A New Simple Authenticated Key Agreement and Protected Password Change Protocol

Eun-Jun Yoon and Kee-Young Yoo*

Department of Computer Engineering, Kyungpook National University,
Daegu 702-701, South Korea
Tel.: +82-53-950-5553; Fax: +82-53-957-4846
ejyoon@infosec.knu.ac.kr, yook@knu.ac.kr

Abstract. In 2005, Chang et al. proposed a simple authenticated key agreement and protected password change protocol. However, Chang et al.'s schemes are still susceptible to stolen-verifier attack and Denial-of-Service attacks. Accordingly, the current paper demonstrates the vulnerability of Chang et al.'s schemes to two simple attacks and then presents an improved scheme to resolve such problems. In contrast to Chang et al.'s protected password change protocol, the proposed protected password change protocol can securely update user passwords without a complicated process, while also providing greater security.

Keywords: Cryptography, Authentication, Password, Key agreement, Password change, Dffie-Hellman key agreement.

1 Introduction

The Dffie-Hellman key agreement scheme [1], which developed to provide a common session key between two parties, is an epochal breakthrough that can produce a common session key without any prior common information. However, this method has a weakness of possible man-in-the middle attacks [2]. Recently, Seo and Sweeney [3] proposed a new key agreement protocol based on the Diffie-Hellman protocol called the simple authenticated key agreement algorithm (SAKA). In the SAKA protocol, two parties have a pre-shared password [4] for data communication, produce a session key by exchanging messages, and confirm each other. Because they can simplify key agreement, SAKA-like protocols are widely used in research on key agreement, and therefore there have been numerous attempts to enhance SAKA-like protocols.

In 2002, Yeh and Sun [5], and Kobara and Imai [6] combined the pre-shared password technique and the Diffie-Hellman scheme to achieve the same purpose as the SAKA-like schemes. In 2005, Chang et al. [7] modified Yeh and Sun's SAKA-like protocol to improve its efficiency and also proposed a protected password change protocol to achieve user authentication and to arbitrarily change a password. Chang et al.'s key agreement protocol requires fewer steps and less computation cost then the Kobara-Imai scheme. Moreover, Chang et al. not

* Corresponding author.

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 955–964, 2005.

only give a heuristic security analysis, but also formally prove it using Ballare, Poincheval and Rogaway's model [8].

However, Chang et al.'s schemes are still susceptible to stolen-verifier attack [9], in which obtaining the secret data stored in a server can allow an illegitimate user to login to a server as a legitimate user. Additionally, their protected password change protocol suffers from a Denial-of-Service attacks [9], in which an attacker can easily make the server reject all subsequent login requests from any user. Accordingly, the current paper demonstrates that Chang et al.'s schemes are vulnerable to stolen-verifier attack and Denial-of-Service attacks and also presents an improved scheme to isolate such problems. In contrast to Chang et al.'s protected password change protocol, the proposed schemes can securely update user passwords without a complicated process, while also providing greater security.

The remainder of this paper is organized as follows: Section 2 defines the security properties. Section 3 briefly reviews Chang et al.'s schemes, then Section 4 demonstrates stolen-verifier attack and Denial-of-Service attacks with their schemes. The proposed schemes are presented in Section 5, while Section 6 discusses the security of the proposed schemes. Our conclusions are presented in Section 7.

2 Security Properties

The following security properties of the authentication protocols should be considered. Password authentication protocols are very subject to replay, password guessing, and stolen-verifier attacks [9].

(1) Replay attack: A replay attack is an offensive action in which an adversary impersonates or deceives another legitimate participant through the reuse of information obtained in a protocol.
(2) Guessing attack: A guessing attack involves an adversary simply (randomly or systematically) trying passwords, one at a time, in hope that the correct password is found. Ensuring that passwords are chosen from a sufficiently large space can resist exhaustive password searches. However, most users select passwords from a small subset of the full password space. Such weak passwords with low entropy are easily guessed by using so-called dictionary attack.
(3) Stolen-verifier attack: In most applications, the server stores verifiers of users' passwords (e.g., hashed passwords) instead of the clear text of passwords. The stolen-verifier attack means that an adversary who steals a password-verifier from the server can use it *directly* to impersonate a legitimate user in a user authentication execution. Note that the main purpose of an authentication scheme against the stolen-verifier attack is to reduce the immediate danger to user authentication. In fact, an adversary who has a password-verifier may further mount a guessing attack.

Password change protocols allow an authenticated user to change his/her password. Besides those attacks mentioned above, a password change protocol is very vulnerable to Denial-of-Service attacks [9].

(1) Denial-of-Service attack: A Denial-of-Service attack prevents or inhibits the normal use or management of communications facilities. This attack may be directed to a specific user. For example, an adversary may perform this attack to cause the server to reject the login of a specific user.

In addition, the following security properties of session key agreement protocols should be considered since they are often desirable in some environments [10].

(1) Implicit key authentication: Implicit key authentication is the property obtained when identifying a party based on a shared session key, which assures that no other entity than the specifically identified entity can gain access to the session key.
(2) Explicit key authentication: Explicit key authentication is the property obtained when both implicit key authentication and key confirmation hold.
(3) Mutual authentication: Mutual authentication means that both the client and server are authenticated to each other within the same protocol, while explicit key authentication is the property obtained when both implicit key authentication and key confirmation hold.
(4) Perfect forward secrecy: Perfect forward secrecy means that if a long-term private key (e.g. user password or server private key) is compromised, this does not compromise any earlier session keys. In password authentication with key distribution, forward secrecy is a highly desirable security feature.

3 Review of Chang et al.'s Schemes

This section briefly reviews Chang et al.'s key agreement protocol and protected password change protocol and then show how stolen-verifier attacks and Denial-of-Service attacks can work on their protocol. Abbreviations used in this paper are as follows:

- id: public user identity of client.
- pw: secret and possibly weak user password.
- K: strong secret key of server.
- p, q: large prime numbers p and q such that $q|p-1$.
- g: generator with order q in the Galois field $GF(p)$, in which Diffie-Hellman problem is considered hard.
- a, b: session-independent random exponents $\in [1, q-1]$ chosen by client and server, respectively.
- sk: shared session key computed by client and server.
- $H(\cdot)$: strong one-way hash function.
- $\oplus$: bit-wise XOR operation.

3.1 Chang et al.'s Simple Authenticated Key Agreement Protocol

Chang et al's simple authenticated key agreement protocol works as follows:

Step 1. Client → Server: $R_A \oplus pw$
The client chooses a random number $a \in [1, q-1]$, computes $R_A = g^a \bmod p$, and then sends $R_A \oplus pw$ to the server.

Step 2. Server → Client: $R_B||H(K_B, R_A)$
After receiving $R_A \oplus pw$, the server recovers R_A by computing $(R_A \oplus pw) \oplus pw$. Then the server chooses a random number $b \in [1, q-1]$, computes $R_B = g^b \bmod p$ and $K_B = R_A^b = g^{ab} \bmod p$, and then sends $R_B||H(K_B, R_A)$ to the client.

Step 3. Client → Server: $H(K_A, R_B)$
After receiving $R_B||H(K_B, R_A)$, the client computes $K_A = R_B^a = g^{ab} \bmod p$ and verifies whether the received $H(K_B, R_A)$ is equal to $H(K_A, R_A)$. If it holds, the client computes $H(K_A, R_B)$ and sends it to the server.

Step 4. Server → Client: *Access granted / denied*
After receiving $H(K_A, R_B)$, the server verifies whether $H(K_A, R_B)$ is equal to $H(K_B, R_B)$. If it is equal, the client and server agree on the common session key $Key = H(K_A) = H(K_B) = H(g^{ab} \bmod p)$.

3.2 Chang et al.'s Protected Password Change Protocol

In Chang et al.'s protected password changing protocol, the server allows the client to change their old password pw to a new password $newpw$. Chang et al's protected password change protocol works as follows:

Step 1*. Client → Server: $R_A \oplus pw||R_A \oplus newpw$
The client chooses a random number $a \in [1, q-1]$, computes $R_A = g^a \bmod p$, and then sends $R_A \oplus pw||R_A \oplus newpw$ to the server.

Step 2*. Server → Client: $R_B||H(K_B, R_A)$
After receiving $R_A \oplus pw||R_A \oplus newpw$, the server recovers R_A by computing $(R_A \oplus pw) \oplus pw$ and uses the recovered R_A to derive $newpw$ by computing $(R_A \oplus newpw) \oplus R_A$. Then the server chooses a random number $b \in [1, q-1]$, computes $R_B = g^b \bmod p$ and $K_B = R_A^b = g^{ab} \bmod p$, and sends $R_B||H(K_B, R_A)$ to the client.

Step 3*. Client → Server: $H(K_A, R_B) \oplus newpw$
After receiving $R_B||H(K_B, R_A)$, the client computes $K_A = R_B^a = g^{ab} \bmod p$ and verifies whether the received $H(K_B, R_A)$ is equal to $H(K_A, R_A)$. If it holds, the client computes $H(K_A, R_B) \oplus newpw$ and sends it to the server.

Step 4*. Server → Client: *Access granted / denied*
After receiving $H(K_A, R_B) \oplus newpw$, the server uses the recovered $newpw$ in Step 2* to derive $H(K_A, R_B)$ by computing $(H(K_A, R_B) \oplus newpw) \oplus newpw$. Then the server verifies whether the recovered $H(K_A, R_B)$ is equal to $H(K_B, R_B)$ or not. If it is equal, the client and server have successfully changed their shared password pw to the new password $newpw$ and they can agree on the common session key $Key = H(K_A) = H(K_B) = H(g^{ab} \bmod p)$.

4 Cryptanalysis of Chang et al.'s Schemes

This section shows that Chang et al.'s schemes is vulnerable to a Denial-of-Service attacks and a stolen-verifier attack.

4.1 Denial-of-Service Attack on Chang et al.'s Protected Password Change Protocol

Usually, the server closes a login session if the number of error login attempts of an account exceeds a limited value (e.g. 3 times). Even so, such a client's account is still workable and later login requests will pass as long as the correct password is provided. However, Chang et al.'s protected password change protocol can suffer from a Denial-of-Service attacks, in which an attacker can easily make the server reject all subsequent login requests from any client.

In Step 1* of Chang et al.'s protected password change protocol, an attacker can simply replace new password digest $R_A \oplus newpw$ with forged new password digest $R_A \oplus newpw \oplus X$, where X is a random number chosen by the attacker. After receiving the replaced messages $(R_A \oplus pw || R_A \oplus newpw \oplus X)$, the server can retrieve R_A from $R_A \oplus pw$ by computing $R_A \oplus pw \oplus pw$. Then, the server uses the recovered R_A to obtain modified new password $newpw \oplus X$ from $R_A \oplus newpw \oplus X$ by computing $R_A \oplus newpw \oplus X \oplus R_A$.

In step 3*, an attacker can replace $H(K_A, R_B) \oplus newpw$ with $H(K_A, R_B) \oplus newpw \oplus X$ by using the chosen random number in Step 1*. After receiving the replaced messages $H(K_A, R_B) \oplus newpw \oplus X$, the server can retrieve $H(K_A, R_B)$ from $H(K_A, R_B) \oplus newpw \oplus X$ using the obtained value $newpw \oplus X$ in the Step 1* and checks whether $H(K_A, R_B)$ is equal to $H(K_B, R_B)$ holds or not. Because it holds, the server will pass the authentication and update a new password as $newpw \oplus X$. If value $newpw \oplus X$ is not equal to the client's new password $newpw$, all subsequent login requests of that client will be rejected until that client has re-registered with the server. Therefore, Chang et al.'s protected password change protocol is insecure against Denial-of-Service attacks.

4.2 Stolen-Verifier Attack on Chang et al.'s Schemes

Servers are always the target of attacker, because numerous clients' secrets are stored in their databases. The client password stored in the server can be eavesdropped and then used to impersonate as the original client. Chang et al. do not explain stolen-verifier attacks, where obtaining the client password pw stored in a server can allow an illegitimate client to login to the server as a legitimate client.

In Chang et al.'s protected password change protocol (which includes Chang et al.'s simple authenticated key agreement protocol), if an attacker has stolen the password pw from the server, he or she can choose a random number a', computes $R'_A = g^{a'} \bmod p$, choose a new password $newpw'$, and uses R'_A to compute client password digest $R'_A \oplus pw$ and client new password digest $R'_A \oplus newpw'$ in Step 1*. Then the attacker can send its as a login request to the server and can impersonate the original client. Therefore, Chang et al.'s schemes is insecure against stolen-verifier attacks.

5 Proposed Schemes

This section proposes an improved simple authenticated key agreement protocol and protected password change protocol to overcome the above mentioned problems inherent in Chang et al.'s scheme. In the proposed scheme, the server stores $vpw = (H(id, pw) \oplus K) + K$ using the server's secret key K instead of pw for each client in the database to overcome the stolen-verifier attack.

5.1 Proposed Simple Authenticated Key Agreement Protocol

The proposed simple authenticated key agreement protocol works as follows:

Step 1 Client → Server: $id||R_A \oplus H(id, pw)$
The user submits his id and pw to the client. The client then chooses a random number $a \in [1, q-1]$, computes $R_A = g^a \bmod p$, and then sends $id||R_A \oplus H(id, pw)$ as a login request to the server.

Step 2 Server → Client: $R_B||H(K_B, R_A)$
After receiving $id||R_A \oplus H(id, pw)$, the server retrieves R_A from $R_A \oplus H(id, pw)$ by computing $R_A \oplus H(id, pw) \oplus (vpw - K) \oplus K$. Then, the server chooses a random number $b \in [1, q-1]$ and computes $R_B = g^b \bmod p$ and $K_B = (R_A)^b = g^{ab} \bmod p$. Then, the server uses its own K_B and the recovered R_A to compute $H(K_B, R_A)$. The server then sends $R_B||H(K_B, R_A)$ as the server's authentication token to the client.

Step 3 Client → Server: $id||H(K_A, R_B)$
After receiving $R_B||H(K_B, R_A)$, the client computes $K_A = (R_B)^a = g^{ab} \bmod p$ and $H(K_A, R_A)$, and then verifies the consistency between the retrieved $H(K_A, R_A)$ and the received $H(K_B, R_A)$. If the result is positive, the client computes $H(K_A, R_B)$ and sends this client authentication token $H(K_A, R_B)$ with the id to the server.

Step 4 Server→Client: *Access granted / denied*
After receiving $id||H(K_A, R_B)$, the server computes $H(K_B, R_B)$ using its own copies of K_B and R_B, and then checks whether $H(K_B, R_B) = H(K_A, R_B)$ holds or not. If it holds, the server can ensure the client is legal.

After mutual authentication between the client and the server, $H(K_A) = H(K_B) = H(g^{ab} \bmod p)$ is used as the session key, respectively.

5.2 Proposed Protected Password Change Protocol

The protected password change protocol allows a client to change his or her old password pw to a new password $newpw$. The proposed protected password change protocol works as follows:

Step 1* Client → Server: $id||R_A \oplus H(id, pw)||R_A \oplus H(id, newpw)$
The user submits his or her id and pw to the client. The client then chooses a random number $a \in [1, q-1]$, computes $R_A = g^a \bmod$

p, chooses a new password $newpw$, and uses R_A to compute $R_A \oplus H(id, pw)$ and $R_A \oplus H(id, newpw)$. Finally, the client sends $id||R_A \oplus H(id, pw)||R_A \oplus H(id, newpw)$ as a login request to the server.

Step 2* Server → Client: $R_B||H(K_B, R_A)$
After receiving $id||R_A \oplus H(id, pw)||R_A \oplus H(id, newpw)$, the server retrieves R_A from $R_A \oplus H(id, pw)$ by computing $R_A \oplus H(id, pw) \oplus (vpw - K) \oplus K$. Then, the server uses the recovered R_A to obtain $H(id, newpw)$ from $R_A \oplus H(id, newpw)$ by computing $R_A \oplus H(id, newpw) \oplus R_A$. Then, the server chooses a random number $b \in [1, q-1]$ and computes $R_B = g^b \bmod p$ and $K_B = (R_A)^b = g^{ab} \bmod p$. Then, the server uses its own K_B and the recovered R_A to compute $H(K_B, R_A)$. The server sends $R_B||H(K_B, R_A)$ as the server's authentication token to the client.

Step 3* Client → Server: $id||H(K_A, R_B, H(id, newpw))$
After receiving $R_B||H(K_B, R_A)$, the client computes $K_A = (R_B)^a = g^{ab} \bmod p$ and $H(K_A, R_A)$, then verifies the consistency between the retrieved $H(K_A, R_A)$ and the received $H(K_B, R_A)$. If the result is positive, the client computes $H(K_A, R_B, H(id, newpw))$ and sends this client authentication token $H(K_A, R_B, H(id, newpw))$ with the id to the server.

Step 4* Server→Client: *Access granted / denied*
After receiving $id||H(K_A, R_B, H(id, newpw))$, the server computes the hash value $H(K_B, R_B, H(id, newpw))$ using its own copies of K_B, R_B and the recovered $H(id, newpw)$ in the Step 1*, and then checks whether $H(K_B, R_B, H(id, newpw)) = H(K_A, R_B, H(id, newpw))$ holds or not. If it holds, the server can ensure the client is legal and replaces vpw with $(H(id, newpw) \oplus K) + K$.

After mutual authentication between the client and the server, $H(K_A) = H(K_B) = H(g^{ab} \bmod p)$ is used as the session key, respectively.

6 Security Analysis

This subsection provides the security analysis of the proposed schemes. First, we define the security terms [10] needed for security analysis of the proposed schemes as follows:

Definition 1. *A weak secret (password) is a value of low entropy $W(k)$, which can be guessed in polynomial time.*

Definition 2. *A strong secret key (K) is a value of high entropy $H(k)$, which cannot be guessed in polynomial time.*

Definition 3. *The discrete logarithm problem (DLP) is explained by the following: Given a prime p, a generator g of Z_p^*, and an element $\beta \in Z_p^*$, find the integer α, $0 \le \alpha \le p-2$, such that $g^\alpha \equiv \beta (\mathrm{mod} p)$.*

Definition 4. *The Diffie-Hellman problem (DHP) is explained by the following: Given a prime p, a generator g of Z_p^*, and elements $g^a (\mathrm{mod} p)$ and $g^b (\mathrm{mod} p)$, find $g^{ab} (\mathrm{mod} p)$.*

Definition 5. *A secure one-way hash function $y = H(x)$ is one where given x, computing y is easy and given y, computing x is hard.*

Here, seven security properties: replay attack, password guessing attack, stolen-verifier attack, server spoofing attack, Denial-of-Service attack, mutual authentication, and perfect forward secrecy, must be considered for the proposed schemes. Under the above definitions, the following theorems are used to analyze seven security properties in the proposed schemes.

Theorem 1. *The proposed schemes can resist the replay attack.*

Proof. The attacker intercepts $id||R_A \oplus H(id, pw)$ sent by the client in Step 1 and uses it to impersonate the client when sending the next login message. However, he/she has no ability to make a correct response $id||H(K_A, R_B)$ in Step 3 because the random challenge R_A and R_B separately generated by the client and server are different every time. On the other hand, since the messages sent by the server and the client are different, the attacker cannot intercept any messages between them and then replay them to the other parties. Furthermore, obtaining $R_A = g^a \bmod p$ and $R_B = g^b \bmod p$ is computationally infeasible, as it is a discrete logarithm problem. Therefore, without knowing R_A and R_B, the attacker cannot impersonate the client or the server.

Theorem 2. *The proposed scheme can resist the password guessing attacks.*

Proof. On-line guessing attacks can be prevented by letting the server take appropriate intervals between trials. As described in Definition 1, weak passwords with low entropy are easily guessed in off-line guessing attacks. To avoid this problem, there must be no verifiable information on passwords in message exchanges. In the improved schemes, the password pw is protected by the client's random integer R_A. As such, no one can reveal the pw from the client's login message $id||R_A \oplus H(id, pw)$ without knowing the client's random integer R_A. If the attacker wants to guess the client's password, he or she first must guess a password pw' and then finds $R_A = R_A \oplus H(id, pw) \oplus H(id, pw')$. However, the attacker has to break the discrete logarithm problem and Diffie-Hellman problem to find $R_A = g^a \bmod p$ in Step 1 and $K_B = g^{ab} \bmod p$ in Step 2, respectively. Hence, without knowing R_A and K_B, the attacker cannot verify the correctness of the guessed password by checking $R_A \oplus H(id, pw) = R_A \oplus H(id, pw')$ in Step 1 and $H(K_A, R_B) = H(K'_A, R_B)$ in Step 3, respectively. For the same reason, the attacker cannot guess session key $H(K_A) = H(K_B) = H(g^{ab} \bmod p)$ from the server's response message $R_B||H(K_B, R_A)$ in Step 2 or from the client's response message $id||H(K_A, R_B)$ in Step 3 because $H(\cdot)$ is a secure one-way hash function.

Theorem 3. *The proposed scheme can resist the stolen-verifier attack.*

Proof. Servers are always the target of attacks. An attacker may acquire $vpw = (H(id, pw) \oplus K) + K$ stored in the server. However, without knowing the server's strong secret key K, the attacker cannot forge a login request to pass the authentication, as $H(id, pw)$ is hidden in $vpw = (H(id, pw) \oplus K) + K$ using the server's

strong secret key K. Thus, the correctness of the guessed password cannot be verified by checking $(H(id, pw') \oplus K') + K' = vpw$, where pw' is the guessed client's password and K' is the guessed server's strong secret key.

Theorem 4. *The proposed scheme can resist the server spoofing attack.*

Proof. The proposed schemes use the client's password $H(id, pw)$ to ensure that only the real server can obtain R_A from the client's login message $id||R_A \oplus H(id, pw)$. After verifying the identity of the client, the server sends a correct response $R_B||H(K_B, R_A)$ to the client to achieve mutual authentication in Step 2. Due to the discrete logarithm problem and Diffie-Hellman problem, an illegal client cannot compute Diffie-Hellman key $K_A = K_B = g^{ab} \bmod p$ from $R_B||H(K_B, R_A)$ and then make a correct response $id||H(K_A, R_B)$ in Step 3.

Theorem 5. *The proposed protected password change protocol can resist a Denial-of-Service attacks.*

Proof. In step 1* of the proposed protected password change protocol, an attacker can replace new password digest $R_A \oplus H(id, newpw)$ with forged client password digest $R_A \oplus H(id, newpw) \oplus X$, in which X is a random number chosen by the attacker. In Step 2*, after receiving the replaced messages $id||R_A \oplus H(id, pw)||R_A \oplus H(id, newpw) \oplus X$, the server retrieves R_A from $R_A \oplus H(id, pw)$ by computing $R_A \oplus H(id, pw) \oplus (vpw - K) \oplus K$. Then, the server uses the recovered R_A to obtain replaced new password verifier $H(id, newpw) \oplus X$ from $R_A \oplus H(id, newpw) \oplus X$ by computing $R_A \oplus H(id, newpw) \oplus X \oplus R_A$. However, in the proposed protocol, a check item $H(K_A, R_B, H(id, newpw))$ for new password is added in Step 3*. The server then updates replaced password verifier $H(id, newpw) \oplus X$ only if the computed hash value $H(K_B, R_B, H(id, newpw))$ is equivalent to the received $H(K_A, R_B, H(id, newpw))$. But an attacker cannot computes this session key $K_A = g^{ab} \bmod p$ in hashed value $H(K_A, R_B, H(id, newpw))$ because the discrete logarithm problem, the Diffie-Hellman problem, and a secure one-way hash function.

Theorem 6. *The proposed scheme provides mutual authentication.*

Proof. The proposed schemes use the Diffie-Hellman key exchange algorithm [1] to provide mutual authentication; then the key is explicitly authenticated by a mutual confirmation session key, $K_A = K_B = g^{ab} \bmod p$.

Theorem 7. *The proposed scheme provides the perfect forward secrecy.*

Proof. In the proposed schemes, since the Diffie-Hellman key exchange algorithm is used to generate a session key $K_A = K_B = g^{ab} \bmod p$, perfect forward secrecy is ensured, as an attacker with a compromised client's password pw is only able to obtain the $R_A = g^a \bmod p$ and $R_B = g^b \bmod p$ from an earlier session. In addition, it is also computationally infeasible to obtain the session key g^{ab} from g^a and g^b, as it is a discrete logarithm problem and Diffie-Hellman problem.

7 Conclusions

The current paper demonstrated that Chang et al.'s simple authenticated key agreement and protected password change protocol is vulnerable to stolen-verifier attack and also demonstrated that their protected password change protocol suffers from a Denial-of-Service attacks. We present an improved scheme to isolate such problems. In contrast to Chang et al.'s schemes, the proposed schemes can securely update user passwords without a complicated process, while also providing more security. Therefore, the proposed protocols are more secure than other SAKA-like schemes and protected password change protocols.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments in improving our manuscript. This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

References

1. Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transaction on Information Theory. Vol. IT-22. No. 6. (1976) 644-654
2. Schneier, B.: Applied Cryptography-Protocols. Algorithms and Source Code in C. 2nd edi. John Wiley & Sons Inc. (1995)
3. Seo, D.H., Sweeney, P.: Simple Authenticated Key Agreement Algorithm. Electronics Letters. Vol. 35. No. 13. (1999) 1073-1074
4. Bellovin, S., Merritt, M.: Encrypted Key Exchange: Password-based Protocols Secure Against Dictionary Attacks. Proc. of IEEE Conf. on Research in Security and Privacy. (1992) 72-84
5. Yeh, H.T., Sun, H.M.: Simple Authenticated Key Agreement Protocol resisant to Password Guessing Attacks. ACM SIGOPS Operation Systems Review. Vol. 36. No. 4. (2002) 14-22
6. Kobara, K., Imai, H.: Pretty-simple Password-authenticated Key-exchange Protocol Proven to be Secure in the Standard Model. IEICE Transactions on Fundamentals. Vol. E85-A. No. 10. (2002) 2229-2237
7. Chang, T.Y., Yang, W.P., Hwang, M.S.: Simple Authenticated Key Agreement and Protected Password Change Protocol. Computers & Mathematics with Applications. Vol. 49. No. 5-6. (2005) 703-714
8. Ballare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure Against Dictionary Attack. In Proc. of EUROCRYPT 2000. LNCS 1807. (2000) 139-155
9. Lin, C.L., Hwang, T.: A Password Authentication Scheme with Secure Password Updating. Computers & Security. Vol. 22. No. 1. (2003) 68-72
10. Menezes, A.J., Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptograph. CRC Press. New York. (1997)