A secure chaotic key agreement without clock synchronization

Xianfeng Guo
College of Computer Science and Technology
Southwest University for Nationalities
Chengdu, 610041, China
Key Laboratory of SIP of Sichuan Province,
Southwest Jiaotong University,
Chengdu, 610031, China
guoxianf@126.com

Jiashu Zhang
Key Laboratory of SIP of Sichuan Province,
Southwest Jiaotong University,
Chengdu, 610031, China
guoxianf@126.com

***Abstract*—Recently, Han-Chang proposed a flexible chaotic key agreement protocol based on nonce. In this paper, we demonstrate that it is vulnerable to spoofing attack and replaying attack. Furthermore, a secure improvement is suggested, which avoids the flaws while keeping all the merits of the original scheme.**

Keywords-Chaos; Key agreement; replaying attack; mutual authentication

I. INTRODUCTION

Group key establishment protocol [1,2] allows participants to construct a common conference key for secure communication over an open channel. The key agreement protocol [3,4] is a key establishment technique in which all participants cooperatively establish the communication key by using the contribution of every group member, and no one can predetermine the key. Generally speaking, most of the key agreement protocols are based on lower efficiency traditional public-key cryptography [5,6]. Therefore, how to achieve the performance lower bound in terms of time, communication, and computation cost have attracted more and more attention.

Over the past decade, chaotic dynamics system has aroused extensive concern to construct secure and efficient cryptosystems [7-13]. Among these schemes, L. Kocarev et al. presented a novel public key encryption algorithm [8] by utilizing the semi-group property of Chebyshev polynomial map. The public key encryption algorithm is not only efficient but also extensible to be applied to multi-party. D.Xiao et al. [14] firstly proposed an original chaotic key agreement protocol based on algorithm [8]. Then presented two improved protocols [15,16] to resist man-in-the-middle attack [17,18] and replaying attack, respectively. Unfortunately, Han [19] pointed out that in the system of D.Xiao et al. [15], an adversary can prevent the user and the server from establishing a shared session key by replaying the eavesdropping message of former protocol run. Later on, Han-Chang [20] proposed an improved one. The improved protocol based on nonce can support an environment where the timeline is not so critical to the communication parties. Moreover, it utilizes mutual authentication to enhance its security.

In this paper, we demonstrate that the mutual authentication of Han-Chang's scheme is impolitic, i.e., neither user authentication nor server authentication can resist the spoofing attack. Furthermore, we point out that their scheme is susceptible to replaying attack. To surmount the aforementioned pitfalls, we propose a secure protocol based on Han-Chang's. The proposed scheme utilizes Hash function to avoid those security flaws and keeps all the merits of the original scheme at the same time.

The rest of this paper is organized as follows. Section 2 briefly reviews Han-Chang's scheme. Section 3 elaborates the cryptanalysis of their scheme. Section 4 presents a secure key agreement protocol based on chaotic Hash. Section 5 demonstrates the security analysis of the proposed protocol. Concluding remarks are given in section 6.

II. REVIEW OF HAN-CHANG'S SCHEME

Let PW denotes the password of the User A. H(·) is a chaotic Hash function. ID_A is the user's identity number. ID_B is the server's identity number. β is a random number and the private key of the server. Server B and the User A secretly share the hash value $h = H(ID_B, ID_A, \beta, PW)$, where ID_B, ID_A, β and PW are concatenated as the pending message from left to right. The scheme works as follows:

(1) User A chooses a random number $r_1 \in [-1,1]$, a random nonce n_1, and juxtaposes h, r_1, n_1 and ID_A from left to right as the pending message, and use the hash function H(·) to compute $AU_1 = H(h, r_1, n_1, ID_A)$.

(2) User A sends AU_1, r_1, n_1 and ID_A to Server B. After receiving AU_1, r_1, n_1 and ID_A, the server B uses the hash function H(·) to compute $AU_2 = H(h, r_1, n_1, ID_A)$. Server B then compares whether $AU_2 = AU_1$. If not, then B stops here; otherwise, User A is authenticated and Server B goes to the next step.

(3) Server B chooses a random number $r_2 \in [-1,1]$, a random nonce n_2, and juxtaposes h, r_2, n_2 and ID_B from left to right as the pending message, and use the hash function H(·) to compute $AU_3 = H(h, r_2, n_2, ID_B)$. Server B then sends AU_3, r_2, n_2 and ID_B to User A.

(4) After receiving AU_3, r_2, n_2 and ID_B, User A computes $AU_4 = H(h, r_2, n_2, ID_B)$. User A then compares whether $AU_4 = AU_3$. If not, then A stops here; otherwise, Server B is authenticated and User A goes to the next step.

978-1-4244-8270-2/10/$26.00©2010 IEEE

(5) User A chooses a random integer j (where $j \le j_0$, j_0 is a threshold value such that the semi-group property holds) and computes $X = E_h(n_1, T_j(x))$ (where E_h is a symmetric encryption algorithm with h as the encryption/decryption key). A then sends X to Server B.

(6) Server B chooses a random integer i (where $i \le j_0$), computes $Y = E_h(n_2, T_i(x))$, and sends it to User A.

(7) After receiving X, Server B can get n_1' and $T_j(x)$ by decrypting X. Before computing the shared secret session key, B should check whether the relation $n_1' = n_1$ holds. If the relation holds, B can compute the shared secret session key as $k = T_i(T_j(x)) = T_{ij}(x) = T_{ji}(x) = T_j(T_i(x))$. If B found that the relation does not hold, then B stops here and restarts the key agreement process with A.

(8) After receiving Y, User A can get n_2' and $T_i(x)$ by decrypting Y. Before computing the shared secret session key, A should check whether $n_2' = n_2$. If equal, A can compute the shared secret session key as $k = T_j(T_i(x)) = T_{ji}(x) = T_i(T_j(x)) = T_{ij}(x)$. If A found that the relation does not hold, then A stops here and restarts the key agreement process with B.

III. CRYPTANALYSIS OF HAN-CHANG'S SCHEME

Han-Chang's scheme is vulnerable and can easily be cryptanalyzed. That is, neither spoofing attack nor replaying attack can be resisted.

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

Suppose the seeds for the Chebyshev polynomial map in the pth protocol run and the qth protocol run are x and y, respectively. Here, $p < q$ and $x \ne y$ ($x, y \in [-1,1]$ are random numbers).

A. Spoofing attack

In the pth protocol run, we inspect Step (2) and (3) to analyze the spoofing attacks.

(pth-2) User A sends $AU_1^p = H(h, r_1^p, n_1^p, ID_A)$, r_1^p, n_1^p and ID_A to Server B.

(pth-3) Server B sends $AU_3^p = H(h, r_2^p, n_2^p, ID_B)$, r_2^p, n_2^p and ID_B to User A.

An adversary *Eve* can overhear the above message from the communication channel. Then *Eve* can utilize the obtained message to personate User A or Server B or both of them in the qth protocol run. Eve personate Server B to play server spoofing attack can be elaborated as follows:

(qth-1) User A chooses a random number $r_1 \in [-1,1]$, a random nonce n_1, and juxtaposes h, r_1, n_1 and ID_A from left to right as the pending message, and use the hash function H(·) to compute $AU_1 = H(h, r_1, n_1, ID_A)$.

(qth-2) User A sends AU_1, r_1, n_1 and ID_A to Server B. After receiving AU_1, r_1, n_1 and ID_A, the Server B uses the hash function H(·) to compute $AU_2 = H(h, r_1, n_1, ID_A)$. Server B then compares whether $AU_2 = AU_1$. If not, then B stops here; otherwise, User A is authenticated and Server B goes to the next step.

(qth-3) Server B chooses a random number $r_2 \in [-1,1]$, a random nonce n_2, and juxtaposes h, r_2, n_2 and ID_B from left to right as the pending message, and use the hash function H(·) to compute $AU_3 = H(h, r_2, n_2, ID_B)$. Server B then sends AU_3, r_2, n_2 and ID_B to User A.

(qth-4) The adversary *Eve* intercepts AU_3, r_2, n_2 and ID_B from arriving at User A, and then replays $AU_3^p = H(h, r_2^p, n_2^p, ID_B)$, r_2^p, n_2^p, and ID_B, which were eavesdropped in the pth protocol run, to User A. After receiving AU_3^p, r_2^p, n_2^p and ID_B, User A computes $AU_4 = H(h, r_2^p, n_2^p, ID_B)$, then compares whether $AU_4 = AU_3$ to verify Sever B. Apparently, Eve is authenticated successfully.

The analysis indicates that, in the qth protocol run, the adversary *Eve* can perform the server spoofing attack by utilizing the message of pth protocol run. Similarly, *Eve* personates User A to spoof Server B can be briefly described as follows:

(1) Intercepts AU_1, r_1, n_1 and ID_A in the qth protocol run;

(2) Replaces AU_1, r_1, n_1 and ID_A with $AU_1^p = H(h, r_1^p, n_1^p, ID_A)$, r_1^p, n_1^p and ID_A in the Step 2 of qth protocol run.

Indeed, the spoofing attack, applies to both of User A and Server B, is working on the same principle.

B. Replaying attack

In the pth protocol run, Steps (5) and (6) are analyzed to demonstrate the replaying attacks.

(pth-5) User A chooses a random integer j_p, computes $X_p = E_h(n_1^p, T_{jp}(x))$ and sends X_p to Server B, where n_1^p is the nonce selected by User A.

(pth-6) Server B chooses a random integer i_p, computes $Y_p = E_h(n_2^p, T_{ip}(x))$, and sends Y_p to User A, where n_2^p is the nonce selected by Server B.

The adversary Eve can overheard X_p and Y_p from the communication channel freely. *Eve* replace $X = E_h(n_1, T_j(y))$ of qth protocol run with $X_p = E_h(n_1^p, T_{jp}(x))$ to perform replaying attack in the qth protocol run can be described as follows:

(qth-5) User A chooses a random integer j (where $j \le j_0$, j_0 is a threshold value such that the semi-group property holds) and computes $X = E_h(n_1, T_j(y))$ (where E_h is a symmetric encryption algorithm with h as the encryption/decryption key). A then sends X to Server B.

(qth-6) Server B chooses a random integer i (where $i \le j_0$), computes $Y = E_h(n_2, T_i(y))$, and sends it to User A.

(qth-7) After receiving X, Server B can get n_1' and $T_j(y)$ by decrypting X. Before computing the shared secret session key, Sever B should check whether the relation $n_1' = n_1$ holds. If yes, Sever B computes the shared secret session key $k = T_i(T_j(y)) = T_{ij}(y) = T_j(T_i(y))$.

(qth-8.1) Eve intercepts $Y = E_h(n_2, T_i(y))$, and replays $Y_p = E_h(n_2^p, T_{ip}(x))$ to User A, where Y_p is overheard in the pth protocol run.

(qth-8.2) After receiving $Y_p = E_h(n_2^p, T_{ip}(x))$, User A can get n_2' and $T_i(y)$ by decrypting Y_p. Before computing the shared secret session key, A should check whether $n_2' = n_2^p$. If equal, User A computes the shared secret session key $k_A = T_j(T_{ip}(x)) = T_{j.ip}(x) \neq T_{ij}(y) = k$.

The analysis show that Eve can successfully replay the former message to prevented User A and Server B from establishing a shared session key. It's the same principle, *Eve* can replace $X = E_h(n_1, T_j(y))$ of qth protocol run with $X_p = E_h(n_1^p, T_{jp}(x))$. Of course, this kind of replaying attack can be applied to both of Serve B and User A by replacing $X = E_h(n_1, T_j(y))$ and $Y = E_h(n_2, T_i(y))$ with $X_p = E_h(n_1^p, T_{jp}(x))$ and $Y_p = E_h(n_2^p, T_{ip}(x))$ at the same time, respectively.

IV. Proposed Secure Protocol

The proposed secure scheme provides server authentication by nonce and user authentication by acknowledging a Hash value. Moreover, it utilizes the received nonce to prevent replaying attack.

The denotations are the same as Section 2. *PW* denote the password of the User A. H(·) is a chaotic hash function. ID_A and ID_B are User A and Server B's identity number, respectively. β is a random number and the private key of Server B. Server B and the User A secretly share the hash value $h = H(ID_B, ID_A, \beta, PW)$, where ID_B, ID_A, β and PW are concatenated as the pending message from left to right.

(1) User A chooses a random number $r_1 \in [-1,1]$, a random nonce n_1, and juxtaposes h, r_1, n_1 and ID_A from left to right as the pending message, and use the hash function H(·) to compute $AU_1 = H(h, r_1, n_1, ID_A)$.

(2) User A sends AU_1, r_1, n_1 and ID_A to Server B. After receiving AU_1, r_1, n_1 and ID_A, the Server B uses the hash function H(·) to compute $AU_2 = H(h, r_1, n_1, ID_A)$. Server B then compares whether $AU_2 = AU_1$. If not, then B stops here; otherwise, Server B goes to the next step.

(3) Server B chooses a random number $r_2 \in [-1,1]$, a random nonce n_2, and juxtaposes h, r_2, n_2 and ID_B from left to right as the pending message, and use the hash function H(·) to compute $AU_3 = H(h, r_2, n_2, ID_B, n_1)$. Server B then sends AU_3, r_2, n_2 and ID_B to User A.

(4) After receiving AU_3, r_2, n_2 and ID_B, User A computes $AU_4 = H(h, r_2, n_2, ID_B, n_1)$. User A then compares whether $AU_4 = AU_3$. If not, then A stops here; otherwise, Server B is authenticated, then User A use the hash function H(·) to compute $AU_5 = H(h, n_1, n_2)$ and sends AU_5 to Server B.

(5) Upon receiving AU_5, Server B calculates $AU_6 = H(h, n_1, n_2)$ and compares whether $AU_6 = AU_5$. If not, then Sever B stops here; otherwise, User A is authenticated.

(6) User A chooses a random integer j (where $j \le j_0$, j_0 is a threshold value such that the semi-group property holds) and computes $X = E_h(n_2, T_j(x))$ (where E_h is a symmetric encryption algorithm with h as the encryption/decryption key). A then sends X to Server B.

(7) Server B chooses a random integer i (where $i \le j_0$), computes $Y = E_h(n_1, T_i(x))$, and sends it to User A.

(8) After receiving X, Server B can get n_2' and $T_j(x)$ by decrypting X. Before computing the shared secret session key, B should check whether the relation $n_2' = n_2$ holds. If the relation holds, B can compute the shared secret session key as $k = T_i(T_j(x)) = T_{ij}(x) = T_{ji}(x) = T_j(T_i(x))$. If B found that the relation does not hold, then B stops here and restarts the key agreement process with A.

(9) After receiving Y, User A can get n_1' and $T_i(x)$ by decrypting Y. Before computing the shared secret session key, A should check whether $n_1' = n_1$. If equal, A can compute the shared secret session key as $k = T_j(T_i(x)) = T_{ji}(x) = T_i(T_j(x)) = T_{ij}(x)$. If A found that the relation does not hold, then A stops here and restarts the key agreement process with B.

V. Security Analysis of the Proposed Secure Scheme

A. Spoofing attack

If a masqueraded server or intruder intends to deceive User A, it has to generate a valid authentication message, i.e. $\{AU_3, r_2, n_2, ID_B\}$, for User A checking in Step (4) of Section 4. As an example, we assume that an adversary *Eve* is over the insecure network and he has intercepted the transmission between User A and Server B. *Eve* can't generate forge $AU_3 = H(h, r_2, n_2, ID_B, n_1)$, because it is computed by the secret key h. The security of the secret key h is protected by one-way property of Hash function H(·) [11]. Furthermore, *Eve* can't replay the authentication message of former protocol run, because every $AU_3 = H(h, r_2, n_2, ID_B, n_1)$ has a fresh random number n_1

generated by User *A*. Therefore, the server spoofing attack is aborted in our improved scheme.

If a masqueraded user or intruder intends to deceive Server *B*, it has to generate a valid authentication message $AU_5 = H(h, n_1, n_2)$. For the same principle, it is infeasible. As a result, our improved scheme can successfully resist the spoofing attack. Neither server spoofing attack nor user spoofing attack can be applied to the proposed scheme.

B. Replaying attack

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

In the *p*th protocol run of our improved scheme in Section 4, we assume that an adversary *Eve* has intercepted $X_p = E_h(n_2^p, T_{jp}(x))$ and $Y_p = E_h(n_1^p, T_{ip}(x))$ from the communication channel. If *Eve* replaces $X = E_h(n_2, T_j(y))$ with $X_p = E_h(n_2^p, T_{jp}(x))$ in the *q*th protocol run, it will be exposed, because Server *B* can get n_2^p by decrypting X_p, and check whether $n_2^p = n_2$ in Step (8) of Section 4, where n_2 is a fresh random number selected by Sever *B*. Replacing $Y = E_h(n_1, T_i(x))$ with $Y_p = E_h(n_1^p, T_{ip}(x))$ will be found when check whether $n_1^p = n_1$ in step (9) of Section 4.

VI. CONCLUSIONS

This paper pointed out that Han-Chang's scheme can resist neither spoofing attack nor replaying attack. Moreover, it proposed a secure group key agreement protocol based on chaotic Hash. The proposed scheme utilizes the chaotic Hash function and the nonce to achieve the secure mutual authentication and avoid the replaying attacks. Analysis indicates that the proposed scheme can overcome all the weaknesses and keep all the merits of Han-Chang's.

ACKNOWLEDGEMENTS

This work described here was supported in part by the National Natural Science Foundation of China (No. 60971104), the Fundamental Research Funds for the Central Universities (No. SWJTU09ZT16), the Science & Technology Key Plan Project of Chengdu (NO. 10GGYB649GX- 023) and the Foundation of Southwest University for Nationalities (No. 09NYB002 and Y-2010-08).

REFERENCES

[1] S. Setia, S.Koussih, S. Jajodia, Kronos: a scalable group re-keying approach for secure multicast, in Proc. IEEE Symp. Security and Privacy, May 2000, pp. 215–228.

[2] C. K.Wong, M. Gouda, S. S. Lam, Secure group communications using key graphs, IEEE/ACM Trans. Netw., vol. 8, no. 1, Feb. 2000, pp. 16-30.

[3] W. Diffie, M. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory, vol. 22, no. 6, pp. 644–654, 1976.

[4] Patrick P. C. Lee, John C. S. Lui, David K. Y. Yau, Distributed Collaborative Key Agreement and Authentication Protocols for Dynamic Peer Groups, IEEE/ACM Transactions on Networking, 14(2006) 263–276.

[5] M. Steiner, G. Tsudik, , and M. Waidner, Key agreement in dynamic peer groups, IEEE Transactions on Parallel and Distributed Systems, 11(2000)769–780.

[6] W. Yu, Y. Sun, K.J. Ray Liu, Minimization of Rekeying Cost for Contributory Group Communications, IEEE GLOBECOM, December 2005, pp.1716–1720.

[7] R. Tenny, L. Tsimring, L. Larson, H. Abarbanel, Using distributed nonlinear dynamics for public key encryption, Physical Review Letters 90 (2003) 047903.

[8] L. Kocarev, Z. Tasev, Public key encryption based on Chebyshev maps, in: Proc. 2003 IEEE Symposium on Circuits and Systems, Bangkok, TH, vol. 3, pp. 28–31.

[9] E. Alvarez, A. Fernandez, P. Garcia, J. Jimenez, A. Marcano, New approach to chaotic encryption, Physics Letters A 263 (1999) 373–375.

[10] K. Wong, A fast chaotic cryptographic scheme with dynamic look-up table, Physics Letters A 298 (2002) 238–242.

[11] D. Xiao, X. Liao, S. Deng, One-way Hash function construction based on the chaotic map with changeable-parameter, Chaos, Solitons & Fractals 24 (2005) 65–71.

[12] J. Zhang, X. Wang, W. Zhang, Chaotic keyed hash function based on feedforward–feedback nonlinear digital filter, Physics Letters A, 362 (2007), 439-448.

[13] X. Guo, J. Zhang, Keyed one-way Hash function construction based on the chaotic dynamic S-Box , Acta Physica Sinica, 55(2006), 4442-4449.

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

[15] D. Xiao, X. Liao, S. Deng, A novel key agreement protocol based on chaotic maps, Information Sciences 177(2007)1136-1142.

[16] D. Xiao, X. Liao, S. Deng, Using time-stamp to improve the security of a chaotic maps-based key agreement protocol, Information Sciences 178 (2007), 1598-1602.

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

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

[19] Song Han, Security of a key agreement protocol based on chaotic maps, Chaos, Solitons and Fractals 38 (2007), 764-768.

[20] S. Han, E. Chang, Chaotic map based key agreement with/out clock synchronization, Chaos, Solitons and Fractals 39 (2009) 1283–1289.