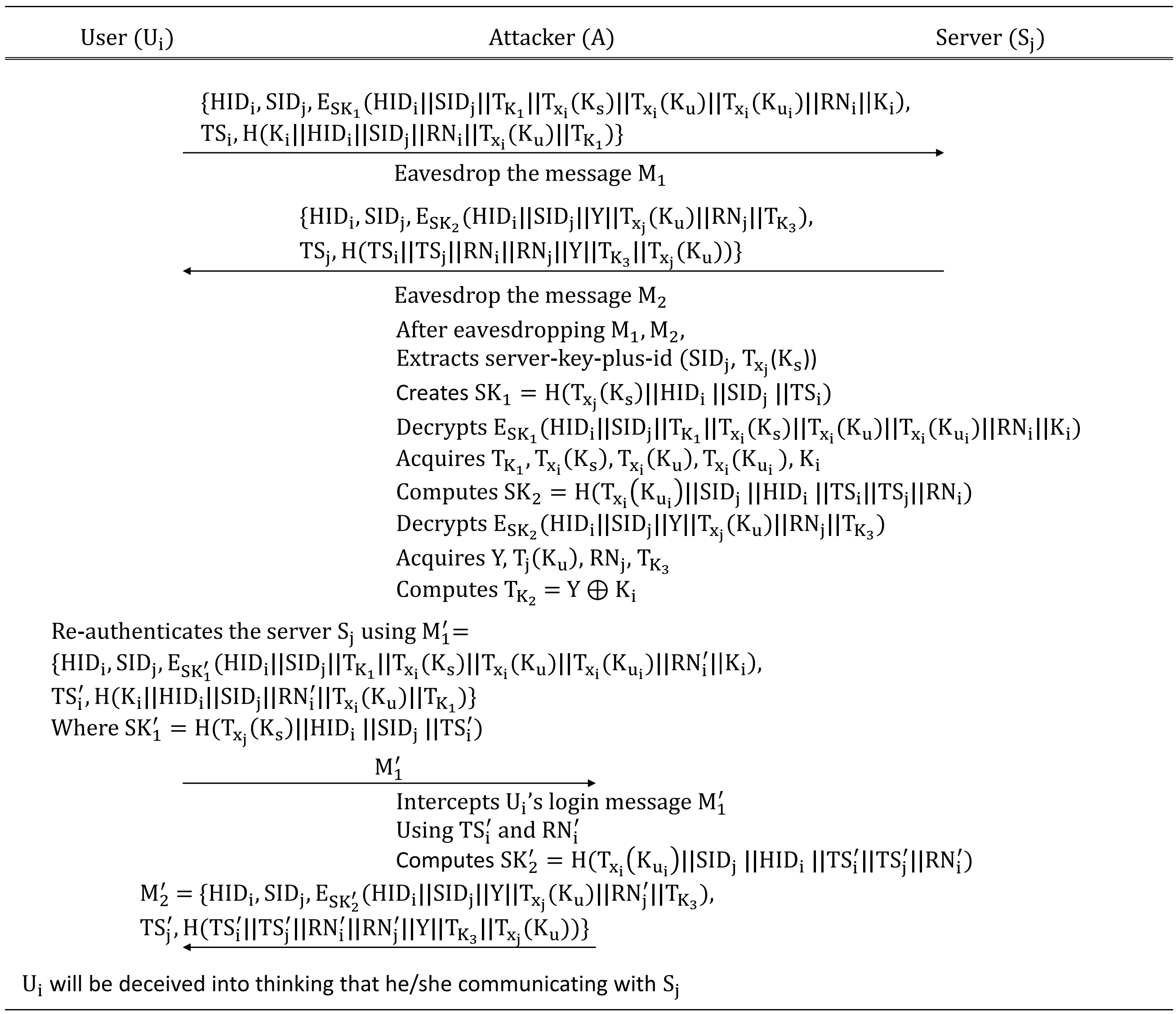

FIGURE 3. Server impersonation attack.

the message. Then, $T_n(x)$ is an encapsulated message that no one can derive n from without knowing x.

Here, L denotes the precision length. To check the validity, one should prove that it is only possible to derive n by using x. This explanation is discussed in Section VII.

B. REGISTRATION PHASE

We now introduce a new authentication protocol that incorporates the Chebyshev chaotic map. This new scheme consists of four phases: registration, login and authentication, password change, and user revocation and re-registration. Furthermore, the registration phase is split into sub-phases of server and user registration.

The registration phase occurs in a secure channel. The server and user register their information by communicating 1:1 with the secure channel provided by the registration center. It is assumed that attackers cannot access this channel, so an attack is impossible.

1) SERVER REGISTRATION PHASE

The following steps are performed to register a server S_j:

1) First, a server S_j selects a unique server identifier SID_j and transmits it to the registration center RC through a secure channel.
2) After receiving SID_j from S_j, RC selects a random x_j in Z_p, where p is a predefined large prime. x_j is used as an element of a public key pair. Furthermore, RC chooses a large natural number s_j (must be greater than two), which is used as the secret key of S_j.
3) RC then computes the discrete Chebyshev polynomial modular p, $T_{s_j}(x_j)$ (mod p). Then, RC broadcasts S_j's

Server (S_j)		Registration Center (RC)
Select SID_j		
	$\{SID_j\}$ →	
		Select random $x_j \in Z_p$, secret key $s_j \in N$, $(s_j > 2)$
		Compute discrete Chebyshev polynomial $T_{s_j}(x_j)$ (mod p)
		Post public key $\{SID_j, x_j, T_{s_j}(x_j)$ (mod p)$\}$
	← $\{SID_j, x_j, T_{s_j}(x_j), s_j\}$	

FIGURE 4. Server registration phase.

identifier SID_j and public key pair $(x_j, T_{s_j}(x_j))$ (mod p) to all users.

4) Finally, RC transmits $\{SID_j, x_j, T_{s_j}(x_j), s_j\}$ to S_j through a secure channel.

The overall phase is described in Fig. 4.

2) USER REGISTRATION PHASE

User U_i takes the following steps to register with the registration center RC.

1) First, user U_i selects their identification ID_i and password PW_i, and imprints their biometric information B_i. Then, using biometric information, U_i computes $b_i = BH(B_i)$, where $BH(\cdot)$ is a secure biohashing function. Then, U_i selects a random number R_i and computes $C_i = R_i \oplus H(ID_i \parallel PW_i \parallel b_i)$, $V_i = H(ID_i \parallel PW_i \parallel b_i \parallel R_i)$, and hashed ID $HID_i = H(ID_i \parallel b_i)$. Subsequently, U_i sends $\{HID_i$, C_i, and $V_i\}$ to RC through a secure channel.
2) When U_i's registration request arrives, RC first chooses a random nonce Ur_i and computes $UID_i = H(HID_i \parallel Ur_i)$. Next, RC converts the bit string into a floating point number between 0 and 1, and checks the last digit. If the last digit is 0, $gcd(UID_i \cdot 2^L, 2^L) \neq 1$; thus, RC picks a new Ur_i' until $gcd(UID_i \cdot 2^L, 2^L) = 1$.
3) If RC generates Ur_i successfully, it also generates U_i's smart card, and inputs $\{V_i, C_i, UID_i\}$ into the smart card securely. RC then sends U_i to the smart card through a reliable process, such as a secure channel.
4) If U_i's registration is complete, RC transmits U_i's information (HID_i, UID_i) to all registered servers.
5) When a server S_j receives U_i's information (HID_i, UID_i), S_j hashes the information with its secret key s_j, and stores $\{H(H(HID_i \parallel x_j) \parallel s_j), UID_i\}$ into its database. If this process is completed successfully, S_j discards all information relating to HID_i.
6) RC stores $\{HID_i, UID_i\}$ in its secure database to be used if U_i loses their smart card and revokes their identity.

the overall phase is described in Fig. 5.

After the registration phase is completed, the private and public information held by the user and server are presented in Table 3, where s_j is the private server key and $[x_j, T_{s_j}(x_j)]$ is the public key pair of the Chebyshev Public Key Cryptosystem. The server's public information is available to all users who want to access the server, and through this information, the user can specify the server to access.

User (U_i)		Registration Center (RC)
Select identity ID_i, password PW_i		
Biometric imprint B_i		
Compute $b_i = BH(B_i)$		
Where $BH(\cdot)$: secure biohash function		
Select random secret number R_i		
Compute $C_i = R_i \oplus H(ID_i\|\|PW_i\|\|b_i)$		
$V_i = H(ID_i\|\|PW_i\|\|b_i\|\|R_i)$		
$HID_i = H(ID_i\|\|b_i)$		
	$\{HID_i, C_i, V_i\}$ →	
		Select random nonce U_{r_i}
		Compute $UID_i = H(HID_i\|\|U_{r_i})$
	← Smartcard $= \{V_i, C_i, UID_i\}$	
		Sends $\{HID_i, UID_i\}$ to all server S_j
		(S_j : stores $\{H(HID_i\|\|s_j), UID_i\}$)
		Where s_j : server's secret key
		Keeps $\{HID_i, UID_i\}$

FIGURE 5. User registration phase.

TABLE 3. Secret, public information of server and user.

	Secret Information	Public Information
Server	s_j $(UID_i, H(H(HID_i\|\|x_j)\|\|s_j))$ matching table	SID_j $x_j\ (modp)$ $T_{s_j}(x_j)\ (modp)$
User	User's identity and password Biometric information Smartcard : $<V_i, C_i, UID_i>$	$None$

C. LOGIN AND AUTHENTICATION PHASE

When a user U_i completes the registration process and wants to log in to a server S_j, U_i performs the following steps:

1) First, a user U_i inputs their identity ID_i, password PW_i, and biometric information B_i, and their smart card computes the following equations: $b_i = BH(B_i)$, where BH is a secure biohashing function; $R_i' = C_i \oplus H(ID_i \parallel PW_i \parallel b_i)$, where C_i is the stored parameter; and $V_i' = H(ID_i \parallel PW_i \parallel b_i \parallel R_i')$.
2) U_i's smart card compares V_i' with the stored parameter V_i. If their values are equal, U_i selects a server S_j that they want to log on to. U_i's smart card then finds S_j's public information $\{SID_j, x_j, T_{s_j}(x_j)\,(\text{mod}\,p)\}$. Next, the smart card generates random natural numbers r_i and RN_1. With the parameter RN_1, it calculates the Chebyshev polynomial $T_{RN_1}(UID_i)$ and checks whether its last digit is odd or even. If the digit is even, the smart card chooses another RN_1' until the last digit of $T_{RN_1'}(UID_i)$ is odd.
3) In addition, the smart card computes the discrete Chebyshev polynomial $T_{r_is_j}(x_j)(\text{mod } p) = T_{r_i}(T_{s_j}(x_j))(\text{mod } p)$ and $T_{r_i}(x_j)(\text{mod } p)$, and multiplies $H(HID_i||x_j)$ by $T_{r_is_j}(x_j)(\text{mod } p)$. Finally, the smart card transmits $\{SID_j$, $T_{r_i}(x_j)(\text{mod } p)$, $H(HID_i||x_j) \cdot$

$T_{r_i s_j}(x_j)$(mod p), $T_{RN_1}(UID_i)$, and $H(UID_i \parallel RN_1)\}$ to S_j via public channel.

4) Upon receiving the message from U_i, S_j computes $T_{r_i s_j}(x_j) = T_{s_j}(T_{r_i}(x_j))$ (mod p) and acquires $H(HID_i||x_j) = (H(HID_i||x_j) \cdot T_{r_i s_j}(x_j))/ T_{s_j}(T_{r_i}(x_j))$ (mod p)
5) S_j finds UID_i from its database using the acquired $H(HID_i||x_j)$ and S_j's private key s_j. With UID_i and $T_{RN_1}(UID_i)$, S_j obtains RN_1' using Bergamo's approach. Furthermore, S_j verifies whether RN_1' matches RN_1 by checking if $H(UID_i \parallel RN_1')$=$H(UID_i \parallel RN_1)$. If RN_1' fails the test, S_j finds another RN_1' until the hashed value is equal to the received hash value. When S_j finds the correct value, S_j chooses a random nonce RN_2 until $T_{RN_2}(H(UID_i \parallel RN_1'))$'s last digit is odd. Lastly, S_j sends U_i $\{T_{RN_2}(H(HID_i \parallel RN_1'))$, $H(UID_i \parallel RN_1' \parallel RN_2)\}$.
6) After receiving the message from S_j, U_i obtains RN_2 using Bergamo's approach on $T_{RN_2}(H(HID_i \parallel RN_1))$ with $H(HID_i \parallel RN_1)$. U_i then identifies the correct RN_2' from the possible RN_2' by comparing $H(UID_i \parallel RN_1 \parallel RN_2')$ and $H(UID_i \parallel RN_1 \parallel RN_2')$.
7) Finally, U_i and S_j compute a session key $SK_{ij} = T_{RN_1}(T_{RN_2}(H(HID_i \parallel RN_1 \parallel RN_2'))$ using their received or generated parameters.

The summarized login and authentication process is displayed in Fig. 6.

D. PASSWORD CHANGE PHASE

A user can change their password without any involvement from the registration center RC. Unlike the scheme proposed by Chatterjee *et al.*, our scheme does not include a biometric information change phase because the user's biometric information experiences minimal changes. This is explained in detail as follows:

1) First, a user U_i enters their identity ID_i, password PW_i^{old}, and biometric information B_i into U_i's smart card.
2) Then, the smart card computes $b_i = BH(B_i)$, $R_i' = C_i \oplus H(ID_i \parallel PW_i^{old} \parallel b_i)$, and $V_i' = H(ID_i \parallel PW^{old} \parallel b_i \parallel R_i')$ using the stored parameter C_i. Thereafter, the smart card compares V_i' with the stored parameter V_i. If the two values are equal, the card requests U_i to enter the new password. Otherwise, the smart card terminates the password-change phase.
3) After U_i inputs a new password PW_i^{new}, U_i's smart card calculates $C_i^{new} = R_i' \oplus H(ID_i \parallel PW_i^{new} \parallel b_i)$ and $V_i^{new} = H(ID_i \parallel PW_i^{new} \parallel b_i \parallel R_i')$.
4) Finally, the smart card replaces C_i^{old} and V_i^{old} with C_i^{new} and V_i^{new}, respectively.

E. USER REVOCATION AND RE-REGISTRATION PHASE

If a user U_i loses their smart card, they should revoke their account by following the user revocation phase. Our scheme incorporates a procedure similar to that of Chatterjee *et al.*'s scheme. U_i does not have to change their ID_i during the re-registration phase because the identity ID_i is not leaked. For revocation, U_i may need to prove their identity by providing authorized documentation; however, this is not necessary as RC contains $HID_i = H(ID_i \parallel b_i)$, which is hashed with biometric information. Consequently, U_i only needs their hashed identity HID_i using the identity ID_i and hashed biometric information b_i. Further information regarding this topic is detailed as follows:

1) For the revocation, a user U_i needs to prove themselves by providing HID_i to the registration center RC through a secure channel. Prior to the request, U_i calculates $b_i = BH(B_i)$ and $HID_i = H(ID_i \parallel b_i)$ using ID_i and B_i, respectively, and sends HID_i to RC. Subsequently, RC finds UID_i by searching for HID_i in its database and transmits a revocation message of UID_i to all servers through a secure channel.
2) After receiving the revocation message, all servers mark the revocation flag on UID_i in their database. Thus, each server rejects all login requests with UID_i.
3) When U_i wants to re-register, U_i transmits the re-registration message containing HID_i. After receiving U_i's re-registration request, RC initiates the registration phase and sends the re-registration message with UID_i to all servers, which then remove the revocation flag in their database.

VII. SECURITY ANALYSIS

In this section, we assess the security of our new scheme by performing several analyses. First, we present a mathematical proof of our new symmetric Chebyshev chaotic map. This proof is divided into two parts: correctness, which shows that there are at most two possible answers if the decryptor knows the private key, and time complexity, which shows that an attacker cannot find the correct answers within a reasonable timeframe. Furthermore, we simulated our new authentication scheme using automated tools such as ProVerif and AVISPA, both popular tools for providing formal proof of authentication. However, each tool has its own advantages and disadvantages. Although ProVerif allows users to define specific functions, it does not support associative functions; therefore, the user might be stuck when associative functions, such as XOR, are required. In contrast, AVISPA includes a predefined XOR function; however, the user cannot declare a custom function. Furthermore, these tools cannot simulate advanced features, such as weak authentication and infinite transmission. In addition, neither of the automated tools can assess the safety of a simulation. In particular, ProVerif cannot prove user authenticity in our scheme. Therefore, we use these tools to show a more rigid authentication scheme that can resist diverse attacks. Finally, we also include a security requirement analysis in which the formal proof cannot be determined. We list more than 10 attacks and check whether our authentication scheme is robust

User (U_i)	Server (S_j)
	Public key : $\{SID_j, x_j \pmod p, T_{s_j}(x_j) \pmod p\}$
Inputs user's identity ID_i, password PW_i Biometric imprint B_i Compute $b_i = BH(B_i)$ $R_i' = C_i \oplus H(ID_i \| \| PW_i \| \| b_i)$ $V_i' = H(ID_i \| \| PW_i \| \| b_i \| \| R_i')$ Checks $V_i' = V_i$. If failed, terminates process Select server S_j Get S_j's public key $\{SID_j, x_j, T_{s_j}(x_j) \pmod p\}$ Select random $r_i, RN_1 \in N$ Computes $T_{r_i s_j}(x_j) \pmod p = T_{r_i}(T_{s_j}(x_j)) \pmod p$	
$\{SID_j, T_{r_i}(x_j) \pmod p, HID_i, T_{r_i s_j}(x_j) \pmod p, T_{RN_1}(UID_i), H(UID_i \| \| RN_1)\}$ →	
	Compute $T_{r_i s_j}(x_j) \pmod p = T_{s_j}(T_{r_i}(x_j)) \pmod p$ $H(HID_i \| \| x_j) = \frac{(H(HID_i \| \| x_j) \cdot T_{r_i s_j}(x_j))}{T_{s_j}(T_{r_i}(x_j))} \pmod p$ Find UID_i using $H(HID_i \| \| x_j)$ and S_j's secret key s_j By Bergamo approach, $RN_1' = T_{RN_1}(UID_i)$ Check $h(UID_i \| \| RN_1') = H(UID_i \| \| RN_1)$ If failed, find other RN_1' Otherwise, select random RN_2
← $\{T_{RN_2}(H(HID_i \| \| RN_1')), H(UID_i \| \| RN_1' \| \| RN_2)\}$	
By Bergamo approach, $RN_2' = T_{RN_2}(H(HID_i \| \| RN_1))$ Check $H(UID_i \| \| RN_1 \| \| RN_2') = H(UID_i \| \| RN_1' \| \| RN_2)$ If failed, find other RN_2' Generate $SK_{ij} = T_{RN_1}(T_{RN_2'}(H(HID_i \| \| RN_1 \| \| RN_2')))$	Generate $T_{RN_1'}(T_{RN_2}(H(HID_i \| \| RN_1' \| \| RN_2)))$

FIGURE 6. **Login and authentication phase.**

against them; thereafter, we can confirm that our scheme is secure.

A. FORMAL PROOF OF THE SYMMETRIC CHEBYSHEV MAP

To use a symmetric Chebyshev chaotic map in practice, a mathematical proof is required to show that this method is sufficiently safe. Such a mathematical proof is presented in this subsection. First, we prove that the number of results of the symmetric Chebyshev chaotic map is at most two, and then we evaluate the time complexity of the map.

Theorem (Correctness of the Cryptosystem): Let $x \in [-1, 1]$, $arccos(x) \nmid \pi$, *and* $\frac{2\pi}{arccos(x)}$ *'s last digit be odd* ($mod\ 2^L$) *or* $gcd(\lfloor \frac{2\pi}{arccos(x)} \cdot 2^L \rfloor, 2^L) = 1$)*, where L is the number of digits; then, by knowing x, we can decrypt* $T_n(x)$ *and obtain a maximum of two possible users* n'.

Proof (Bergamo's Approach): Let $x \in [-1, 1]$ and $n \in N$; then,

$$T_n(x) = cos(n \cdot arccos(x))$$
$$n' = \frac{\pm arccos(T_n(x)) + 2k\pi}{arccos(x)}, \quad k \in N.$$

If $n' \in N$, $T_{n'}(x) = T_n(x)$ because

$$\begin{aligned} T_{n'}(x) &= cos(\frac{\pm arccos(T_n(x)) + 2k\pi}{arccos(x)} \cdot arccos(x)) \\ &= cos(\pm arccos(T_n(x)) + 2k\pi) \\ &= cos(\pm arccos(T_n(x))) \\ &= cos(arccos(T_n(x))) \\ &= T_n(x). \end{aligned}$$

This shows that the number of users n' is important. Let $a = \frac{\pm arccos(T_n(x))}{arccos(x)}$ and $b = \frac{2k\pi}{arccos(x)}$. Then,

$$n' = \pm a + bk \tag{5}$$

In addition, a and b can be separated into integer and decimal parts. $a = \lfloor a \rfloor + a'$, $b = \lfloor b \rfloor + b'$ (where $0 \le a'$, $b' < 1$), and only the fractional part of Eq 5 is considered.

$$0 = \pm a' + b'k. \tag{6}$$

Now, Eq 6 can be considered with two cases: a real number case and a finite precision (mod 2) (for the real implementation) case.

Case 1. (real number case) The following equation holds:

$$b'k \equiv \mp a' \ (\text{mod}1)$$
$$b'k = \mp a' + k', \ k' \in Z$$
$$k = \mp \frac{a'}{b'} + \frac{k'}{b'}, \ k' \in Z$$

We assume that $arccos(x)$ is not divisible by π; thus, b' is irrational. This means that $\frac{1}{b'}$ is also irrational and $\frac{k'}{b'}$ cannot be an integer. Therefore, there are only two solutions to this equation.

$$n' = \pm(a - b\frac{k' + a'}{b'})$$

Because one of the users n' must be n, $n' = \pm n$. Moreover, $n' > 0$ by definition; therefore, the only possible solution is $n' = n$.

Case 2 (Finite precision (mod 2^L) case). Let us assume that the finite precision is 2^{-L}. Then,

$$b'k \equiv \mp a' \ (\text{mod}1).$$

Multiplying 2^L on both sides, we have

$$b'' \cdot k \equiv \mp a'' \ (\text{mod}2^L),$$

where $b'' = b' \cdot 2^L$ and $a'' = a' \cdot 2^L$. $gcd(b'', 2^L) = 1$ on the premise; therefore, the inverse b'^{-1} exists mod 2^L. Therefore,

$$k \equiv \mp(b''^{-1} \cdot a'') \ (\text{mod}2^L)$$

As a result, the possible k are $k = 2^L - (b''^{-1} \cdot a'')$ and $k = b''^{-1} \cdot a''$.

Theorem (Complexity of the Cryptosystem): Let $x \in [-1, 1]$ *and* $n \in N$. *Given the value of* $T_n(x)$*, it takes a time complexity of* $O(2^{2L-1})$ *to guess the value of* n *using a brute-force method if the value of* x *is not given.*

Proof: Because $T_n(x) = cos(n \cdot arccos(x))$ (Eq 1),

$$n \cdot arccos(x) + 2k\pi = arccos(T_n(x)), k \in Z.$$

$arccos(x) : [-1, 1] \to [0, \pi]$ is a continuous function; thus, $x' \in [0, \pi] = arccos(x)$ is still possible. As a result,

$$n \cdot x' = arccos(T_n(x)) - 2k\pi.$$

For a random n,

$$x' = \frac{arccos(T_n(x)) - 2k\pi}{n}.$$

```
(* channel *)
free ca: channel[private].
free cb: channel[private].
free cc: channel.
```

FIGURE 7. ProVerif channel code.

Because $\frac{arccos(T_n(x))}{n} \to [0, \frac{\pi}{n}]$, $\frac{2k\pi}{n} \to [-\pi, \frac{\pi}{n}]$ and $k \to [-\frac{n}{2}, \frac{1}{2}]$. As a result, there are at least $\lfloor \frac{n+1}{2} \rfloor$ possible (x', n) pairs for each n. Moreover, if an exhaustive search is applied from 1 to n, the time complexity of finding the correct pair (x, n) is $O(\frac{n(n+1)}{2}) = O(2^{2L-1})$, where L is the number of digits.

B. SIMULATION USING THE ProVerif TOOL

ProVerif is an automated analysis tool for cryptographic protocols based on the Dolev-Yao model. This tool can systematically prove cryptographic properties, such as reachability, secrecy, correspondence, and some observational equivalence properties.

ProVerif features two unique characteristics in its design. First, it uses an extension of the pi-calculus with cryptography; thus, it supports various types of cryptographic primitives. In addition, ProVerif analyzes protocols after translating them into Horn clauses; therefore, it can verify security features with an unbounded number of sessions. More information on ProVerif can be found in [28], [31], [32]. In this study, we only explain an important part of our ProVerif code. We have uploaded the complete code to the following figure: https://doi.org/10.6084/m9.figshare.12198834 [5].

In our ProVerif code, there are three channels, as shown in Fig. 7 (i.e., *ca*, *cb*, and *cc*).

Channel *ca* is a registration channel between a user U_i and the registration center RC, and channel *cb* is a registration channel between RC and a server S_j. These channels are considered secure, and an attacker A cannot intercept the registration message from *ca* or *cb*. In contrast, *cc* is a public channel where anyone can access all authentication messages, and most authentications are done through this public channel.

Fig. 8 shows the ProVerif user code. In regard to U_i's private key, we use three variables: ID_i, PW_i, and Bio_i.

We first declared ID_i and PW_i as weak secrets because users want to remember their identities and passwords with ease; therefore, their identities and passwords have low entropies. Consequently, we designated ID_i and PW_i as weak secrets and performed an offline guessing attack to assess secrecy. In addition, we defined *secretU* and *secretS* to check whether an attacker can obtain the session key.

To verify the security level of the secret values (ID_i, PW_i, Bio_i, *secretU*, and *secretS*), we included *query attacker*$(\cdot)$.

```
(* user's secret *)
type identity.
type password.
type biometric.
type nonce.
free ID_i: identity [private].
weaksecret ID_i.
free PW_i: password [private].
weaksecret PW_i.
free Bio_i: biometric [private].
```

FIGURE 8. ProVerif user code.

```
(* query *)
query attacker(ID_i).
query attacker(PW_i).
query attacker(Bio_i).
query attacker(secretU).
query attacker(secretS).
query x: bitstring; event(endUi(x)) -> event(beginUi(x)).
query x: identity; inj-event(endSj(x)) -> inj-event(beginSj(x)).
```

FIGURE 9. ProVerif query code.

```
(* chebyshev polynomial *)
type S [large].
fun cheb(bitstring, S): bitstring.
reduc forall x1:bitstring, s1:S; bergamo(cheb(x1, s1), x1) = s1.
(* chebyshev polynomial on large prime P*)
type X [bounded].
fun discheb(X, S): X.
fun mult(bitstring, bitstring): bitstring.
equation forall a:bitstring, b:bitstring; mult(a, b) = mult(b, a).
reduc forall a1:bitstring, b1:bitstring; div(mult(a1, b1), a1) = b1.
```

FIGURE 10. ProVerif definition code.

```
(* Verification table *)
table verif(bitstring, bitstring).
```

FIGURE 11. ProVerif table code.

We also accounted for the correspondence assertion by adding a *query event*$(\cdot)->$ *event*$(\cdot)$ (Fig. 9).

Fig. 10 shows the definitions of the functions used in our code. As the proposed scheme is Chebyshev chaotic map-based, we defined some related functions.

Finally, we included a verification table *verif* for use during the authentication process when a server S_j authenticates a user U_i with the stored $\{H(H(HID_i \parallel x_j)||s_j), UID_i\}$ in the *verif* table (Fig. 11).

The authentication process is divided among its participants: the user, server, registration center, registration server, and main process. We briefly summarize the ProVerif code, as the complete authentication process has already been described in Section V.

```
(* User process *)
let processU(ID_i: identity, PW_i: password, Bio_i: biometric) =
```

FIGURE 12. ProVerif user main code 1.

```
(* session key *)
let sess = cheb(cheb(hash(concat(concat(HID_i,
        S2bits(RN_1)), S2bits(RN_2))), RN_1), RN_2) in
out(cc, xor(secretU, sess)).
```

FIGURE 13. ProVerif user main code 2.

```
(* Server process *)
let processS(SID_j: identity, s_j: S) =
```

FIGURE 14. ProVerif server main code 1.

```
let HHID_i = hash(concat(HID_i, S2bits(s_j))) in
get verif(=HHID_i, UID_i) in
let RN_1 = bergamo(Trn1x, UID_i) in
```

FIGURE 15. ProVerif server main code 2.

The user process *processU* simulation code is shown in Fig. 12 and 13. *processU* has three parameters: identity (ID_i), password (PW_i), and biometric information (Bio_i).

processU is further divided into three parts: registration, login and authentication, and session key. The registration part is identical to the user registration phase (Fig. 5). In particular, all transmissions occur in a secure channel *ca*. Likewise, the login and authentication parts are identical to the login and authentication phases (Fig. 6). All messages are transmitted through the public channel *cc*. Finally, in the session key part, U_i creates a session key *sess* and transmits a secret message *secretU* encrypted using *sess* to the public channel *cc*. This part does not exist in the proposed scheme, and was added to the simulation to verify that the session key is secure from an attacker. If the attacker can derive the session key, it is implied that *secretU* can be determined using the exclusive or operation, so the result of the *query attacker (SecretU)* is true.

The server process *processS* simulation code is shown in Fig. 14 and 15. *processS* has two parameters: the identity (SID_j) and the discrete Chebyshev secret key (s_j).

The server process *processS* checks U_i's identity and creates a session key. Because S_j uses a database in the authentication phase, we implement the *get* command to collect the proper data from S_j's database.

The registration center process *processRC* simulation code is shown in Fig. 16. This process does not have any parameters. Furthermore, all transmissions are performed on a secure channel (*ca* and *cb*). Overall, the code of *processRC* is the same as that of the user registration phase (Fig. 5).

```
(* Registration process(RC) : attacker *)
let processRC =
        in(ca, (HID: bitstring, C: bitstring, V: bitstring));
        new Ur: nonce;
        let UID = hash(concat4(HID, Ur)) in
        out(cb, (HID, UID)).
```

FIGURE 16. ProVerif registration sever main code.

```
(* Registration process(S_j) : attacker *)
let processRCS(HHID_i: bitstring, SID_j: identity, ps_j: S) =
        in(cb, (HID: bitstring, UID: bitstring));
        let HHID = hash(concat(HID, S2bits(ps_j))) in
        if HHID != HHID_i then insert verif(HHID, UID).
```

FIGURE 17. ProVerif server main code 3.

```
(* Main process *)
process
        (* Constructing private key of serverS *)
        new seed: srand;
        let px_j = gpkey(seed) in
        let ps_j = gskey(seed) in
        let pTx_j = discheb(px_j, ps_j) in out (cc, SID_j, px_j, pTx_j));

        (* Inserting userU's HID into verification table *)
        new Ur_i: nonce;
        let HID_i = hash(concat5(ID_i, bhash(Bio_i))) in
        let HHID_i = hash(concat(HID_i, S2bits(ps_j))) in
        let UID_i = hash(concat4(HID_i, Ur_i)) in
        insert verif(HID_i, UID_i);

        ((!processU(ID_i, PW_i, Bio_i)) | (!processS)SID_j, ps_j))
                | (!processRC) | (!processRCS(HHID_i, SID_j, ps_j)))
```

FIGURE 18. ProVerif main code.

After user registration is complete, the server performs an additional process to store the user's information in the server's database matching table (*verif*). We define this process as *processRCS*, and the simulation code is shown in Fig. 17. During *processRCS*, there are three parameters: hashed ID with the server's public key ($H(HID_i||x_j)$), server ID (SID_j), and server's secret key (ps_j). SID_j and ps_j are used to check the correct server S_j, and $HHID_i$ is used to check whether a user's identification pair $\{HHID_i, UID_i\}$ conflicts with an existing pair.

Finally, the main process generates an overall environment by creating a public-private key pair of the discrete Chebyshev chaotic map system and registering the user U_i. Furthermore, it runs *processU*, *processS*, *processRC*, and *processRCS*. The main process simulation code is shown in Fig. 18.

After executing ProVerif, the main process prints three types of results, and each result is expressed as follows:

1) RESULT [Query] is true: The query is proven and there is no feasible attack.
2) RESULT [Query] is false: The query is false and that ProVerif has discovered a potential attack.
3) RESULT [Query] cannot be proven: Because verifying the protocols for an infinite number of sessions is infeasible, ProVerif cannot prove the query.

```
--------------------------------------------------------------------------
RESULT Weak secret ID_i is true (bad not derivable).
RESULT Weak secret PW_i is true (bad not derivable).
RESULT not attacker(ID_i[]) is true.
RESULT not attacker(PW_i[]) is true.
RESULT not attacker(Bio_i[]) is true.
RESULT not attacker(secretU[]) is true.
RESULT not attacker(secretS[]) is true.
RESULT event(endUi(x)) ==> event(beginUi(x)) is true.
RESULT inj-event(endSj(x_90)) ==> inj-event(beginSj(x_90)) is true.
--------------------------------------------------------------------------
```

FIGURE 19. Results of ProVerif simulation.

The results of the ProVerif simulation are shown in Fig. 19. User U_i's private attributes ID_i, PW_i, and Bio_i are secure under the ProVerif simulation. In particular, ID_i and PW_i, which we defined as weak secrets, are secure against offline guessing attacks. Moreover, an attacker cannot procure a session key from the server side and decrypt or obtain the server's secret message *secretS* and the user's secret message *secretU*. Regarding the correspondence assertion, ProVerif proves that server S_j corresponds to the server U_i wants to send to. However, ProVerif's attack trace includes a secure registration channel (*ca*) that should not be eavesdropped. Therefore, the proposed scheme passes the secrecy and correspondence tests, and no attacker can violate the authentication system by design.

C. SIMULATION USING THE AVISPA TOOL

In this subsection, we verify the security our proposed scheme by simulating it in AVISPA, another popular automated verification tool [33]. The advantage of AVISPA is that mathematical cryptographic primitives, such as XOR and exponential functions, are well-defined. However, a user cannot produce a new cryptographic function using this tool. Our scheme is written in a high-level protocol specification language (HLPSL). HLPSL is a role-based specification language; therefore, it is expressive enough to describe large-scale Internet protocols [34], [35]. The complete AVISPA code has been uploaded to https://doi.org/10.6084/m9.figshare.12198834 [5].

AVISPA provides four different analysis techniques: 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). Among these techniques, we use CL-AtSe as an analytical tool, as it allows extensions pertaining to the algebraic properties of cryptographic functions and the associativity of message concatenation.

The user roles are presented in Table 4. Unlike Proverif, AVISPA cannot define a secure channel. Therefore, we used a symmetric key *RPkeyi* to create a secure channel between the user and *RC*. In addition, the hash function, biohash function, and channels that follow the Dolev-Yao model are defined.

Because we cannot construct a custom Chebyshev function using AVISPA, we used the encryption operation as a countermeasure for the Chebyshev polynomial or multiplication operation. For example, the user sends $H(HID_i||x_j)\cdot$

TABLE 4. Role specification in HLPSL for the user.

```
role user(
Ui, Sj, RC : agent,
RPkeyi : symmetric_key, % used for registration phase(secure channel)
Hash : hash_func, % cyptographic hash function
BH : hash_func, % biohash function
Snd, Rcv : channel(dy))
played_by Ui def=
local
State : nat,
Pubj : (agent.text.message) set,
Ci, EHIDi : message,
Vi : hash(text.text.hash(text).text)
HIDi : hash(text.hash(text)),
UIDi : hash(hash(text.hash(text)).text),
Tsj, Tri : message,
IDi, PWi, Bi, Ri, RNi, RN1: text,
Xj, Trisj, TRN1, RN2: text,
RN3: hash(text.text.text),
Bhi : hash(text)
init

State := 0
/\Pubj := {} Database of Server public key pair
transition
% Registration phase
0. State = 0 /\Rcv(Sj.Xj'.Tsj')
=|>State' := 2 /\Pubj' := cons(Sj.Xj'.Tsj', Pubj)
/\IDi' := new() /\PWi' := new() /\Bi' := new()
% Predefined secrecy test goal about user secret information
/\secret(IDi', idi, Ui) /\secret(PWi', pwi, Ui) /\secret(Bi', bi, Ui)
/\Bhi' := BH(Bi') /\Ri' := new()
/\Ci' := xor(Ri', Hash(IDi'.PWi'.Bhi'))
/\Vi' := Hash(IDi'.PWi'.Bhi'.Ri')
/\HIDi' := Hash(IDi'.Bhi')
/\secret(HIDi, hidi, Ui, RC, Sj) % Check identity traceability
/\Snd({HIDi'.Ci'.Vi'}_RPkeyi) % Send HID_i, C_i, V_i to server

2. State = 2 /\Rcv({UIDi'.Ci.Vi}_RPkeyi) /\in(Sj.Xj'.Tsj', Pubj) % Store server's public key pair
=|>State' := 4
% login phase
/\RNi' := new() /\RN1' := new() /\Tri' := exp(Xj, RNi')
% encryption instead of multiplication
/\EHIDi' := {Hash(HIDi.Pubj)}_exp(Tsj',RNi')
% symmetric encryption instead of a Chebyshev polynomial
/\TRN1' := {RN1'}_UIDi
/\Snd(Sj.Tri'.EHIDi'.TRN1')
% Send SID_j, T_{r_i}(x_j), HID_i · T_{r_i s_i}(x_j), T_{RN_1}(UID_i) to the server
% authentication property
/\witness(Ui, Sj, rn1, RN1')
/\RN3 := Hash(HIDi.RN1)

4. State = 4 /\Rcv({RN2'}_RN3) % Receive T_{RN_2}(H(HID||RN_1)) from the server
=|>State' := 6
/\request (Ui, Sj, rn2, RN2')
end role
```

$T_{r_is_j}(x_j)$ *modp* to the server in the login phase, and the server cannot derive $H(HID_i||x_j)$ unless the value of $T_{r_is_j}(x_j)$. For this reason, we assumed that $T_{r_is_j}(x_j)$ was used as a cryptographic symmetric key and wrote a simulation code in which $H(HID_i||x_j)$ is encrypted with $T_{r_is_j}(x_j)$. Likewise, UID_i is treated as a symmetric key to derive RN_1 in $T_{RN_1}(UID_i)$. Moreover, a user U_i obtains the server's identity and public key pair (SID_j, x_j, $T_{s_j}(x_j)$) during the registration phase. However, AVISPA supports only private and symmetric keys on a one-to-one basis. Thus, we used an alternative method that stores all discrete Chebyshev public key pairs and server identifiers in the user's memory (*Pubj*). By doing so, we could perform cryptographic procedures similar to those in the discrete Chebyshev chaotic map. We then predefined the goal that the user has witnessed the value RN_1, and now requests a check of the value RN_2 from the server.

Next, we describe the role of the server in Table 5. KeyRing implements $H(H(HID_i||x_j)||s_j)$ and the UID_i matching table database. In State 0, the server selects its SID_j value and sends it to the registration center. In State 1, the public key pair (PKeyj, TPKeyj) and private key pair (SKeyj) issued from the registration center are stored, and the public key pair is sent to the user along with their SID_j. In State 2, the user's

TABLE 5. Role specification in HLPSL for the server.

```
role server(
Ui, Sj, Rc : agent,
RPKeyj : symmetric_key, % used for registration phase(secure channel)
Hash : hash_func,
BH : Bio hash_func,
Snd, Rcv : Channel(dy)
played_by Sj def=
local
State : nat,
RN1, RN2 : text,
RN3 : text.text,
KeyRing: (hash(hash(text.hash(text)).text).hash(hash(text.hash(text)).text)) set,
PKeyj : text, %public key (x_j) of discrete chebyshev
TPKeyj : message, %public key (T_{s_j}(x_j)) of discrete chebyshev
SKeyj : text, %secret key of discrete chebyshev (s_j)
HIDi : hash(text.hash(text)),
UIDi, XIDi: hash(hash(text.hash(text)).text),
Tri, Ex1 : message
init

State := 0
/\KeyRing := {} Database of user information
transition
% Server Registration Phase
0. State = 0 /\Rcv(start) =|>State' := 1 /\Snd({Sj}_RPKeyj) %Send SID_j to Registration Center

1. State = 1 /\Rcv({PKeyj'.TPKeyj'.SKeyj'}_RPKeyj) %Receive server secret key and public key pair
=|>State' := 2
/\Snd(Sj.PKeyj'.TPKeyj') % Release public key pair to user

2. State = 2 /\Rcv({HIDi'.UIDi'}_RPKeyj) /\not(in(XIDi'.UIDi', KeyRing))
=|>State' := 3
/\XIDi' := Hash(Hash(HIDi'.SKeyj).SKeyj)
/\KeyRing' := cons(XIDi'.UIDi', KeyRing) %Store User information to database

% Login and Authentication Phase
3. State = 3 /\Rcv(Sj.Tri') /\Ex1' = exp(Tri',SKeyj) /\
Rcv(Sj.{HIDi'}_Ex1') /\in(Hash(Hash(HIDi'.PKeyj).SKeyj').UIDi', KeyRing)/\Rcv(Sj.{RN1'}_UIDi')
=|>State' := 5
/\RN2' := new()
/\RN3' := Hash(HIDi.RN1)
/\Snd({RN2'}_RN3')
%Authentication Property
/\request(Sj, Ui, rn1, RN1)
/\witness(Sj, Ui, rn2, RN2)
```

HID_i and UID_i values are received and stored in the matching table KeyRing. The remaining parameters and processes are defined in the same manner as those in Section V. Then, we predefined the goal that the server requests a check of the value of RN_1, under the condition that the server witnessed the value of RN_2.

In Table 6, we delineate the role of the registration center. During server registration, a secret was declared to confirm the security of "SKj" (S_j in our scheme), which is the long term private key of the server. After user registration, the registration center sends UID_i, V_i, C_i to the user, and HID_i, UID_i to the server, as stated in our proposed scheme.

We describe the session role in Table 7. In the session role, one usually declares all the channels used by the basic roles and a composition section in which the basic roles are instantiated. The /\operator indicates that the roles should be executed in parallel. attribute, in parentheses, which specifies the intruder model one assumes for that channel. Here, the type declaration channel (dy) represents the Dolev–Yao intruder model. Our session role is described as follows:

Table 8 describes the overall environment and goal of the simulation. We establish two users ($a1$ and $a2$) and conduct sessions between these users, the server (b), and the registration center (r). All symmetric keys ($kar1$, $kar2$, and kbr) are used to build secure channels between each user and server. Intruder knowledge is limited by a set of constants and assigned variables, which are provided to the user in the role of an intruder. Suppose the intruder is playing the user $a1$;

TABLE 6. Role specification in HLPSL for the registration center.

```
role regcenter(
Ui, Sj, Rc : agent,
RPKeyi : symmetric_key, % used for registration phase(secure channel)
RPKeyj : symmetric_key, % used for registration phase(secure channel)
Hash : hash_func,
BH : Bio hash_func,
Sndi, Sndj, Rcv : Channel(dy))
played_by RC def=
local
State : nat,
Uri : text,
Ci : message,
Vi : hash(text.text.hash(text).text),
Xj, SKj : text,
HIDi : hash(text.hash(text)),
UIDi : hash(hash(text.hash(text)).text),
TXj : message
init

State := 0
transition
% Server Registration
0. State = 0 /\Rcv({Sj}_RPKeyj)
=|>State' := 1
/\Xj' := new()
/\SKj' := new()
/\TXj' := exp(Xj', SKj')
/\Sndj(Xj'.TXj'.SKj'_RPKeyj) % Send public key pair, secret key to server
/\secret(SKj', skj, {Sj, RC}) % Predefined secrey test goal about server secret information

% User Registration
1. State = 1 /\Rcv({HIDi'.Ci'.Vi'}_RPKeyi)
=|>State' := 1
/\Uri' := new()
/\UIDi' := Hash(HIDi'.Uri')
/\Sndi({UIDi'.Ci'.Vi'}_RPKeyi) %Send UID_i, C_i, V_i(smartcard) to user through secure channel
/\Sndj({HIDi'.UIDi'}_RPKeyj) %Send HID_i, UID_i to the server through secure channel
end role
```

TABLE 7. Role specification in HLPSL for session.

```
role session(
Ui, Sj, Rc : agent,
RPKeyi : symmetric_key, % used for registration phase(secure channel)
RPKeyj : symmetric_key, % used for registration phase(secure channel)
Hash : hash_func,
BH : Bio hash_func)
def=
local SI, SJ, RI, RJ, RR, SRI, SRJ : Channel(dy)
composition
user(Ui, Sj, RC, RPKeyi, Hash, BH, SI, SJ)
/\server(Ui, Sj, RC, RPKeyj, Hash, BH, RI, RJ)
/\regcenter(Ui, Sj, RC, RPKeyi, RPKeyj, Hash, BH, SRI, SRJ, RR)
end role
```

then, the intruder's knowledge contains all the terms given as parameters of the corresponding instance of the role $a1$. In this case, the total intruder knowledge is intended to be the union of sets defined in those declarations. Finally, we conduct a confidentiality test of the user's identity, password, biometric information, and pseudo-identity (ID_i, PW_i, B_i, HID_i), and the server's secret key SK_j, to verify that sensitive information does not leak. We also perform an authentication test that is predefined in the user and server roles ($rn1$, $rn2$).

The results (Fig. 20) indicate that our scheme is secure against active attacks, such as replay and man-in-the-middle attacks, as well as passive attacks. The summary of the results under CL-AtSe (Constraint Logic-based Attack Searcher) also indicates that the protocol is safe.

D. SECURITY REQUIREMENT ANALYSIS

In this subsection, we demonstrate that our scheme can resist several known attacks. Table 9 presents the results of the overall comparison among the different authentication schemes.

TABLE 8. Role in HLPSL for the goal and environment of our scheme.

```
role environment() def=
const
idi, pwi, bi, hidi, rn1, rn2, skj : protocol_id,
a1, a2, b, r : agent,
kar1, kar2, kbr : symmetric_key,
skeyj, pkeyj : text,
h, bh : hash_func,
intruder_knowledge = {a1, a2, b, r, h, bh}
composition
session(a1, b, r, kar1, kbr, h, bh)
/\session(a2, b, r, kar2, kbr, h, bh)
end role

goal
secrecy_of idi, pwi, bi, hidi, skj
authentication_on rn1, rn2
end goal
```

```
-----------------------------------------------------------------------
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  UNTYPED_MODEL
  BOUNDED_SEARCH_DEPTHPROTOCOL

/home/span/span/testsuite/results/AVISPA_test (revised).if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS
 Analysed   : 327 states
  Reachable  : 149 states
  Translation: 0.04 seconds
  Computation: 5.00 seconds
-----------------------------------------------------------------------
```

FIGURE 20. Results of AVISPA simulation.

TABLE 9. Security requirement comparison.

	[2]	[3]	[14]	[19]	Our
Privileged Insider Attack	×	×	✓	✓	✓
User Anonymity	×	×	✓	×	✓
User Untraceability	×	×	×	×	✓
Mutual Authentication	✓	✓	✓	✓	✓
Offline Identity Guessing	×	×	✓	×	✓
Offline Password Guessing	✓	✓	✓	×	✓
User Impersonation Attack	×	✓	✓	✓	✓
Server Impersonation Attack	×	✓	✓	✓	✓
Stolen Smart Card Attack	×	✓	✓	×	✓
Replay Attack	✓	✓	✓	✓	✓
Session Key Compromise	✓	✓	✓	✓	✓
No Time Synchronization	×	✓	×	×	✓
Convenient Password Change	✓	×	✓	×	✓
Provision for Revocation	×	×	✓	×	✓
Known-Key Security Attack	✓	✓	✓	✓	✓
No Need for a Trusted Third Party During Login	✓	✓	×	✓	✓

1) PRIVILEGED INSIDER ATTACK

Although an attacker A has access to a server S_j, A can only obtain $H(H(HID_i \parallel x_j)||s_j)$ and $UID_i = H(HID_i \parallel Ur_i)$. As a result, A cannot obtain any information relating to the user U_i. Moreover, even if S_j is vulnerable such that a private key s_j is leaked, the attacker cannot find HID_i because $HID_i = H(ID_i \parallel b_i)$, where the hashed biometric information b_i is considered to be robust against offline guessing attacks.

2) USER ANONYMITY AND UNTRACEABILITY

During the authentication phase, an attacker A might obtain a login and authentication message between user U_i and server S_j. However, A cannot acquire any partial information from the message. This is because all pieces of critical information (HID_i and UID_i) are encrypted by Chebyshev and discrete Chebyshev polynomials. Moreover, nonces r_i and RN_1 are changed for every login session; thus, the resulting ciphertexts of the Chebyshev and discrete Chebyshev polynomials also vary every time. Consequently, A cannot determine whether the query is from the same user U_i or from another user $U_{i'}$.

3) MUTUAL AUTHENTICATION

A server S_j can authenticate a user U_i by verifying that $HHID_i = H(H(HID_i \parallel x_j)||s_j)$, where $HHID_i$ is stored in its database. Furthermore, U_i can authenticate S_j by checking whether RN_1' is equal to RN_1, where U_i is sent to S_j. Therefore, the proposed scheme satisfies mutual authentication.

4) OFFLINE IDENTITY/PASSWORD GUESSING ATTACK

An attacker A cannot acquire any partial information about a user U_i's identity and password during the authentication phase. Let us assume that A obtains C_i and V_i using a stolen smart card or other attack. In this case, A can collect C_i and V_i; however, they cannot guess U_i's identity and password because they are concatenated and hashed with the hashed biometric information b_i, which cannot be derived. Consequently, A cannot succeed in offline identity and password-guessing attacks.

5) USER IMPERSONATION ATTACK

An attacker A requires HID_i and UID_i to impersonate a user U_i. However, these two parameters were encrypted during the authentication phase. Moreover, even if A acquires UID_i via stolen card attack, they cannot produce HID_i because the user's identity ID_i and hashed biometric information b_i are required to generate it. Therefore, A cannot successfully carry out a user impersonation attack. Furthermore, when the server tries to impersonate a user, the server also needs access to the user's HID_i. However, the value of HID_i in the login message that the user sends to the server is transmitted in the form of $H(HID_i||x_j)$, and the server needs to guess HID_i given this information. Even though the server's public key x_j is public information, $HID_i = H(ID_i \parallel b_i)$, where the hashed biometric information b_i is considered to be robust against offline guessing attacks. For this reason, a user impersonation attack is also impossible on any server.

6) SERVER IMPERSONATION ATTACK

To impersonate a server S_j, an attacker A requires S_j's secret key, which cannot be acquired. Therefore, it is infeasible for A to successfully impersonate S_j.

7) STOLEN SMART CARD ATTACK

A user U_i might lose their smart card, or have it stolen by an attacker. However, as mentioned earlier, even if A obtains C_i, V_i, and UID_i, they can neither succeed in any attack nor access any piece of U_i's private information.

8) REPLAY ATTACK

An attacker A might resend a user U_i's previous login message to server S_j. However, A cannot generate the session key SK_{ij} because they do not have access to HID_i and RN_1. Therefore, it is impossible to carry out a replay attack.

9) SESSION KEY COMPROMISE

To compromise the session key SK_{ij} within our protocol, an attacker A needs access to HID_i, RN_1, and RN_2. However, access to this information requires either a server S_j's private key s_j, or a user U_i's private information HID_i and UID_i, which are infeasible to obtain unless A is U_i or S_j. Therefore, A cannot compromise the session key SK_{ij} between U_i and S_j.

10) NO TIME SYNCHRONIZATION

A timestamp-based authentication scheme suffers from synchronization problems. However, our proposed scheme does not used any timestamps; thus, it is free from time synchronization problems.

11) FAST AND CONVENIENT PASSWORD CHANGE

When a user U_i wants to change their password, they can do so without communicating with the registration center RC and server S_j. Upon changing their password, the user's smart card replaces C_i^{old} and V_i^{old} with C_i^{new} and V_i^{new}, respectively. This process occurs entirely within U_i's smart card, and the smart card does not have to transmit the changed password to RC and S_j.

12) PROVISION FOR REVOCATION

A user U_i may lose their smart card or have it stolen. In this case, U_i needs to revoke their identity and create a new identity. Our proposed scheme includes a revocation phase, and the registration center RC can verify U_i with $HID_i = H(ID_i \parallel b_i)$; therefore, U_i can revoke and register with their identity ID_i.

13) KNOWN-KEY SECURITY (Forward Security)

Even if an earlier session key SK_{ij} is leaked, an attacker cannot recover a new SK_{ij}' because the nonces RN_1 and RN_2 are changed for every session. Therefore, the proposed scheme guarantees forward privacy.

TABLE 10. Simulation environment.

Feature	Description
Operating System	64-bit Window 10
Compiler	Visual C++ 2017 Software
Cryptographic Library	Crypto++ Library 8.1
Processor	Intel(R) Pentium(R) CPU G4600, 3.60 GHz
Memory	8.0 GB

TABLE 11. Simulation time.

Phase	Term	Simulation (μs)
Chebyshev encryption (32 bits)	T_{CHenc}	*0.0261*
Chebyshev decryption (32 bits)	T_{CHdec}	*0.3334*
Discrete Chebyshev (64 bits)	T_{DCH}	*0.1808*
SHA-1	–	4.2066
SHA-256	T_H	4.9465
AES (128 bits)	T_{AES}	5.4097
ECIES encryption	T_{ECCenc}	5855.486
ECIES decryption	T_{ECCdec}	4784.517
Biohashing	T_{BIO}	*4.9465*
Elliptic multiplication	T_{ecpm}	*2226*
Elliptic addition	T_{ecpa}	*28.8*

VIII. PERFORMANCE ANALYZE

In this section, we analyze the performance of our proposed scheme and compare it with that other authentication schemes. We mainly measure the login and authentication phases because the other phases barely occur.

The simulation environment is presented in Table 10. We used Crypto++ Library 8.1 as a cryptographic library. This library supports many cryptographic algorithms such as SHA, AES, and ECC. In addition, it provides a benchmarking program for users to easily compare cryptographic algorithms. General Crypto++ algorithm benchmarks (5.6.0) are discussed in detail in [36]. However, in our case, there is no algorithm for Chebyshev cryptographic algorithms; thus, we performed the simulation experiments manually. We calculated the execution time of the algorithm by computing the average of one million operations (10 experiments with 100,000 operations each). We uploaded the complete code (C code) and relevant sources [5].

The results of the cryptographic algorithms are listed in Table 11. Chebyshev encryption and decryption require very little time because the encryption scheme has a simple structure whereas the decryption scheme uses trigonometric functions and a Euclidean algorithm. Moreover, discrete Chebyshev mapping takes less time because the recurrence relation can be converted into matrix form, as discussed in [37]. However, the security of the discrete Chebyshev method is the same as that of RSA. Therefore, to be safe, the bits in discrete Chebyshev should be more than 1024, which will require more time than the simulation.

Some studies on authentication use SHA-1 as a hash function; however, this function is considered unsafe. Therefore, we used SHA-256 in our simulation. Fortunately, SHA-256 is not significantly slower than SHA-1; therefore, we assumed that hash functions do not affect the overall authentication scheme time, as assumed by many researchers.

Finally, there is no algorithm for biohashing functions. Therefore, we assume that the time required for biohashing functions is comparable to that of hash functions. Moreover, in accordance with [19], [38], we assume that the time required for elliptic curve point multiplication (T_{ecpm}) is 2226μs (2.226 ms), and that required for elliptic curve point addition (T_{ecpa}) is 22.8μs (0.0228 ms).

The results of the overall performance comparison among the authentication schemes are shown in Table 12. We chose four Chebyshev encryption-based schemes ([2], [3], [21], [22]), including a hash-based scheme ([14]) and an ECC-based scheme ([19]), for comparison.

Our proposed scheme is slower than the protocols presented by Xiao-Liao-Deng and Dharminder-Kumar-Gupta. However, Xiao-Liao-Deng's scheme requires five communication transmissions, which take more than 20μs, whereas Dharminder-Kumar-Gupta's scheme does not incorporate biometric information. Therefore, the security of our scheme is superior. The results suggest that our proposed scheme is more efficient than other authentication protocols.

Our scheme performs 44 times higher than the average value of 2300.24 for the other schemes. ($44.025 = (2300.24 - 51.088)/51.088$) Therefore, we propose the possibility of a user authentication scheme using a novel symmetric Chebyshev chaotic map. Our scheme exhibits the best performance among three factor authentica-

TABLE 12. Comparison of computation time.

Scheme	No. of Communications	Time	Actual Time (μs)
Chatterjee et al. [2]	2	$8T_{AES}$+T_{BIO}+$16T_H$+$6T_{CHenc}$	127.525
Xiao-Liao-Deng [3]	5	$3T_H$+$2T_{AES}$+$2T_{CHenc}$	25.711
Amin et al. [14]	4	$22T_H$	108.803
Chaudhry et al. [19]	2	$6T_{ecpm}$+T_{ecpa}+$10T_h$+$2T_{AES}$	13445.080
Dhaminder-Kunda-Mishra [21]	2	$7T_{CHenc}$ + T_{BIO} + $10T_H$	54.594
Dharminder-Kumar-Gupta [22]	2	$6T_{CHenc}$ + $8T_H$	39.729
Ours	2	$9T_H$+T_{BIO}+$2T_{CHenc}$+$2T_{CHdec}$+$5T_{DCH}$	51.088

tion schemes, and opens up new possibilities for Chebyshev chaotic maps.

IX. CONCLUSION

To create an encryption system that can be maintained for numerous data packets, we introduced a new method of high-speed symmetric encryption using the Chebyshev chaotic map, and proposed a secure and fast authentication protocol based on the map. We proved that our proposed scheme is secure by showing its resistance to well-known attacks and by using the automated authentication verification tools ProVerif and AVISPA. In addition, we have proven that our proposed scheme is fast and efficient by comparing its time and cost consumption with those of other authentication schemes. However, during the simulation, we observed that the decrypted results of the symmetric Chebyshev chaotic map did not match the original message, due to the size of the data type (long double) and the underflow effects. Therefore, it is necessary to create a new data type and its associated trigonometric functions for the symmetric Chebyshev encryption scheme, which we leave as a future discussion. If a library for symmetric Chebyshev encryption is implemented, our cryptographic function will require a very short operation time, and our authentication scheme would offer a robust and efficient environment.

REFERENCES

[1] P. Bergamo, P. D'Arco, A. de Santis, and L. Kocarev, "Security of public-key cryptosystems based on Chebyshev polynomials," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 52, no. 7, pp. 1382–1393, Jul. 2005.
[2] S. Chatterjee, S. Roy, A. K. Das, S. Chattopadhyay, N. Kumar, and A. V. Vasilakos, "Secure biometric-based authentication scheme using Chebyshev chaotic map for multi-server environment," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 5, pp. 824–839, Sep./Oct. 2018.
[3] D. Xiao, X. Liao, and S. Deng, "A novel key agreement protocol based on chaotic maps," *Inf. Sci.*, vol. 177, no. 4, pp. 1136–1142, 2007.
[4] C.-L. Hsu and T.-W. Lin, "Password authenticated key exchange protocol for multi-server mobile networks based on Chebyshev chaotic map," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PERCOM Workshops)*, Mar. 2013, pp. 90–95.
[5] D. Kang, "Simulation code for 'improved secure and efficient Chebyshev chaotic map based user authentication scheme,"' Tech. Rep., 2021, doi: 10.6084/m9.figshare.12198834.
[6] L. Lamport, "Password authentication with insecure communication," *Commun. ACM*, vol. 24, no. 11, pp. 770–772, Nov. 1981.
[7] C.-C. Chang and T. C. Wu, "Remote password authentication with smart cards," *IEE Proc. E (Comput. Digit. Techn.)*, vol. 138, no. 3, pp. 165–168, 1991.
[8] B. Ng, A. Si, R. W. H. Lau, and F. W. B. Li, "A multi-server architecture for distributed virtual walkthrough," in *Proc. ACM Symp. Virtual Reality Softw. Technol. (VRST)*, 2002, pp. 163–170.
[9] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM side—Channel(S)," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.*, 2002, pp. 29–45.
[10] M. L. Das, "Two-factor user authentication in wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 3, pp. 1086–1090, Mar. 2009.
[11] K. H. M. Wong, Y. Zheng, J. Cao, and S. Wang, "A dynamic user authentication scheme for wireless sensor networks," in *Proc. IEEE Int. Conf. Sensor Netw., Ubiquitous, Trustworthy Comput. (SUTC)*, vol. 1, Jun. 2006, p. 8.
[12] B. Vaidya, D. Makrakis, and H. Mouftah, "Two-factor mutual authentication with key agreement in wireless sensor networks," *Secur. Commun. Netw.*, vol. 9, no. 2, pp. 171–183, 2012.
[13] J. Kim, D. Lee, W. Jeon, Y. Lee, and D. Won, "Security analysis and improvements of two-factor mutual authentication with key agreement in wireless sensor networks," *Sensors*, vol. 14, no. 4, pp. 6443–6462, Apr. 2014.
[14] R. Amin, N. Kumar, G. Biswas, R. Iqbal, and V. Chang, "A light weight authentication protocol for IoT-enabled devices in distributed Cloud Computing environment," *Future Gener. Comput. Syst.*, vol. 78, pp. 1005–1019, Jan. 2018.
[15] D. Kang, J. Jung, H. Kim, Y. Lee, and D. Won, "Efficient and secure biometric-based user authenticated key agreement scheme with anonymity," *Secur. Commun. Netw.*, vol. 2018, pp. 1–14, Jun. 2018.
[16] G. Gaubatz, J. P. Kaps, and B. Sunar, "Public key cryptography in sensor networks—Revisited," in *Proc. Eur. Workshop Secur. Ad-Hoc Sensor Netw.*, 2004, pp. 2–18.
[17] E.-J. Yoon, E.-K. Ryu, and K.-Y. Yoo, "Efficient remote user authentication scheme based on generalized ElGamal signature scheme," *IEEE Trans. Consum. Electron.*, vol. 50, no. 2, pp. 568–570, May 2004.
[18] L. Wu, Y. Zhang, and F. Wang, "A new provably secure authentication and key agreement protocol for SIP using ECC," *Comput. Standards Interfaces*, vol. 31, no. 2, pp. 286–291, Feb. 2009.
[19] S. A. Chaudhry, H. Naqvi, M. Sher, M. S. Farash, and M. U. Hassan, "An improved and provably secure privacy preserving authentication protocol for SIP," *Peer–Peer Netw. Appl.*, vol. 10, no. 1, pp. 1–15, Jan. 2017.
[20] D. Dharminder and P. Gupta, "Security analysis and application of Chebyshev Chaotic map in the authentication protocols," *Int. J. Comput. Appl.*, vol. 43, no. 10, pp. 1095–1103, 2021.
[21] D. Dharminder, N. Kundu, and D. Mishra, "Construction of a chaotic map-based authentication protocol for TMIS," *J. Med. Syst.*, vol. 45, no. 8, p. 77, 2021.
[22] D. Dharminder, U. Kumar, and P. Gupta, "A construction of a conformal Chebyshev chaotic map based authentication protocol for healthcare telemedicine services," *Complex Intell. Syst.*, vol. 7, pp. 2531–2542, Jun. 2021.
[23] T. T. K. Hue, T. M. Hoang, and A. Braeken, "Lightweight signcryption scheme based on discrete Chebyshev maps," in *Proc. 12th Int. Conf. Internet Technol. Secured Trans. (ICITST)*, Dec. 2017, pp. 43–47.
[24] G. Gao, X. Peng, Y. Tian, and Z. Qin, "A chaotic maps-based authentication scheme for wireless body area networks," *Int. J. Distrib. Sensor Netw.*, vol. 12, no. 7, Jul. 2016, Art. no. 2174720.
[25] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. 29, no. 2, pp. 198–208, Mar. 1983.
[26] S. Roy, S. Chatterjee, A. K. Das, S. Chattopadhyay, S. Kumari, and M. Jo, "Chaotic map-based anonymous user authentication scheme with user biometrics and fuzzy extractor for crowdsourcing Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2884–2895, Aug. 2018.
[27] A. K. Das, P. Sharma, S. Chatterjee, and J. K. Sing, "A dynamic password-based user authentication scheme for hierarchical wireless sensor networks," *J. Netw. Comput. Appl.*, vol. 35, no. 5, pp. 1646–1656, 2012.
[28] J. Ryu, H. Lee, H. Kim, and D. Won, "Secure and efficient three-factor protocol for wireless sensor networks," *Sensors*, vol. 18, no. 12, p. 4481, Dec. 2018.
[29] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Examining smart-card security under the threat of power analysis attacks," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 541–552, May 2002.
[30] D. Wang, Q. Gu, H. Cheng, and P. Wang, "The request for better measurement: A comparative evaluation of two-factor authentication schemes," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, 2016, pp. 475–486.
[31] H. Lee, D. Lee, J. Moon, J. Jung, D. Kang, H. Kim, and D. Won, "An improved anonymous authentication scheme for roaming in ubiquitous networks," *PLoS ONE*, vol. 13, no. 3, Mar. 2018, Art. no. e0193366.
[32] B. Blanchet, "Automatic verification of security protocols in the symbolic model: The verifier proverif," in *Foundations of Security Analysis and Design VII*. 2013, pp. 54–87.
[33] L. Viganò, "Automated security protocol analysis with the AVISPA tool," *Electron. Notes Theor. Comput. Sci.*, vol. 155, pp. 61–86, May 2006.
[34] *AVISPA V1. 1 User Manual*, Information Society Technologies Programme, T. A. Team, 2006.

[35] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P. C. Heam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Vigano, and L. Vigneron, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Proc. Int. Conf. Comput. Aided Verification*, 2005, pp. 281–285.
[36] W. Dai. (2009). *Crypto++ 5.6. 0 Benchmarks*. [Online]. Available: http://www cryptopp com/benchmarks html
[37] G. Maze, "Algebraic methods for constructing one-way trapdoor functions," Univ. Notre Dame, Notre Dame, IN, USA, Tech. Rep., 2003.
[38] H. H. Kilinc and T. Yanik, "A survey of SIP authentication and key agreement schemes," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1005–1023, 2nd Quart. 2013.

JIHYEON RYU received the B.S. degree in mathematic and computer science from Sungkyunkwan University, South Korea, in 2018, where she is currently pursuing the Ph.D. degree with the Department of Software. Her current research interests include cyber security, machine learning, and user authentication.

DONGWOO KANG received the B.S. degree in electrical and computer engineering and the M.S. degree in electrical and computer engineering from Sungkyunkwan University, South Korea, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. His current research interests include cryptography, forensic, authentication protocol, and mobile security.

DONGHO WON received the B.S., M.S., and Ph.D. degrees in electronic engineering from Sungkyunkwan University, South Korea. After working with the Electronics and Telecommunication Research Institute for two years, he joined Sungkyunkwan University. He also worked as the President of the Korea Institute of Information Security and Cryptography. His research interests include cryptology and information security.