Research Article

Improved Chebyshev Polynomials-Based Authentication Scheme in Client-Server Environment

Toan-Thinh Truong,1 Minh-Triet Tran,1 and Anh-Duc Duong2

1Faculty of Information Technology, University of Science, VNU-HCM, 700000, Vietnam
2University of Information Technology, VNU-HCM, 700000, Vietnam

Correspondence should be addressed to Toan-Thinh Truong; ttthinh@fit.hcmus.edu.vn

Received 15 July 2018; Revised 10 December 2018; Accepted 20 December 2018; Published 17 January 2019

Guest Editor: Esther Palomar

Copyright © 2019 Toan-Thinh Truong et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With nonstop development of communication technologies, all aspects of social life continuously change and so do network systems. When establishing connection is easy, the convenience of online-service receives many users' attentions, for example, the patients directly access medical system to be advised by doctors at any time. Therefore, user authentication scheme is necessary when we want to provide privacy and security for working sessions. Storing a password list for verification is an old method and not secure. This list can be easily leaked, and adversary can launch an offline password-guessing attack. In addition, information exchanged between user and server needs being prevented from attacker's decryption. It can be said that current authentication schemes are unsuitable for new security standard. We need a strong user authentication scheme using new approach to overcome existing limitations and guarantee time efficiency. In this paper, we make a design with Chebyshev polynomial to achieve our goals and resist some kinds of attacks.

1. Introduction

User authentication is one of the first important parts in all remote services. Furthermore, after successful authentication, partners secretly exchange the messages to each other and we need a session key to encrypt all these messages. Therefore, authentication scheme also needs a session key agreement phase. Especially, when the wearable devices become popular, such as smart-glasses or smart-watch, a user wants to connect to remote service through these low-power computing devices. Therefore, in addition to security, also we consider the time efficiency which is one of the important factors. There are many proposed results using cryptography primitives to make a reasonable user authentication scheme. Lamport [1] is the pioneer using hash function with password. His method is a usage of password-table for user verification in login phase. This is a simple way and easily implemented, but his scheme is vulnerable to verification stolen attack, and inappropriately using password can result in offline password-guessing attack. Then, there are many proposed schemes to enhance security. Typically, in 2004 Das et al. [2] proposed dynamic identity to provide user anonymity in his scheme. This is a positive idea, but in his scheme, he uses password instead of real user's identity to create a dynamic login message. This causes their scheme cannot resist password-related attacks, and even the server may launch a password-guessing attack to find real user's password.

In 2006, Yoon et al. [3] proposed dynamic identity scheme using time-stamp. This scheme overcomes the reflection attack existing in Liao et al.'s scheme [4]. Clearly, Yoon's scheme has important improved ideas to isolate such problems. However, they also use password to authenticate with online server, so their scheme is still vulnerable to password-related attacks. Until now, password is still one of the most convenient factors in many authentication schemes, if only using this factor can be insecure. Using reasonable encryption scheme with block-cipher, such as Advance Encryption Standard (AES) or Triple Data Encryption Standard (T-DES), can enhance security for authentication scheme. Furthermore, if we only use hash function in scheme, this can increase authentication speed because time-cost of hash function is lower than the encryption scheme.
In addition to applying cryptography primitives, there is an approach using hard problems as security foundation, such as RSA or Elliptic curve crypto-systems (ECC). In 2009, Yang et al. [5] proposed a scheme in ECC. This is an efficient scheme because it uses discrete logarithm and Diffie-Hellman problems in elliptic curve. However, instead of using random values, they use point's coordinates to create a session key which does not satisfy perfect forward session key secrecy (PFS), one of the most standards to evaluate a strong authentication scheme. Therefore, some improved schemes were proposed, for instance, Islam et al. [6]. Their scheme used random values in creation of session key. However, his scheme is still vulnerable to known session-specific temporary information and denial of service attacks. In 2015 and 2016, Huang et al. [7] and Chaudhry et al. [8] proposed ECC-based authentication schemes, but these schemes cannot resist malicious user attack and does not provide PFS. Also, in 2015 Chaudhry et al. [9] proposed an authentication scheme in multiserver environment with general public key cryptography (RSA or ECC). However, their scheme needs a certificate agency (CA) to check the validity for the server’s key pairs. Furthermore, all previous schemes were proposed, for instance, Islam et al. [6]. Their approach many authors apply in authentication scheme lowers:

2.1. Chebyshev Chaotic Maps. Let \( n \in \mathbb{N} \) and \( x \in [-1, 1] \); we define Chebyshev polynomial \( T_0(x) \): \([-1, 1] \rightarrow [-1, 1] \) as \( T_n(x) = \cos(n \pi \arccos(x)) \). Its semigroup property is as follows:

\[
T_a(T_b(x)) = \cos(a \pi \arccos(b \pi \arccos(x)))
\]

\[
= \cos(b \pi \arccos(a \pi \arccos(x)))
\]

\[
= \cos(a \pi \arccos(b \pi \arccos(x)))
\]

\[
= \cos(b \pi \arccos(a \pi \arccos(x)))
\]

\[
= T_b(T_a(x))
\]

In 2008, Zhang [16] extended (1) to the interval \((-\infty, +\infty)\). Therefore, we have a different formula of Chebyshev polynomial as follows:

\[ T_0(x) = 1 \mod p \]

\[ T_1(x) = x \mod p \] (2)

\[ T_n(x) = 2 \times x \times T_{n-1}(x) - T_{n-2}(x) \mod p \]

where \( p \in \mathbb{P}, x \in [0, p-1] \) and \( n \in \mathbb{N} \). We see that (2) can be changed to

\[ T_n(x) \mod p = \frac{\lambda_1^n + \lambda_2^n}{2} \mod p \] (3)

2.2. The Hard Problems. In addition to four important properties, we have two computational problems on chaotic maps we apply in proposed user authentication scheme.

(i) The first problem is chaotic maps based discrete logarithm (CMDLP): Given \( y \in [0, p-1], p \in \mathbb{P}, \) and \( x \), it is hard to find \( r \) value such that \( T_r(x) = y \mod p \). We call this discrete logarithm problem on chaotic maps.

(ii) The second problem is chaotic maps based discrete logarithm (CMDHP): Given \( x \in [0, p-1], p \in \mathbb{P}, \) \( T_r(x) \mod p \), and \( T_s(x) \mod p \), it is hard to find
In this section, we review some typical related works applying Chebyshev chaotic map in user authentication schemes. Also, we analyse on their security.

3. Cryptanalysis of Typical Related Works

In this section, we review some typical related works applying Chebyshev chaotic map in user authentication schemes. Also, we analyse on their security.

3.1. Han-Yu Lin’s Scheme. Lin’s scheme [23] includes four phases: system initialization, user registration, authentication and password change phases.

(1) Initialization phase. The server S chooses all necessary parameters \((r, x, T_r(x), h, E_s)\). Especially \((x, T_r(x))\) is written into user’s smartcard.

(2) Registration phase. The user U chooses identity ID, password PW, and random value t, then computes \(H = h(PW, t)\), and sends \([ID, H]\) to S through a secure channel. Once receiving U’s messages, S checks ID’s validity and uses master key s to compute \(R = E_s(ID, H)\). \(D = H \oplus (x \parallel T_r(x))\). Finally, S sends \([R, h, E_s, D]\) to U through a secure channel. U receives S’s incoming smartcard and inserts t into it.

(3) Authentication phase. When U authenticates with S, U provides \((ID, PW)\) and smartcard into the terminal. Below are some steps for authentication (see Figure 1):

(a) Smartcard chooses \(j\) and computes \((x \parallel T_r(x)) = h(PW, t) \oplus D\). \(v = T_j(T_r(x))\) and \(Q = h(ID, H)\).

(b) Next, U sends \((T_j(x), E_s(Q, R, T_1))\) to S, where \(T_1\) is receiving time-stamp.

(c) Once receiving U’s messages, S computes \(v = T_j(T_r(x))\) and \(E_s(Q, R, T_1)\) and then checks \(T_1\).

(d) Next, S decrypts \(R\) with \(s\) to recover \((IDt, Ht)\) and computes \(Qt = h(IDt, Ht)\). If \(Q = Qt\) then S successfully authenticates with U. Otherwise, S terminates the session.

(e) Then, S chooses \(j\) and sends \(E_s(T_j(x), h(ID, T_2), T_2)\) to U, where \(T_2\) is time-stamp when S sends the message to U.

(f) Once receiving S’s message, U decrypts and checks \(T_2\)’s validity. At the same time, U computes \(ht(ID, T_2)\). U checks the validity of \(ht(ID, T_2)\). If this condition holds, U successfully authenticates with S; otherwise U terminates the session.

(g) After successfully authentication phase, both U and S compute session key \(\lambda = T_j(T_r(x))\) for later usage.

(4) Password change phase. U provides smartcard, old PW and new PW*. Then, smartcard randomly chooses \(i\) and computes \(Ht = h(PW, t)\). \((x \parallel T_r(x)) = Ht \oplus D\), \(\eta = T_j(T_r(x))\) and \(H^* = h(PW^*, t)\) and then sends \([T_j(x), E_s(ID, H^*, R)]\) to S. Once receiving U’s messages, S computes \(\eta = T_j(T_r(x))\) and decrypts \(E_s(Ht, H^*, R)\) with \(\eta\) and \(s\). Finally, S compares \(Ht\). If this holds, S returns \(R^* = E_s(ID, H^*)\) to smartcard, then it updates \(R = R^*\).

3.2. Security Analysis on Han-Yu Lin’s Scheme. In this subsection, we also review some limitations existing in this scheme.

(i) In this phase, \((x \parallel T_r(x))\) is the same in all users’ smartcard. Therefore, malicious user can exploit this to launch an offline password-guessing attack if another user’s smartcard is lost. Suppose malicious user extracts \([R, h, E_s, D, t]\) of another user. Then, malicious user computes \(H\) by performing \(H = D \oplus (x \parallel T_r(x))\), where \((x \parallel T_r(x))\) belongs to malicious user.
user’s smartcard. With $H$, malicious user builds $H := h(PW, t)$ and Han-Yu Lin’s scheme is vulnerable to this kind of attack.

(ii) Also, Han-Yu Lin’s scheme is vulnerable to contributory property of key agreement. In this scheme, $S$ can determine common session key without $U$’s random value. Below are some steps $S$ can perform:

1. Find $j_t$ such that $T_j(x) = T_j(x)$, where $j_t = (cos^{-1}(T_j(x)) + k2\pi)/cos^{-1}(x) | k \in \mathbb{Z}$.
2. Then, $S$ chooses session key $\lambda I$ and computes $\nu = (cos^{-1}(\lambda I) + k2\pi)/(j_t \times cos^{-1}(x)) | k \in \mathbb{Z}$.
3. $S$ has $\lambda' = T_j(T_j(x))$ and transmits $T_j(x)$ to $U$.
4. When $U$ receives $T_j(x)$, $U$ computes $T_j(T_j(x)) = T_j(T_j(x)) = \lambda$. So, Lin’s scheme is vulnerable to this property.

3.3. Hongfeng Zhu’s Scheme. Zhu’s scheme [24] includes four phases: registration, login, authentication, and password change phases.

(1) **Registration phase.** In this phase, the user $U$ chooses password $PW_u$, then randomly chooses value $t$, and computes $W_u = h(PW_u || t)$. Next, $U$ sends $(I_{D_1}, W_u)$ to the server $S$ through a secure channel. After receiving $U$’s request $(I_{D_1}, W_u)$, $S$ computes $H_s = h(I_{D_1} || D_1)$, $n_s = h(W_u || I_{D_1} \oplus H_s)$, and sends $\{n_s, x, T_x(x)\}$ to $U$ through a secure channel. Once receiving $S$’s messages, $U$ computes $N_s = h(I_{D_2} || PW_u) \oplus n_s \oplus h(W_u || I_{D_2} || PW_u) \oplus H_s$. Finally, $U$ saves $\{N_s, x, T_x(x)\}$ into $U$’s device.

(2) **Login-authentication phases** (see Figure 2). In this phase, $U$ inputs $(I_{D_2}, PW_2)$ and $U$’s device randomly chooses two values $k, R$ to compute $H_k = N_s \oplus h(I_{D_2} || PW_2)$, $T_k(x) = T_k(x)$, $K_{SA} = T_k(T_k(x), H) = h(H_k || I_{D_1} || ID_2 || ID_1 || R)$. Then, $U$ sends $\{C, T_k(x)\}$ to $S$. After receiving $U$’s messages, $S$ computes $K_{SA} = T_k(T_k(x))$. Figure 2: Hongfeng Zhu’s authentication scheme.

and master key $s$. Then, $S$ decrypts $C$ to recover $H_k || ID_2 || ID_1 || R$. $S$ computes $H_k = h(ID_2 || PW_2)$, $T_k(x)$, $K_{SA} = T_k(T_k(x), H) = h(H_k || I_{D_1} || ID_1 || ID_1 || R)$. Finally, $S$ checks $H_k || ID_2 || ID_1 || R$. If this holds, $S$ randomly chooses $r$ and computes $V_r = h(h(r || r))$. $V_r$ and $H_k || ID_2 || ID_1 || R$ and sends $\{V_r, V_2\}$ to $U$. Otherwise, $S$ terminates the session. Once receiving $S$’s $\{V_r, V_2\}$, $U$’s device checks $V_2 = h(V_2 || H_k)$. If this does not hold, $U$’s device terminates the session, otherwise it computes $SK = h(ID_2 || ID_2 || R || h(r || r))$.

(3) **Password change phase.** In this phase, $U$ provides old $PW_u$, new $PW_u'$, and $ID$. Then, the device chooses random value $kt$ and computes $H_k = N_s \oplus h(ID_2 || PW_2)$, $N_s' = N_s \oplus h(ID_2 || PW_2')$, $T_k(x)$, $K_{SA} = T_k(T_k(x), H) = h(H_k || I_{D_1} || ID_2 || ID_1 || R)$. Then, $S$ checks $H_k || ID_2 || ID_1 || R$. If this does not hold, $S$ rejects. Otherwise, password-update is accepted and device computes $V_2 = h(K_{SA} || response)$, where response = “update” or “refuse”. Finally, $S$ returns $V_2$ to $U$. Once $U$’s device receives $\{V_2, response\}$, it checks $V_2 = h(K_{SA} || response)$. If this holds, $U$ updates $\{N_s' \oplus N_s, x, T_x(x)\}$ into $U$’s device.

3.4. Security Analysis on Hongfeng Zhu’s Scheme. Next, we also review some limitations existing in this scheme.

(i) Hongfeng Zhu’s scheme does not provide PFS. If important information such as master key $s$ is leaked, $U$’s session key will be easily computed with previous exchanged messages. Suppose an adversary captures $\{C, T_k(x)\}$ and $\{V_1, V_2\}$ at another session between $U$ and $S$. With $s$, the adversary computes
4. Proposed Scheme

Our proposed scheme using Chebyshev polynomial includes five phases: initialization, registration, authentication, and biometrics update phases. Below are some notations used in our scheme:

(i) \( U_i \): the \( i \)th user
(ii) \( ID_i \): the \( i \)th user’s identification
(iii) \( B_i \): the \( i \)th user’s biometrics
(iv) \( S \): the server
(v) \( q_S \): the server’s master key
(vi) \( h(.) \): hash function
(vii) \( sk \): common session key
(viii) \( SC \): the smartcard
(ix) \( \oplus, \| \): the XOR and concatenation operations
(x) \( T(.) \): Chebyshev polynomial operation

\[
K_{SA} = T_s T_k(x) \text{ and decrypts } D_{K_{SA}}(C) = H_A \| ID_A \| ID_B \| R. \text{ Next, he/she computes } H_A = h(s \| ID_A) \text{ and extracts } h(R \| r) = V_2 \oplus H_A. \text{ Finally, he/she computes } SK = h(ID_A, B_i, R \| h(R \| r)). \text{ Clearly, this scheme does not satisfy PFS.}
\]

(ii) This scheme does not store password-confirmation message at \( U \)’s device, so password-update phase must connect to \( S \). However, adding a value in \( U \)’s device helps login phase be more secure if the device is stolen. For example, we add \( dev_i \) to the process. When logging, \( U \) inputs \( ID_A \) and \( PW_A \). \( U \)’s device computes \( H_A = N_a \oplus h(ID_A, PW_A) \) and then checks \( L = h(ID_A, PW_A) \oplus H_A \). If this holds, \( U \) is real device’s owner. With \( L \), this scheme can resist offline password-guessing attack if device’s information is leaked because \( L \) contains authentication key \( H_A = h(s \| ID_A) \). In password change phase, \( U \) needs correcting old \( PW \) to pass \( L = h(ID_A, PW_A) \oplus H_A \). If this holds, \( U \)’s device will accept new \( PW \) intended by \( U \). Next, \( U \)’s device recomputes \( N'_a = N_a \oplus h(ID_A, PW_A) \) and \( L = h(ID_A, PW_A, N'_a) \). Clearly, password change phase is more efficient than previous old phase.

4.1. Initialization Phase. In this phase, we choose a huge prime number \( k \)-bit \( p \). Then \( S \) chooses \( q_S \in [1, p - 1] \) and \( h: \{0, 1\}^* \rightarrow \{0, 1\}^k \). Finally, \( S \) publishes \( \{p, T(.), h(.)\} \)

4.2. Registration Phase. \( U_i \) provides \( B_i \) and \( ID_i \). Also, \( U_i \) randomly chooses \( N \). Then, \( U_i \) computes \( B_i = h(N \| B_i) \) and \( V_i = h(B_i \| N) \). Finally, \( U_i \) sends \( \{hB_i, ID_i\} \) to \( S \) through a secure channel. On receiving the \( U_i \’s \) information, \( S \) checks \( ID_i \’s \) validity. Then, \( S \) randomly chooses \( X_i \). \( S \) computes \( hAID_i = T_{q_S}(h(ID_i \| X_i)) \) and \( mod p + T_{X_i}(h(ID_i \| hB_i)) \mod p \), then \( S \) sends \( \{hAID_i, X_i\} \) to \( U_i \) through a secure channel. \( U_i \) stores the information sent from \( S \) into a SC (see Figure 3).

4.3. Authentication Phase. \( U_i \) provides \( B_i \) and \( ID_i \) at the terminal. Then \( S \) checks if \( V_i \) is stolen. For example, \( U_i \) adds \( dev_i \) to \( X_i \) and checks if \( ID_i \’s \) device computes \( hB_i \) and \( h(ID_i, B_i) \) mod \( p \). \( SC \) chooses \( r_i \in [1, p - 1] \) and computes \( R' = T_r(h(ID_i \| X_i)) \) mod \( p \). If this holds, \( S \) sends \( \{hB_i, ID_i\} \) to \( U_i \) through a common channel (see Figure 4).

When receiving \( U_i \’s \) login message, \( S \) computes \( R'_i \) and \( AID_i^* \), where \( R'_i = T_{q_S}(R') \) mod \( p \). \( ID_i = CIP \oplus h(R'_i) \) and \( AID_i^* = T_{q_S}(h(ID_i \| X_i)) \) mod \( p \). Then \( S \) checks if \( M_i = h(R'_i, AID_i^*) \). If this does not hold, \( S \) terminates the session. Otherwise, \( S \) chooses \( r_i \in [1, p - 1] \) and computes \( R_S = T_s(R'_i) \) mod \( p \), \( S' = R_S + R'_i \), \( sk = h(T_r(R'_i)) \) and \( M_S = h(R_S, AID_i^*) \). \( S \) sends \( \{S', M_S\} \) to \( U_i \) through a common channel.

When receiving \( \{S', M_S\} \), \( U_i \’s \) SC computes \( R'_i = S' - R_i \) and checks if \( M_S = h(R'_i, AID_i^*) \). If this holds, \( S \) computes \( sk = h(T_r(R'_i)) \) and \( M_{US} = h(R_S, T_r(R'_i)) \). Then, \( S \) sends \( \{M_{US}\} \) to \( S \) through a common channel.

When receiving \( \{M_{US}, M_{US}\} \), \( S \) accepts \( U_i \), \( U_i \) and \( S \) use \( sk \) to encrypt the information after authentication phase.

4.4. Biometrics Update Phase. When \( U_i \) changes his/her biometrics, \( U_i \’s \) SC checks if \( V_i = h(B_i \| N) \). If this holds, \( S \) computes \( V_{new} = h(B_{new} \| N) \) and \( hAID_{new} = T_{X_i}(h(ID_i \| hB_{new})) \) mod \( p \). Finally, \( SC \) replaces \( V_i \) and \( hAID_i \) with \( V_{new} \) and \( hAID_{new} \).
5. Security and Efficiency Analyses

In this section, we analyse our scheme on security and efficiency aspects. Also, our scheme's design is correctly proved with BAN-logic [25], while its security is presented in each concrete attack case.

5.1. Correctness Analysis. Before getting into details about security, we will prove our scheme's correctness with BAN-logic. We inherit some objectives from [26] because we see that they are reasonable ones, which authentication scheme must achieve to successfully share partner's identities and session keys. For simplicity, we let $K$ denote user's long-term key, $sk$ denote session key, and $\otimes$ denote Chebyshev operation. Firstly, our scheme must satisfy some assumptions as shown in Table 1 (this is a must in this model)

These assumptions represent the first necessary believes of user and server. For example, when the users register with server, it is mean that they believe they can share identity with server ($A_1$). Next, we will normalize all messages exchanged between user and server.

(i) From the message $\{CID\}$ we have $<U^{ID}S, U^{K}S, r_U \otimes K>$

(ii) From the message $\{M_j\}$ we have $<r_U \otimes K, U^{K}S>$

(iii) From the third messages $\{M_{US}\}$ we have $<r_S \otimes K, U^{K}S>$

(iv) From the fourth message $\{M_{US}\}$ we have $<U^{K}S, U^{sk}S>$

The normalization is an arrangement of information exchanged between user and server. For example, $CID$ contains identity, challenge information $r_U \otimes K$ and long-term key $K$. Normalization helps to highlight the important data in the messages. Next, we will demonstrate how our scheme satisfies seven lemmas that we reorganized from [32].

Lemma 1. If the server believes authentication key (long-term key) is successfully shared with user and the user's messages encrypted with this key are fresh, the server will believe that the user believes his/her identity is successfully shared with server.

\[
S \Vdash (S^{K}U), S \Vdash \# \langle r_U \otimes K \rangle
\]

\[S \Vdash \left( U \Vdash (U^{ID}S) \right) \]

\[(4)\]
Proof. With \( A_6 \) and CID, we apply message-meaning rule to have

\[
S \vdash S \xleftrightarrow[\#]{K} U, S \xleftrightarrow[\langle\rangle]{ID} S, U \xleftrightarrow[\#]{K} S, r_U \otimes K >
\]

\[
S \vdash U \xleftrightarrow[\langle\rangle]{ID} S, U \xleftrightarrow[\#]{K} S, r_U \otimes K >
\]

(5)

With (5), we apply nonce-verification rule to have

\[
S \vdash U \xleftrightarrow[\langle\rangle]{ID} S, U \xleftrightarrow[\#]{K} S, r_U \otimes K >
\]

(6)

With (12) and (13), we apply nonce-verification rule to have

\[
U \xleftrightarrow[\langle\rangle]{K} S, U \xleftrightarrow[\#]{K} S
\]

(14)

With (14), we apply believe rule to have

\[
U \xleftrightarrow[\langle\rangle]{K} S, U \xleftrightarrow[\#]{K} S
\]

(15)

So, with \( A_4 \) and \( A_6 \), we successfully demonstrate how our scheme satisfies Lemma 2.

Lemma 2. If the server believes the user also believes his/her identity is successfully shared with each other and user totally controls this identity's sharing, the server also believes user's identity is successfully shared with each other.

\[
S \vdash (U \xleftrightarrow[\langle\rangle]{K} S), S \vdash (U \xleftrightarrow[\#]{K} S)
\]

(9)

Proof. With Lemma 1 and \( A_4 \), we apply jurisdiction rule to have

\[
S \vdash (U \xleftrightarrow[\langle\rangle]{K} S), S \vdash (U \xleftrightarrow[\#]{K} S)
\]

(10)

So, with Lemma 1 and \( A_4 \), we successfully demonstrate how our scheme satisfies Lemma 2.

Lemma 3. If the user believes authentication key is successfully shared with server and the server's messages encrypted with this key are fresh, the user will believe the server also believes session key is successfully shared with each other.

\[
U \xleftrightarrow[\langle\rangle]{K} S, U \xleftrightarrow[\#]{K} S
\]

(16)

Proof. With \( A_2 \) and \( M_{US} \), we apply message-meaning rule to have

\[
U \xleftrightarrow[\langle\rangle]{K} S, U \xleftrightarrow[\#]{K} S
\]

(17)

With \( A_7 \) and \( M_{US} \), we apply freshness rule to have

\[
U \xleftrightarrow[\#]{K} S, U \xleftrightarrow[\#]{K} S
\]

(18)

With (17) and (18), we apply believe rule to have

\[
U \xleftrightarrow[\#]{K} S, U \xleftrightarrow[\#]{K} S
\]

(19)
With (19), we apply believe rule to have
\[
\frac{U \models S \equiv U \xleftarrow{\text{SK}} S, U \xleftarrow{\text{sk}} S >}{U \models S \equiv S \xleftarrow{\text{sk}} U}
\]  
(20)

So, with \(A_2\) and \(A_3\) we successfully demonstrate how our scheme satisfies Lemma 4.

Lemma 5. If the user believes the server totally controls session key's sharing and the server also believes session key is successfully shared with user, the user will believe this session key's sharing.

\[
U \models \left( S \Rightarrow \left( U \xleftarrow{\text{SK}} S \right) \right), U \models \left( S \equiv \left( S \xleftarrow{\text{sk}} U \right) \right)
\]  
(21)

Proof. With \(A_3\) and Lemma 4, we apply jurisdiction rule to have
\[
U \models S \Rightarrow U \xleftarrow{\text{sk}} S, U \models S \equiv S \xleftarrow{\text{sk}} U
\]  
(22)

So, with \(A_2\) and Lemma 4, we successfully demonstrate how our scheme satisfies Lemma 5.

Lemma 6. If the server believes authentication key is successfully shared with user and the user's messages encrypted with this key are fresh, the server will believe the user also believes this session key's sharing.

\[
S \models \left( S \xleftarrow{\text{K}} U \right), S \models \left( S \xleftarrow{\text{sk}} U \right)
\]  
(23)

Proof. With \(A_6\) and \(M_{\text{US}}\), we apply message-meaning rule to have
\[
S \models S \xleftarrow{\text{K}} U, S \xleftarrow{\text{sk}} U \xleftarrow{\text{sk}} S >
\]  
(24)

With \(A_6\) and \(M_{\text{US}}\), we apply freshness rule to have
\[
S \models \# \left( r_U \otimes AID \right), S \xleftarrow{\text{sk}} U \xleftarrow{\text{sk}} S >
\]  
(25)

With (24) and (25), we apply nonce-verification to have
\[
S \models U \xleftarrow{\text{K}} S, U \xleftarrow{\text{sk}} S >, S \models \# \left( U \xleftarrow{\text{K}} S, U \xleftarrow{\text{sk}} S >
\]  
(26)

With \(A_6\) and (26), we apply believe rule to have
\[
S \models S \xleftarrow{\text{K}} U, S \equiv U \xleftarrow{\text{sk}} S
\]  
(27)

So, with \(A_6\) and \(A_8\), we successfully demonstrate how our scheme satisfies Lemma 6.

Lemma 7. If the server believes the user totally controls the session key's sharing, the server will believe the session key is successfully shared with user.

\[
S \equiv \left( U \Rightarrow \left( U \xleftarrow{\text{SK}} S \right) \right)
\]  
(28)

Proof. With (26) and \(A_3\), we apply message-meaning rule to have
\[
S \equiv U \xleftarrow{\text{sk}} S, U \equiv U \xleftarrow{\text{K}} S, U \xleftarrow{\text{sk}} S >
\]  
(29)

With (29), we apply believe rule to have
\[
S \equiv S \xleftarrow{\text{sk}} U
\]  
(30)

So, with \(A_3\) we completely demonstrate how our scheme satisfies Lemma 7. Finally, we can say that both server and user believe the common session key in our scheme.

5.2. Security Analysis. Before getting into details about some kinds of attacks, we will use random oracle model to prove the security for the session key in Chebyshev polynomial case (see [27, 28] for more details). At first, we need to remind the model's circumstance. Assuming another actor \(B\) has \(\Omega = \{ T_p(x), T_q(x) \}\) and \(B\) needs to compute \(T_{pxq}(x)\), \(B\) has some oracles Client and Server with all their instances at different times. \(B\) also has an algorithm \(A\) being able to break our scheme to compute the session key with given probability \(\varepsilon\). \(B\) will use \(A\) to find the session key and then compute \(T_{pxq}(x)\) to solve CMDHP. To achieve this, \(B\) must “inject” \(\Omega\)’s parameters into the messages when \(A\) interacts with the oracles' instances, and \(B\) also simulates an appropriate environment suitable for \(A\) to operate. Note that our scheme uses hash function and it is considered as an oracle. Next, we claim our theorem about the session key’s security.

Theorem 8. Let \(A\) be an adversary breaking our scheme in the meaning of AKESecurity in time \(t_A\), using \(q_{\text{Send}}\) Send queries and \(q_{\text{Hash}}\) Hash queries. We have
\[
\text{Adv}_{\text{p}}^{\text{AKE}} \left( A, t_A, q_{\text{Send}}, q_{\text{Hash}} \right) \leq q_{\text{Hash}} \times q_{\text{Send}}
\]  
(31)

\[
\times \text{Adv}_{\text{ECMDHP}}^{\text{CMDHP}} \left( B, t_B, q_{\text{Send}}, q_{\text{Hash}} \right)
\]

where \(B\) is an adversary breaking CMDHP in \(t_B\). The meaning of theorem is that \(A\)'s successful probability breaking our scheme in the meaning of AKESecurity is less than \(B\)'s successful probability breaking CMDHP. According to CMDHP, \(B\)'s success probability is extremely low, and so is \(A\)'s successful probability breaking our scheme. Therefore, we can claim that our scheme has secure session key in the meaning of AKESecurity.
Proof. Assume that we have actor B. This time, B needs to create some instances of oracles Client and Server with \( \Omega \)'s parameter. B also simulates an appropriate environment where A can operate.

When A sends Send("Start") to \( U_i^\text{Client} \), \( U_i^\text{Client} \) replies \( m_1 \) to A (note that \( m_1 = \{ \text{identity encrypted, challenge information, confirmation} \} \) depends on concrete scheme). Maybe, A sends Send("Start") to some simulated oracle, for example, \( U_i^\text{Client} \). This time, B needs to inject \( \Omega \)'s parameters into \( m_1 \) with concrete scheme's rules. Finally, B sends \( m_1 \) to A. When A sends Send\( (m_1) \) to \( S_i^\text{Server} \), \( S_i^\text{Server} \) replies \( m_2 \) to A (note that \( m_2 = \{ \text{confirmation, challenge information} \} \) depends on concrete scheme). Maybe, A sends Send\( (m_2) \) to simulated oracle \( S_i^\text{Server} \). This time, B also needs to inject \( \Omega \)'s parameters into \( m_2 \) and sends \( m_2 \) to A. When A sends Send\( (m_2) \) to \( S_i^\text{Server} \), \( S_i^\text{Server} \) replies \( m_3 = \{ \text{final confirmation} \} \) to A. Maybe, A sends Send\( (m_3) \) to simulated oracle \( S_i^\text{Server} \). B randomly chooses \( s_k \in \{ 0, 1 \}^n \), computes \( \text{final confirmation message} \) with this random \( s_k \), and sends \( m_3 \) to A.

Sometimes, A sends wrong Send\( (\)queries to the instances, so there are some oracles with “Accept” state and some oracles with “Reject” state. However, when A finishes sending \( q_{\text{Send}} \) Send\( (\)queries, all instances A interacts with must have “Terminated” state which is true.

When A sends Corrupt and Reveal\( (\)queries to these instances, their state will determine what A obtains, such as long-term key or session key. When A sends Corrupt and Reveal\( (\)queries to the oracles B simulates, B will generate a random string representing session key for them. When A sends hash\( (\)queries, B will let hash oracle interact with A.

Finally, B activates A to sends a unique Test\( (\)query to simulated oracle or indicated oracle, and B expects A to correctly guess bit b of this instance. In other words, B wants A to correct guess this instance's session key with A's successful probability. We see that if B is success, B needs three following consecutive factors:

(i) B needs A to correctly find \( s_k \) of simulated oracle, and A's successful probability is \( \epsilon = \text{Ad}_{\mathcal{F}}(A, t_A) \).

(ii) Furthermore, when A correctly finds \( s_k = h(T_{\text{psq}}(x) \ldots) \), A had found \( T_{\text{psq}}(x) \) satisfying with this \( s_k \). Clearly, if A sends \( q_{\text{Hash}} \) queries to hash oracle, there is at least one-time A succeeds. So, \( \gamma \geq 1/q_{\text{Hash}} \). (Note that: the session key \( s_k \) will be always computed with hash function.)

(iii) Next, when A correctly finds \( T_{\text{psq}}(x) \), A must correct guess \( q \) or \( s \). Clearly, if A sends \( q_{\text{Send}} \) queries to the oracle, there is at least one-time A succeeds. So, \( \mu \geq 1/q_{\text{Send}} \).

Finally, we have \( \text{Ad}_{\mathcal{F}}(A, t_A) = \epsilon \times \gamma \times \mu \geq \epsilon \times 1/(q_{\text{Hash}} \times q_{\text{Send}}) \equiv q_{\text{Send}} \times q_{\text{Hash}}/\ast \ast \text{Ad}_{\mathcal{F}}(A, t_A) \geq \epsilon = \text{Ad}_{\mathcal{F}}(A, t_A, q_{\text{Send}} \times q_{\text{Hash}}) \).

In this subsection, we analyse our scheme on security aspect (see Table 2).

(1) **Password-guessing attack.** If the smartcard's information is leaked, and the adversary can exploit to perform password-guessing attack. Therefore, the adversary has \( V_i = h(B_i \parallel N) \) in the smartcard. Differently from password, \( B_i \) is the user's biometrics and it cannot be predicted. In short, our scheme easily resists this kind of attack.

(2) **Replay attack.** In this kind of attack, the adversary can replay the login message to impersonate the user. In our scheme, the adversary can replay \( [X_i, CID, M_i, R_i] \) to the server. Then, the server replies \( [ST, M_S] \) to the adversary. At this time, the adversary cannot compute \( M_{US} \) because \( R_i = T_{r_i}(AID_i) \) mod p is impossible to know. Therefore, our scheme can resist this kind of attack.

(3) **User anonymity.** In this kind of attack, the adversary eavesdrops \( [X_i, CID, M_i, R_i], [ST, M_S] \) and \( [M_{US}] \) of another user. The user's identity is encrypted with \( R_i \), which includes the secret \( AID_i \). Therefore, the adversary cannot trace who is authenticating and our scheme provides user anonymity.

(4) **Impersonation attack.** In this kind of attack, the adversary can impersonate either user or server. In our scheme, the adversary eavesdrops \( [X_i, CID, M_i, R_i] \) and \( [ST, M_S] \). However, he must send \( M_{US} \) to cheat the server and this is impossible because \( r_i \) and \( AID_i \) are secret. Moreover, if he wants to impersonate the server, he needs to compute \( M_S \) and this is impossible because \( AID_i \) is secret. Therefore, our scheme can resist this kind of attack.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Password-guessing</td>
<td>x</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Replay</td>
<td>( \emptyset )</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>User anonymity</td>
<td>v</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>Impersonization</td>
<td>v</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Man-in-the-middle</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>Parallel</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>Two-factor</td>
<td>x</td>
<td>v</td>
<td>v</td>
</tr>
<tr>
<td>Perfect secrecy session key</td>
<td>( \emptyset )</td>
<td>( x )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

1. The authors do not consider.
(5) **Man-in-the-middle attack.** In this kind of attack, the adversary can cheat both user and server simultaneously. However, he must compute random values \( r_i \) and \( AID_i \). From this information, he can derive \( R_i \) to cheat the server and derive \( M_{US} \) to cheat the user. Clearly, this information is secret, and the adversary cannot steal them.

(6) **Parallel attack.** In this kind of attack, the adversary uses another session's messages to exploit the others. In our scheme, this is impossible because each session has different random values. For example, another session has the unique values \( r_i \) and \( r_s \), so all sessions have no relationship with each other.

(7) **Two-factor attack.** In this kind of attack, the adversary can steal the user's biometrics, and then use this information to compute authentication key. We see that the smartcard includes \{ \( N, V_i, hAID_i, X_j \) \}, so if there is no \( B_i \), the adversary cannot compute \( AID_i \). Of course, if the smartcard is well-protected, the adversary has no way to compute \( AID_i \). Our scheme can resist this kind of attack.

(8) **Perfect secrecy.** In this kind of attack, the adversary has all secret keys of the users and the server. Of course, the service must be stopped at this time. However, we need to prevent the adversary from knowing past-transactions, and this means that all session keys must be secret. In our scheme, the session key is constructed from \( r_i, r_s \), and \( AID_i \). Clearly, if the adversary knows \( T_i(AID_i) \) and \( T_s(AID_i) \), he cannot compute \( T_i(T_s(AID_i)) \) because of facing with CMDHP.

### 5.3. Efficiency Analysis

To compare efficiency between our scheme and previous ones, we let "h" be the hash operation, "e/d" be the encryption/decryption, and "T" be computational operation of polynomial. At registration phase, our scheme uses \( 4 \times h + 1 \times T \) and \( 1 \times e/d \). Lin's scheme uses \( 1 \times h, 1 \times T \), and \( 1 \times e/d \). Zhu's scheme uses \( 5 \times h \). At authentication phase, our scheme uses \( 14 \times h + 8 \times T \). Lin's scheme uses \( 4 \times h, 5 \times T \), and \( 5 \times e/d \). Zhu's scheme uses \( 9 \times h, 3 \times T \), and \( 2 \times e/d \). Our scheme's computational cost is more than previous ones due to security enhancement (see Table 3).

Also, we let \( t_h, t_T, \) and \( t_{e/d} \) denote running-time corresponding to each operation, for example, \( h, T, \) and \( e/d \) (\( t_h \ll t_{e/d} < t_T \)). To relatively measure the running-time of three operations, we conduct an experiment using Java Cryptography Architecture with Bouncy Castle library in Android mobile device, core 4 CPU 1.2 GHz, and we have \( t_h \approx 0.00004 ms, t_{e/d} \approx 0.09385 ms, \) and \( t_T \approx 80 ms \) (see Figure 5).

### Table 3: The efficiency comparison among the schemes.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>( 1 \times h + 1 \times T + 1 \times e/d )</td>
<td>( 5 \times h )</td>
<td>( 4 \times h + 2 \times T )</td>
</tr>
<tr>
<td>Authentication</td>
<td>( 4 \times h + 5 \times T + 5 \times e/d )</td>
<td>( 9 \times h + 3 \times T + 2 \times e/d )</td>
<td>( 14 \times h + 8 \times T )</td>
</tr>
</tbody>
</table>

### Figure 5: The running-time of three schemes.

### 6. Conclusion

This paper proposes a Chebyshev polynomials-based scheme in client-server environment. Although, our scheme takes more time than previous ones, it is advanced and resists some popular kinds of attack. Soon, we improve some techniques to reduce time-cost for computing Chebyshev polynomials.

### Conflicts of Interest

We declare that there are no conflicts of interest regarding the publication of this paper.

### Acknowledgments

This research is funded by Vietnam National University HoChiMinh City (VNU-HCM) under Grant no. B2015-18-01.

### References


