Security of a key agreement protocol based on chaotic maps

Song Han *

Curtin University of Technology
GPO Box U1987 Perth, WA 6845, Australia

Abstract

Kacorev et al. proposed new public key encryption scheme using chaotic maps. Subsequently, Bergamo et al. has broken Kacorev and Tasev's encryption scheme and then applied the attack on a key agreement protocol based on Kacorev et al.'s system. In order to address Bergamo et al.' attack, Xiao et al. proposed a novel key agreement protocol. In this paper, we will present two attacks on Xiao et al.'s key agreement protocol using chaotic maps. Our new attack method is different from the one that Bergamo et al. developed. The proposed attacks work in a way that an adversary can prevent the user and the server from establishing a shared session key even though the adversary cannot get any private information from the user and the server's communications.

Key words: Chaos, chaotic map, secure communication, shared session key

1 Introduction

Chaos is a universal, random-like and robust phenomenon in nonlinear systems. Chaotic systems are characterized by sensitive dependence on initial conditions and similarity to random behavior. Chaotic systems have been used to design and analyze secure communications [3-6, 13]. There are two main methods in the design of chaotic secure communication schemes: one is analog method, the other is discrete digital method. In the analog method, chaos synchronization is achieved using chaotic circuits, where the communication parties take advantage of the chaotic synchronization to implement secure

* Corresponding.
Email address: s.han@curtin.edu.au (Song Han).

Preprint submitted to Chaos, Solitons & Fractals (Accepted for publication)

communication [3, 6, 12]. In the discrete digital method, chaotic systems are used to generate chaotic ciphers for secure communication [4].

A key agreement protocol is used to derive a shared secret by two or more parties as a function of information contributed by, or associated with, each of these, but no party can predetermine the resulting value [8]. Xiao et al. proposed a key agreement protocol based on chaotic maps [9], which was proved to be broken by Bergamo et al.'s method [1, 2]. In Bergamo et al.'s method, an adversary can retrieve the private information from the communication parties. Recently, a new key agreement protocol was proposed based on chaotic maps [11]. This new protocol has greatly enhanced the security against Bergamo et al.'s compromise method.

In this paper, we will provide a security analysis of Xiao et al.'s key agreement protocol based on chaotic maps. We will develop new attacks on their protocol. We will demonstrate that their protocol is not secure against these attacks. In these attacks, an adversary can successfully prevent communication parties from establishing a secret shared session key.

2 Xiao et al.'s key agreement protocol

Xiao et al.'s key agreement protocol uses the one-way hash function based on Chebyshev chaotic maps. The details of the key agreement protocol are:

- (1) User A chooses a random number $ra \in [-1, 1]$, then he sends his user identity number, ID_A, and ra to server B.

- (2) Server B chooses a random number $rb \in [-1, 1]$ and sends it back to user A.

- (3) User A juxtaposes h_{PW}, ra and rb from left to right as the pending message, and uses the one-way hash function $H(\cdot)$ to compute the authentication value $AU = H(h_{PW}, ra, rb)$. Then he sends ID_A and AU to server B.

- (4) Server B takes out his own copies of h_{PW}, ra and rb corresponding to the user identity number ID_A. Then she uses the same Hash function to compute $AU' = H(h_{PW}, ra, rb)$ similarly.

- (5) After receiving AU, server B compares it with the computed AU'. If they are equal, then the identity of A is authenticated.
  By using the random number ra chosen in Step 1 as the seed x of the Chebyshev polynomial map, the key agreement operations are performed as follows:

- (6) User A chooses a random integer r and computes $X = E_{h_{PW}}(T_r(x))$, and then sends it to B. Simple stream XOR approach can be used as the encryption algorithm here.

- (7) Server B chooses a random integer s and computes $Y = E_{h_{PW}}(T_s(x))$, then sends it to A.

- (8) User A and Server B decrypts X and Y, respectively, then computes the shared secret key: $k = T_r(T_x(x)) = T_s(T_r(x)) = T_{rs}(x)$.

3 Hash function based on chaotic maps

The hash function used in the above key agreement is a hash function based on chaotic maps. One dimension piecewise linear chaotic system is defined as

$$X(t+1) = F_p(X(t)) = \begin{cases} X(t)/P & \text{if } 0 \le X(t) < P, \\ (X(t) - P)/(0.5P) & \text{if } P \le X(t) < 0.5, \\ (1 - X(t) - P)/(0.5 - P) & \text{if } 0.5 \le X(t) < 1 - P, \\ (1 - X(t))/P & \text{if } 1 - P \le X(t) \le 1 \end{cases}$$

where $X \in [0, 1]$, $P \in (0, 0.5)$. X_0 and H_0 are secretly chosen from $[0, 1]$ as the keys. The 3-unit iterations, 1st to Nth, (N+ 1)th to 2Nth, (2N+1)th to 3Nth, ensure that each bit of the final hash value will be related to all the bits of messages [10]. The following is a brief referring to how to generate the hash value:

- (1) Translates the pending message to the corresponding ASCII numbers, then maps these ASCII numbers by means of linear transform into an array C, whose elements are numbers in [0, 1] and whose length N is the number of characters in the message.

- (2) The iteration process is as follows:
  1st: $P_1 = (C_1 + H_0)/4 \in [0, 0.5)$, $X_1 = F_{P_1}(X_0) \in [0, 1]$;
  2nd to Nth: $P_i = (C_i + X_{i-1})/4 \in [0, 0.5)$, $X_i = F_{P_i}(X_{i-1}) \in [0, 1]$;
  (N+1)th: $P_{N+1} = (C_N + X_N)/4 \in [0, 0.5)$, $X_{N+1} = F_{P_{N+1}}(X_N) \in [0, 1]$;
  (N+ 2) to 2Nth: $P_i = (C_{2N-i+1} + X_{i-1})/4 \in [0, 0.5)$, $X_i = F_{P_i}(X_{i-1}) \in [0, 1]$;
  (2N + 1): $P_{2N+1} = (C_1 + H_0)/4 \in [0, 0.5)$, $X_{2N+1} = F_{P_{2N+1}}(X_{2N}) \in [0, 1]$;
  (2N+ 2) to 3Nth: $P_i = (C_{i-2N} + X_{i-1})/4 \in [0, 0.5)$, $X_i = F_{P_i}(X_{i-1}) \in [0, 1]$.

- (3) Transforms X_N, X_{2N}, X_{3N} to the corresponding binary format, extracts 40, 40, 48 bits after the decimal point, respectively, and juxtaposes them from left to right to get a 128-bit final hash value.

4 Bergamo et al.'s attack

In order to study the security of the cryptosystems based on chaotic maps in [7], Bergamo et al. proposed security compromise method and successfully broke Kocarev and Tasev's cryptosytem [2, 7]. Bergamo et al.'s method also applied to the security of any Diffie-Hellman like key agreement protocol based on chaotic maps [9] if the adversary can obtain public information x, $T_s(x)$ and $T_r(x)$. In fact,

- the adversary can compute an r' such that $T_{r'}(x) = T_r(x)$:

$$r' = \frac{\arccos(T_r(x)) + 2k\pi}{\arccos(x)} | k \in Z \tag{1}$$

- the adversary evaluates $T_{r's}(x) = T_{r'}(T_s(x))$. Therefore, the shared session key $T_{rs}(x)$ is compromised.

From Bergamo et al.'s attack, we can see that their attack does not work on Xiao et al.'s protocol in Section 2. This is because an attacker cannot get $T_r(x)$ and $T_s(x)$. Therefore, it is impossible to find r' or s' such that $T_{r'}(x) = T_r(x)$ or $T_{s'}(x) = T_s(x)$.

5 Security of Xiao et al.'s key agreement protocol

In this section, we will develop three attacks on Xiao et al.'s key agreement protocol. From these attacks we can see that the novel key agreement protocol in [11] is still vulnerable and cannot help the user and the server to fulfil their purpose in establishing a secure session key.

5.1 Attacks on Xiao et al.'s key agreement protocol

For each full run of the key agreement in Section 2, we call it a protocol run. For the i-th full run of the key agreement, we call it the i-th protocol run.

Suppose the seeds for the Chebyshev polynomial map in the i-th protocol run and the j-th protocol run are x and y, respectively. Here, $i < j$ and $x \neq y$ ($x, y \in [-1, 1]$ are random numbers).

5.1.1 Approach 1

With different seeds x and y

In the i-th protocol run, we inspect the step 6 and the step 7.
(Step 6i) User A chooses a random integer r and computes $X_i = E_{h_{PW}}(T_r(x))$, and then sends it to server B.

(Step 7i) Server B chooses a random integer s and computes $Y_i = E_{h_{PW}}(T_s(x))$, then sends it to user A.

An adversary can easily intercept X_i and Y_i. Actually, this adversary cannot get $T_r(x)$ and $T_s(x)$ since they are encrypted. However, the adversary can take advantage of X_i and Y_i as soon as she tries to prevent user A and server B from establishing a shared session key in the j-th protocol run. In the following, we look at the j-th protocol run through step-by-step to demonstrate how the adversary can prevent user A and server B from establishing a shared session key.

The j-th protocol run:

- (Step 1j) User A chooses a random number $y \in [-1, 1]$, then he sends his user identity number, ID_A, and y to server B.
- (Step 2j) Server B chooses a random number $rb \in [-1, 1]$ and sends it back to user A.
- (Step 3j) User A juxtaposes h_{PW}, y and rb from left to right as the pending message, and uses the one-way hash function $H(\cdot)$ to compute the authentication value $AU = H(h_{PW}, y, rb)$. Then he sends ID_A and AU to server B.
- (Step 4j) Server B takes out his own copies of h_{PW}, y and rb corresponding to the user identity number ID_A. Then she uses the same Hash function to compute $AU' = H(h_{PW}, y, rb)$ similarly.
- (Step 5j) After receiving AU, server B compares it with the computed AU'. If they are equal, then the identity of A is authenticated.
  By using the random number y chosen in Step 1j as the seed of the Chebyshev polynomial map, then
- (Step 6j) User A chooses a random integer r_j and computes $X_j = E_{h_{PW}}(T_{r_j}(y))$, and then sends it to B.
- (Step 7.1j) The adversary intercepts X_j and does not let it arrive at server B.
- (Step 7.2j) The adversary replaces X_j with X_i which was intercepted in the

i-th protocol run. Then the adversary sends X_i to server B.

- (Step 7.3j) After receiving X_i, Server B chooses a random integer s_j and computes $Y_j = E_{h_{PW}}(T_{s_j}(y))$, then sends it to A.

- (Step 8.1j) After receiving it, User A decrypts $Y_j = E_{h_{PW}}(T_{s_j}(y))$, then computes the shared secret key:

$$k_A = T_{r_j}(T_{s_j}(y)) = T_{s_j}(T_{r_j}(y)) = T_{r_j s_j}(y). \tag{2}$$

- (Step 8.2j) After receiving X_i, Server B decrypts $X_i = E_{h_{PW}}(T_r(x))$, then computes the shared secret key:

$$k_B = T_{s_j}(T_r(x)) = T_r(T_{s_j}(x)) = T_{rs_j}(x). \tag{3}$$

After the j-th protocol run is completed, it is easy to see that $T_{r_j s_j}(y) \neq T_{rs_j}(x)$. This is because of the randomness of x and y as well as r_j and r. Therefore, the adversary successfully prevented user A and server B from establishing a shared session key.

5.1.2 Approach 2

With different secret random integers r_j, s and s_j, and different seeds x and y_{j_1}.

The j-th protocol run:

- (Step 1j) User A chooses a random number $y_{j_1} \in [-1, 1]$, then he sends his user identity number, ID_A, and y_{j_1} to server B.

- (Step 2j) Server B chooses a random number $y_{j_3} \in [-1, 1]$ and sends it back to user A.

- (Step 3j) User A juxtaposes h_{PW}, y_{j_1} and rb from left to right as the pending message, and uses the one-way hash function $H(\cdot)$ to compute the authentication value $AU = H(h_{PW}, y_{j_1}, y_{j_3})$. Then he sends ID_A and AU to server B.

- (Step 4j) Server B takes out his own copies of h_{PW}, y_{j_1} and rb corresponding to the user identity number ID_A. Then she uses the same Hash function to compute $AU' = H(h_{PW}, y_{j_1}, y_{j_3})$ similarly.

- (Step 5j) After receiving AU, server B compares it with the computed AU'. If they are equal, then the identity of A is authenticated.

By using the random number y_{j_1} chosen in Step 1j as the seed of the Chebyshev polynomial map, then

- (Step 6j) User A chooses a random integer r_j and computes $X_j = E_{h_{PW}}(T_{r_j}(y_{j_1}))$, and then sends it to B.

- (Step 7.1j) After receiving X_j, Server B chooses a random integer s_j and computes $Y_j = E_{h_{PW}}(T_{s_j}(y_{j_1}))$, then sends it to A.

- (Step 7.2j) The adversary intercepts Y_j and does not let it arrive at user A.

- (Step 7.3j) The adversary replaces Y_j with Y_i which was intercepted in the i-th protocol run. Then the adversary sends Y_i to user A.

- (Step 8.1j) User A decrypts Y_i, then computes the shared secret key:

$$k_A = T_{r_j}(T_s(x)) = T_s(T_{r_j}(x)) = T_{r_j s}(x). \tag{4}$$

- (Step 8.2j) Server B decrypts X_j, then computes the shared secret key:

$$k_B = T_{s_j}(T_{r_j}(y_{j_1})) = T_{r_j}(T_{s_j}(y_{j_1})) = T_{r_j s_j}(y_{j_1}). \tag{5}$$

After the j-th protocol run is completed, because of the randomness of r_j, s_j and s, as well as y_{j_1} and x, it is easy to see that $k_A \neq k_B$. Therefore, the adversary successfully prevented user A and server B from establishing a shared session key.

5.1.3 Approach 3

With different secret random integers r_j, s, s_j and r_j.

The j-th protocol run:

- (Step 1j) User A chooses a random number $y_{j_2} \in [-1, 1]$, then he sends his user identity number, ID_A, and y_{j_2} to server B.

- (Step 2j) Server B chooses a random number $rb \in [-1, 1]$ and sends it back to user A.

- (Step 3j) User A juxtaposes h_{PW}, y_{j_2} and rb from left to right as the pending message, and uses the one-way hash function $H(\cdot)$ to compute the authentication value $AU = H(h_{PW}, y_{j_2}, rb)$. Then he sends ID_A and AU to server B.

- (Step 4j) Server B takes out his own copies of h_{PW}, y_{j_2} and rb corresponding to the user identity number ID_A. Then she uses the same Hash function to compute $AU' = H(h_{PW}, y_{j_2}, rb)$ similarly.

- (Step 5j) After receiving AU, server B compares it with the computed AU'. If they are equal, then the identity of A is authenticated.
  By using the random number y_{j_2} chosen in Step 1j as the seed of the Chebyshev polynomial map, then
- (Step 6j) User A chooses a random integer r_j and computes $X_j = E_{h_{PW}}(T_{r_j}(y_{j_2}))$, and then sends it to B.

- (Step 7.1j) The adversary intercepts X_j and does not let it arrive at server B.

- (Step 7.2j) The adversary replaces X_j with X_i which was intercepted in the i-th protocol run. Then the adversary sends X_i to server B.

- (Step 7.3j) After receiving X_i, Server B chooses a random integer s_j and computes $Y_j = E_{h_{PW}}(T_{s_j}(y_{j_2}))$, then sends it to A.

- (Step 7.4j) The adversary intercepts Y_j and does not let it arrive at user A.

- (Step 7.5j) The adversary replaces Y_j with Y_i which was intercepted in the i-th protocol run. Then the adversary sends Y_i to user A.

- (Step 8.1j) User A decrypts Y_i, then computes the shared secret key:

$$k_A = T_{r_j}(T_s(x)) = T_s(T_{r_j}(x)) = T_{r_j s}(x). \tag{6}$$

- (Step 8.2j) Server B decrypts X_i, then computes the shared secret key:

$$k_B = T_{s_j}(T_r(x)) = T_r(T_{s_j}(x)) = T_{rs_j}(x). \tag{7}$$

After the j-th protocol run is completed, because of the randomness of r_j, r, s_j, and s, it is easy to see that $T_{r_j s}(x) \neq T_{rs_j}(x)$. Therefore, the adversary successfully prevented user A and server B from establishing a shared session key, which will be used for subsequent cryptographic applications and communications.

6 Conclusions

The idea of Xiao et al.'s key agreement is novel. Their new key agreement protocol enhanced the security of key agreement based on chaotic maps such that it can resist Bergamo et al.'s attack. However, in this paper, we pointed out that there still exist security issues in their protocol. The developed new

attacks can hinder the user and the server from accomplishing a shared secret session key.

Acknowledgements

The author sincerely thanks the anonymous reviewers. This work is supported by the CBS and Curtin R& D Office at Curtin University of Technology.

References

[1] G. Alvarez, Security problems with a chaos-based deniable authentication scheme, Chaos, Solitons & Fractals, 2005, 26:7-11.

[2] P. Bergamo, P. D'Arco, A. Santis and L. Kocarev, Security of public key cryptosystems based on Chebyshev polynomials, IEEE Transactions on Circuits and Systems-I, July 2005, 52(7): 1382-1393.

[3] F. Dachselt and W. Schwarz, Chaos and cryptography, IEEE Transactions on Circuits and Systems-I, Fundamental Theory, December 2001, 48(12): 1498-1509.

[4] G. Fridrich, Symmetric ciphers based on two-dimensional chaotic maps, Internatinal Journal of Bifurcation and Chaos, 1998, 8: 1259-1284.

[5] Pecorra, T.L., Carroll, T.L., Driving systems with chaotic signals, Phys. Rev. A, 1991, 44(4): 2374-83.

[6] L. Kocarev, Chaos-based cryptography: A brief overview, IEEE Circuits and Systems Magazine, 2001, 1: 6-21.

[7] L. Kocarev and Z. Tasev, Public-key encryption based on chebyshev maps, in Proceedings of IEEE Symp. Circuits and Systems (ISCAS'03), 2003, 3: 28-31.

[8] A. Menezes, P. van Oorschot and S. Vanstone, Handbook of applied cryptography, CRC Press, Boca Raton, 1997.

[9] D. Xiao, X. Liao, K. Wong, An efficient entire chaos-based scheme for deniable authentication, Chaos, Solitons & Fractals, 2005, 23:1327-1331.

[10] D. Xiao, X. Liao, S. Deng, One-way hash function construction based on the chaotic map with changeable-parameter, Chaos, Solitons & Fractals, 2005, 24:65-71.

[11] D. Xiao, XF. Liao, SJ. Deng, A novel key agreement protocol based on chaotic maps. Information Sciences 2006, doi:10.1016/j.ins.2006.07.026.

[12] G. Grassi and S. Mascolo, IEEE Trans. Circuits Syst., I: Fundam. Theory Appl. 46, 1135 (1999).

[13] K.W. Wong, A fast chaotic cryptographic scheme with dynamic look-up table, Physics Letters A, vol. 298, pp. 238-242, 2002.