

## Short Paper

---

# A Secure Hash-Based Strong-Password Authentication Protocol Using One-Time Public-Key Cryptography

MINHO KIM AND ÇETIN KAYA KOÇ\*

*Department of Computer Science*

*Korea Air Force Academy*

*Sangsu, 363-849, South Korea*

*E-mail: mhkim@afa.ac.kr*

\**School of Electrical Engineering and Computer Science*

*Oregon State University*

*Corvallis, Oregon 97331, U.S.A.*

*E-mail: koc@eecs.oregonstate.edu*

Secure communication is an important issue in networks and user authentication is a very important part of the security. Several strong-password authentication protocols have been introduced, but there is no fully secure authentication scheme that can resist all known attacks. We propose enhanced secure schemes with registration and login protocols, and add the “forget password” and password/verifier change protocols. We show that our scheme is more secure against guessing, stolen-verifier, replay, denial-of-service, and impersonation attacks than previously introduced protocols.

**Keywords:** password authentication, forget password, password/verifier change, guessing attack, stolen-verifier attack, replay attack, denial-of-service attack, impersonation attack

## 1. INTRODUCTION

Password-based authentication mechanisms are the simplest and most convenient way to have a user authenticated in order to provide services of a computing or communication system to a pre-selected group of authorized users. These mechanisms are less costly than the biometric methods of authentication, such as fingerprint, iris scan, voice signature, *etc.* A generic password-based authentication system usually hashes the password of the user with the help of hash function derived from a secret-key cryptographic function, such as MISTY, DES, or FEAL [13, 14, 19]. The hashed password is stored on the server in order to preclude stealing the password by the adversary.

Unfortunately, there are two limitations in password-based authentication systems: (1) the user must submit the bare password at every authentication, and (2) the transmitted password could be stolen by wiretapping or sniffing. One of the remedying is found the use of one-time password method by Lamport [9], but there are some practical diffi-

---

Received July 6, 2006; revised October 17, 2006; accepted March 14, 2007.  
Communicated by Ja-Ling Wu.

culties in implementing this method, such as the problems of high overhead and password resetting. Another related method is CINON [17] which solves these problems, but it requires two random numbers generated by the user, which must be stored by the user in some sort of mobile memory device. On the other hand, the PERM (Privacy Enhanced Information Reading and Writing Management) Protocol [18] stores one random number at the host, which is sent to the user for authentication. However, there are some security flaws in such a system; the adversary can launch a man-in-the-middle attack if he can obtain the logs of two consecutive sessions.

The SAS protocol proposed in [16] is a simple strong-password authentication scheme, which is superior to several well-known schemes. But, it was shown in [11] that the SAS protocol is vulnerable to the replay attack and the denial of service attack. The OSPA (Optimal Strong-Password Authentication) Protocol given in [11] was claimed to be secure against stolen-verifier attacks, replay attacks, and the denial of service attacks. Nevertheless, it was shown in [1] the SAS and OSPA protocols cannot resist to the stolen-verifier attack as claimed. Also, an impersonation attack was described in [20] on the OSPA method without an active attack on the server. Later on, an enhanced OSPA protocol was introduced in [12], which resists to the guessing, replay, impersonation, and stolen-verifier attacks. However, it was shown in [8] that the protocol is still vulnerable to replay and denial-of-service attacks. Furthermore, these two simple attacks can easily be launched without compromising the server in advance.

Recently, a hash-based strong-password authentication scheme was described in [3], which withstands several attacks, including replay, password-file compromise, denial-of-service, and insider attacks. However, Kim-Koç [6] showed that Ku's scheme [3] is still vulnerable to stolen-verifier, denial-of-service, replay, and impersonation attacks.

The Lee-Li-Hwang (LLH) authentication scheme [10] was proposed to circumvent the guessing attack in the Peyravian-Zunic (PZ) password scheme [15]. However, Yoon, Ryu, and Yoo (YRY) [21] discovered that the LLH scheme still suffers from the denial of service attack, and proposed an enhancement for the LLH scheme to solve its security problems. More recently, Ku, Chiang, and Chang (KCC) [4] demonstrated that the YRY scheme is vulnerable to off-line guessing and stolen-verifier attacks. Kim-Koç showed that the YRY scheme is also vulnerable to the denial-of-service attack. Furthermore, it was also claimed in [4] that the YRY scheme cannot achieve backward secrecy. Kim-Koç showed that this claim is not entirely valid [7].

Most of the previous articles deal with registration and login phases. However, we propose enhanced secure schemes with registration and login protocols, and add the "forget password" and password/verifier change protocols. Firstly, we give the basic definitions of these attacks. The remainder of this paper is organized as follows. In sections 2 and 3, we describe the hash-based strong-password authentication scheme introduced in [3], and a hash-based secure user authentication scheme was described in [21], and then explain the details of Kim-Koç's attacks. In section 4, we propose our scheme with four protocols. In section 5, we will briefly analyze how the proposed scheme is secure against guessing, stolen-verifier, replay, denial-of-service, and impersonation attacks. Finally, we shall give a brief conclusion in section 6.

## 2. KU'S SCHEME AND ATTACK

### 2.1 Review of the Ku Scheme

We introduce the notation used to describe the protocols below and explain the detailed steps of both of these protocols.

#### 2.1.1 Notations

- $U$  denotes the User,  $C$  denotes the Client,  $S$  denotes the Server, and  $A$  denotes the Adversary.
- $E_{S_{pu}}$  denotes the encryption with the public key of the server.
- $D_{S_{pr}}$  denotes the decryption with the private key of the server.
- $h$  denotes a cryptographic hash function, such that  $h(m)$  means the message  $m$  is hashed once, while  $h^2(m)$  means it is hashed twice, *i.e.*,  $h^2(m) = h(h(m))$ . Furthermore,  $h(a, b)$  denotes the hash of concatenated  $a$  and  $b$ , *i.e.*,  $h(a, b) = h(a || b)$ .
- $N$  denotes an integer starting from 1 since  $U$ 's initial registration.
- $P$  denotes the strong password of  $U$ .
- $K_u$  is a random generated key selected by  $U$ .
- $K_s$  denotes the secret-key of  $S$ .
- $R_c$  and  $R_s$  denote random numbers generated by Client and Server, respectively.
- $T$  denotes the most recent time  $U$  initially registered or re-registered at  $S$ .
- $T_s$  denotes the timestamp.
- $UID$  denotes the identification of the user.
- $\oplus$  denotes the bitwise XOR operation, and  $||$  denotes the concatenation.
- $Auth_Q/Auth_A$  denotes the authentication question/answer for the registration, "forget password" and password/verifier change protocols.
- The expression  $A \rightarrow B: X$  means  $A$  sends the message  $X$  to  $B$  via an insecure channel.
- The expression  $A \Rightarrow B: X$  means  $A$  sends the message  $X$  to  $B$  via a secure channel.

The hash-based strong-password authentication scheme described in [3] comes with two protocols: the registration protocol and the login protocol.

#### 2.1.2 Registration protocol

This protocol is invoked whenever  $U$  initially registers or re-registers to  $S$ .

- R1.  $U$  sends his registration request to  $S$ .
- R2.  $S \rightarrow U: N, T$ .  
 $S$  sets  $T$  as the currently value of the time. If this is  $U$ 's initial registration,  $S$  sets  $N = 1$ , otherwise  $S$  sets  $N = N + 1$ . Next,  $S$  sends  $N$  and  $T$  to  $U$ .
- R3.  $U \Rightarrow S: h^2(S || P || N || T)$ .  
 $U$  computes the verifier  $h^2(S || P || N || T)$  and sends it to  $S$ .
- R4.  $S$  computes the user storage key  $K_U^{(T)} = h(U || h(K_S || T))$  and the sealed verifier  $sv^{(N)} = h^2(S || P || N || T) \oplus K_U^{(T)}$ , and then he stores  $sv^{(N)}$ ,  $N$ , and  $T$  in the password file.

### 2.1.3 Login protocol

This protocol is invoked whenever  $U$  logs in to  $S$ .

L1.  $U$  sends his login request to  $S$ .

L2.  $S \rightarrow U: r, n, t$ .

$S$  selects a random nonce  $r$  and retrieves the values of  $n = N$  and  $t = T$  from  $S$ 's password file.

L3.  $U \rightarrow S: c_1, c_2, c_3$ .

$U$  sends  $c_1, c_2$ , and  $c_3$  to  $S$ , where

$$\begin{aligned} c_1 &= h^2(S \| P \| n \| t) \oplus h(S \| P \| n \| t), \\ c_2 &= h(S \| P \| n \| t) \oplus h^2(S \| P \| n + 1 \| t), \\ c_3 &= h(h^2(S \| P \| n + 1 \| t) \| r). \end{aligned}$$

L4.  $S$  computes  $K_U^{(t)} = h(U \| h(K_S \| t))$ , and then derives  $h^2(S \| P \| n \| t)$  from the stored sealed verifier  $sv^{(n)}$  using

$$h^2(S \| P \| n \| t) = sv^{(n)} \oplus K_U^{(t)}.$$

Then,  $S$  computes  $u_1$  and  $u_2$  using

$$\begin{aligned} u_1 &= c_1 \oplus h^2(S \| P \| n \| t) = h(S \| P \| n \| t), \\ u_2 &= c_2 \oplus u_1 = h^2(S \| P \| n + 1 \| t). \end{aligned}$$

If the equalities  $h(u_1) = h^2(S \| P \| n \| t)$  and  $h(u_2 \| r) = c_3$  hold, then  $S$  authenticates  $U$ . Otherwise,  $S$  rejects  $U$ 's login request and terminates the session.

After a successful authentication,  $S$  computes a new sealed-verifier using

$$sv^{(n+1)} = u_2 \oplus K_U^{(t)} = h^2(S \| P \| n + 1 \| t) \oplus K_U^{(t)},$$

and replaces  $sv^{(n)}$  with  $sv^{(n+1)}$ , and sets  $N = n + 1$  for  $U$ 's next login protocol. The value of  $T$  is unchanged, *i.e.*,  $T = t$ .

### 2.2 Attack on the Ku Scheme

Kim-Koç [6] devise an attack assumption that the adversary steals a copy of user's password-verifier  $h^2(S \| P \| N \| T)$ . Such scenarios are considered in other paper [1].

The second assumption they make is that  $A$  is capable of blocking the communication from  $U$  to  $S$ . After having stolen a copy of the password verifier,  $A$  launches an attack whenever it can block communication.

Therefore, Kim-Koç attack assumes that a stolen-verifier attack (by obtaining a copy of the password verifier) and a denial-of-service attack (by blocking the communication from  $U$  to  $S$ ) have succeeded. They then show that under these two assumptions (attacks), the attacker can now successfully login to the system using replay, impersonate the user, and thus succeed in the impersonation attack.

1.  $A$  steals a copy of  $U$ 's password-verifier  $h^2(S\|P\|N\|T)$ .
2. During the  $U$ 's  $n$ th login process,  $A$  monitors the communication channel, and then he sees the request  $U$  made to  $S$  and the values  $r$ ,  $n$ , and  $t$  sent by  $S$ . Next,  $A$  captures the values of  $c_1$ ,  $c_2$ , and  $c_3$  sent by  $U$  to  $S$  and blocks the communication channel from  $U$  to  $S$ . These values are not reaching to  $S$  by blocking communication.
3.  $A$  computes  $h(S\|P\|n\|t)$  and  $h^2(S\|P\|n+1\|t)$  with the help of the captured values  $c_1$ ,  $c_2$ , and the previously stolen password-verifier  $h^2(S\|P\|N\|T)$  as

$$\begin{aligned} h(S\|P\|n\|t) &= c_1 \oplus h^2(S\|P\|n\|t), \\ h^2(S\|P\|n+1\|t) &= c_2 \oplus h(S\|P\|n\|t), \end{aligned}$$

where  $N = n$  and  $T = t$ .

4. Next,  $A$  sends  $c_1$ ,  $c_2$ , and  $c_3$  to  $S$ .
5. After receiving this message,  $S$  retrieves  $t$  from the password file and computes

$$K_U^{(t)} = h(U\|h(K_S\|t))$$

and then uses  $K_U^{(t)}$  to compute the verifier  $h^2(S\|P\|n\|t)$  with the help of the stored sealed verifier  $sv^{(n)}$  as

$$h^2(S\|P\|n\|t) = sv^{(n)} \oplus K_U^{(t)}.$$

6. Next,  $S$  computes

$$\begin{aligned} u_1 &= c_1 \oplus h^2(S\|P\|n\|t) = h(S\|P\|n\|t), \\ u_2 &= c_2 \oplus u_1 = h^2(S\|P\|n+1\|t). \end{aligned}$$

If  $h(u_1) = h^2(S\|P\|n\|t)$  and  $h(u_2\|r) = c_3$  hold,  $S$  supposed to authenticate the sender. Since these equalities will hold,  $S$  authenticates  $A$  as being  $U$ . Therefore,  $S$  allows the attacker  $A$  to login.

7. After this successful login,  $S$  updates the sealed verifier according to the step L4 of login protocol. Therefore, the following will be executed by  $S$ .  $S$  computes

$$sv^{(n+1)} = u_2 \oplus K_U^{(t)} = h^2(S\|P\|n+1\|t) \oplus K_U^{(t)},$$

and replaces  $sv^{(n)}$  with  $sv^{(n+1)}$ , and then he sets  $N = n + 1$  for  $U$ 's next login protocol. The value of  $T$  is unchanged, *i.e.*,  $T = t$ .

At the end of step 6, the adversary has successfully logged into the system impersonating the legitimate user. It can now launch other attacks within the system or access to sensitive documents. If the user logs in after the attacker does, it may not be possible to discover that the attacker has logged into the system impersonating the user, unless the user checks the login records. Until the time when the user or the system managers discover the attacker's successful login, the attacker can continue to impersonate the user.

### 3. YRY'S SCHEME AND ATTACK

#### 3.1 Review of the YRY Scheme

A hash-based secure user authentication scheme was described in [21]. The scheme has 3 phases: Registration phase, User authentication phase, and Change password phase.

##### 3.1.1 Registration phase

This registration phase is performed only once when a new user wants to join the system. On the other hand, the authentication phase is executed whenever the user wants to login to the system. The procedures of this phase are as follows:

- R1.  $U \Rightarrow S$ :  $UID, HPW$   $U$  randomly chooses  $UID$  and  $P$ , and then calculates a password verifier  $HPW = h(UID, P)$ .  
 R2.  $S$  stores  $UID$  and  $HPW$  in the verification table.

##### 3.1.2 User authentication phase

In this phase, the user logs in to a server for accessing resources and the server authenticates the user. The procedures of this phase are as follows:

- A1.  $C \rightarrow S$ :  $UID, R_c \oplus HPW, h(R_s)$ .  
 $U$  enters  $UID$  and  $P$  to  $C$ .  $C$  computes  $HPW = h(UID, P)$  and randomly chooses a number  $R_c$ , and then computes the hash value  $h(R_c)$ . Next,  $C$  sends  $UID, R_c \oplus HPW$ , and  $h(R_c)$  to  $S$ .  
 A2.  $S \rightarrow C$ :  $R_s \oplus HPW, h(R_c, R_s)$ .  
 $S$  retrieves the  $U$ 's password verifier  $HPW$  from the verification table, and then obtains  $R_c$  by computing  $(R_c \oplus HPW) \oplus HPW$ . Next,  $S$  verifies the equality of the computed  $h(R_c)$  with the obtained  $R_c$  and the received  $h(R_c)$ . If they are equal,  $S$  randomly generates a number  $R_s$ , and then computes  $R_s \oplus HPW, h(R_c, R_s)$ , and  $AUTH^* = h(HPW, R_c, R_s)$ . Next,  $S$  sends  $R_s \oplus HPW$  and  $h(R_c, R_s)$  to  $C$ .  
 A3.  $C \rightarrow S$ :  $UID, AUTH$ .  
 $C$  retrieves  $R_s$  by using  $(R_s \oplus HPW) \oplus HPW$  and computes  $h(R_c, R_s)$ . If the computed and received  $h(R_c, R_s)$  are equal,  $C$  computes  $AUTH = h(HPW, R_c, R_s)$  and sends  $UID$  and  $AUTH$  to  $S$ .  
 A4.  $S$  compares  $AUTH$  with  $AUTH^*$ . If they are equal,  $S$  authenticates  $U$ . Otherwise,  $S$  rejects  $C$ 's request and terminates the session.

##### 3.1.3 Change password phase

The change password phase is invoked whenever client wants to change its password  $P$  with a new one, say  $NewP$ . The procedures of this phase are given below. Note that steps C1 and C2 are the same as the ones in the user authentication phase.

- C3.  $C \rightarrow S$ :  $UID, AUTH, Mask, V_{Mask}$ .

$C$  retrieves  $R_s$  by using  $(R_s \oplus HPW) \oplus HPW$  and computes  $h(R_c, R_s)$ . If the computed and received  $h(R_c, R_s)$  are equal, then  $C$  computes

$$\begin{aligned} NewHPW &= h(UID, NewP), \\ AUTH &= h(HPW, R_c, R_s), \\ Mask &= NewHPW \oplus h(HPW, R_c + 1, R_s), \\ V_{Mask} &= h(NewHPW, R_s). \end{aligned}$$

Then,  $C$  sends  $UID, AUTH, Mask,$  and  $V_{Mask}$  to  $S$ .

- C4.  $S$  retrieves the  $U$ 's  $HPW$  from the verification table. If  $AUTH = AUTH^*$ ,  $S$  accepts  $C$  to change the  $U$ 's password, and then obtains new password verifier  $NewHPW$  as  $NewHPW = Mask \oplus h(HPW, R_c + 1, R_s)$ . Next,  $S$  calculates  $h(NewHPW, R_s)$  and compares it with  $V_{Mask}$ . If they are equal,  $S$  replaces the old  $HPW$  with the new password verifier  $NewHPW$  in the verification table. Otherwise,  $S$  rejects  $C$ 's change password request and terminates the session.

### 3.2 Denial of Service Attack on the YRY Scheme

The adversary is able to prevent the client from logging in during the user authentication phase or changing its password  $P$  with  $NewP$  in the change password phase by making the server reject all login requests and change password requests. As mentioned in the impersonation attack, the adversary can replace all information that were related to the login and change password phases.

From	To
$R_c$ $NewP$ $NewHPW = h(UID, NewP)$ $AUTH = h(HPW, R_c, R_s)$ $Mask = NewHPW \oplus h(HPW, R_c + 1, R_s)$	$R_a$ $P_a$ $NewHPW^* = h(UID, P_a)$ $AUTH^* = h(HPW, R_a, R_s)$ $Mask^* = NewHPW^* \oplus h(HPW, R_c + 1, R_s)$

After receiving the replaced message, if the user tries to login the server, he will be rejected since both the password and the password verifier were changed.

- DoS1. In the user authentication phase,  $U$  enters  $UID$  and  $P$  to  $C$ .  $C$  computes  $HPW = h(UID, P)$  and randomly chooses a number  $R_c$ , and then computes  $h(R_c)$ . Next,  $C$  sends  $UID, R_c \oplus HPW,$  and  $h(R_c)$  to  $S$  in step A1. Since  $S$  retrieves  $A$ 's new password verifier  $NewHPW^* = h(UID, P_a)$  from the verification table, he obtains  $R_c^*$  that is different from  $R_c$ ,  $R_c^*$  was obtained by computing  $(R_c \oplus HPW) \oplus NewHPW^*$ . Next,  $S$  verifies the equality of the computed  $h(R_c)$  and the received  $h(R_c^*)$ . They are not equal. Therefore,  $S$  rejects  $C$ 's request.
- DoS2. Even though this attack happened after  $U$ 's successful login, the problem is the same as in the user change password phase since the request in step C1 is the same as in step A1.

- DoS3. If this attack happened after step C2,  $C$  computes  $NewHPW = h(UID, NewP)$ ,  $AUTH' = h(HPW, R_c, R_s)$ ,  $Mask = NewHPW \oplus h(HPW, R_c + 1, R_s)$ , and  $V_{Mask} = h(NewHPW, R_s)$ , and then  $C$  sends  $UID$ ,  $AUTH$ ,  $Mask$ , and  $V_{Mask}$  to  $S$  in step C3. At this moment,  $AUTH^* = h(HPW, R_a, R_s)$  is not equal to  $AUTH' = h(HPW, R_c, R_s)$  that  $S$  computed in step C2, not in step C3. Therefore,  $S$  rejects  $C$ 's to change  $U$ 's password.
- DoS4. If this attack happened after step C3,  $C$  computes  $NewHPW$ ,  $AUTH$ ,  $Mask$ , and  $V_{Mask}$  the same as step DoS3, and then  $C$  sends  $UID$ ,  $AUTH$ ,  $Mask$ , and  $V_{Mask}$  to  $S$  in step C3.  $AUTH' = h(HPW, R_c, R_s)$  is equal to  $AUTH = h(HPW, R_c, R_s)$  that  $S$  computes in step C2, accordingly,  $S$  accepts  $C$  to change the  $U$ 's password. However,  $S$  obtains a different password verifier as  $NewHPW' = Mask \oplus h(HPW, R_a + 1, R_s)$ , which is not equal to  $U$ 's new verifier  $NewHPW$ , since  $R_c$  was already changed with  $R_a$  by  $A$ . After that,  $S$  computes  $h(NewHPW', R_s)$  and compares it with  $V_{Mask}$ . The value of  $h(NewHPW', R_s)$  is not equal to  $V_{Mask} = h(NewHPW, R_s)$ . Consequently,  $S$  rejects  $C$ 's change password request and terminates the session.

For those reason, both the user's authentication and change password requests are rejected until the user has re-registered with the server.

The adversary can interrupt or lock the account of any user. In addition, this attack works even if  $P$  is a strong password.

### 3.3 KCC Impersonation Attack with Stolen-Verifier

Suppose that the adversary has stolen the verifier  $HPW = h(UID, P)$  of the user from the server. The adversary can compute  $R_c$ ,  $(R_c \oplus HPW) \oplus HPW$  by XORing, and then he can get more information in sequence, computing  $h(R_c)$ ,  $R_s$  using  $(R_s \oplus HPW) \oplus HPW$ ,  $h(R_c, R_s)$ , and  $AUTH^* = h(HPW, R_c, R_s)$ . After that, the adversary has all the information he needs to login into the server. If the adversary obtains an  $HPW$  through the stolen-verifier attack, he can then perform the following:

- B1.  $A$  can make a random generated number  $R_a$  to compute  $R_a \oplus HPW$  and  $h(R_a)$ . He sends  $UID$ ,  $R_a \oplus HPW$ , and  $h(R_a)$  to the server in step A1.
- B2.  $S$  retrieves the  $R_a = (R_a \oplus HPW) \oplus HPW$  by XORing, and then  $S$  verifies the equality of the computed  $h(R_a)$  and received  $h(R_a)$ . If they are equal,  $S$  randomly generates a number  $R_s$  and computes  $R_s \oplus HPW$ ,  $h(R_a, R_s)$ , and  $AUTH^* = h(HPW, R_a, R_s)$ .  $S$  sends  $R_s \oplus HPW$  and  $h(R_a, R_s)$  to  $A$  in step A2.
- B3.  $A$  retrieves  $R_s$  using  $(R_s \oplus HPW) \oplus HPW$  and computes  $h(R_a, R_s)$ . Next, if the computed and received  $h(R_a, R_s)$  are equal,  $A$  computes  $AUTH = h(HPW, R_a, R_s)$  and sends  $UID$  and  $AUTH$  to  $S$  in step A3.
- B4.  $S$  compares  $AUTH$  with  $AUTH^*$ . If they are equal,  $S$  authenticates  $A$  in step A4. After that,  $A$  can impersonate  $U$ .
- Additionally, this attack can be adapted on the change password phase in the same way. This is described as below.
- B5.  $A$  can get the  $R_s$  and  $AUTH = h(HPW, R_c, R_s)$  after steps C1 and C2, and he can then choose his new password  $P_a$  and the random number  $R_a$ . Next,  $A$  computes  $NewHPW$ ,  $Mask$ , and  $V_{Mask}$  with his own  $P_a$  as

$$\begin{aligned}
NewHPW &= h(UID, P_a), \\
Mask &= NewHPW \oplus h(HPW, R_c + 1, R_s), \\
AUTH &= h(HPW, R_a, R_s), \\
V_{Mask} &= h(NewHPW, R_s).
\end{aligned}$$

Then,  $A$  sends  $UID$ ,  $AUTH$ ,  $Mask$ , and  $V_{Mask}$  to  $S$  in step C3.

- B6. After receiving these values,  $S$  retrieves  $U$ 's  $HPW$  from the verification table and compares  $AUTH = AUTH^*$ . If they are equal,  $S$  accepts  $A$  to change the user  $U$ 's password  $P$  with  $A$ 's password  $P_a$ .
- B7.  $S$  obtains the  $A$ 's new password verifier  $NewHPW$  as  $NewHPW = Mask \oplus h(HPW, R_c + 1, R_s)$ , and then  $S$  compares  $h(NewHPW, R_s)$  with  $V_{Mask}$ . Since  $h(NewHPW, R_s) = V_{Mask}$ , it accepts and  $S$  replaces the old  $HPW$  with the new password verifier  $NewHPW$  in the verification table.

Thus, the adversary can impersonate as the user to login and change the password. He can then launch other attacks within the system. If the user logs in after an attack, she may not be able to discover that the attacker has logged into the system impersonating as her, without checking the login records. Until the user or the system manager discovers the attacker's login, the attacker may continue to impersonate the user.

### 3.4 No Lack of Backward Secrecy

It was supposed in [4] that the adversary has stolen the  $HPW$ . If  $C$  detects that  $HPW$  is compromised, it can invoke the password change phase to change password  $P$  with a new one, say  $NewP$ . However, by intercepting the messages transmitted in steps C1 and C2 of the change password phase, the adversary can use the stolen  $HPW$  to retrieve  $R_c$  and  $R_s$ , and compute  $h(HPW, R_c + 1, R_s)$ . Moreover, by intercepting the message transmitted in step C3 of the change password phase, the adversary can use the computed  $h(HPW, R_c + 1, R_s)$  to retrieve  $NewHPW$  from  $Mask (= NewHPW \oplus h(HPW, R_c + 1, R_s))$ .

However, there is a limitation. Even though the adversary intercepts the messages in steps C1 and C2 of the change password phase, he cannot retrieve  $R_c$  and  $R_s$ , because the  $HPW$  is already changed with  $NewHPW$ , and it is not equal to  $HPW$  of the previous stolen verifier. If the adversary wants to get  $R_c$  and  $R_s$  after the change password phase, he needs to obtain the new password verifier. Only then, the adversary cannot compute  $h(HPW, R_c + 1, R_s)$ . Therefore, the claim in [4] is not valid.

## 4. THE PROPOSED SCHEME

There are some cases for our scheme: (1) If someone forgets his password, he should use the "forget password" protocol. (2) If the user just wants to change password, then he should use the password/verifier change protocol. This protocol should be used after a user logs in successfully. (3) Lastly, if someone wants to change user ID and password, then he should use the register protocol.

#### 4.1 Registration Protocol

R1.  $U \rightarrow S: PV = h(K_u \| P) \oplus K_u$ .

$U$  inputs his ID, password, and private key into the client system. The client system computes the user's password verifier  $PV = h(K_u \| P) \oplus K_u$ , and sends it to  $S$  for a registration request.

R2.  $S \rightarrow U: R, Auth_Q$ .

$S$  stores  $PV$  and computes  $R = PV \oplus T_s$ . Next,  $S$  sends  $R$  and  $Auth_Q$  to  $U$ .

R3.  $U \rightarrow S: E_{S_{pu}}(UV, T'_s, U_{id}, K'_u, P', Auth_Q \oplus Auth_A)$ .

$U$  derives  $T'_s$  by XORing  $r$  with  $PV$ , and computes the user's important verifier  $UV = h(K'_u \| P' \| T'_s \| U_{id}) \oplus K'_u$ . Next,  $U$  encrypts  $UV, T'_s, U_{id}, K'_u, P'$  and  $Auth_Q \oplus Auth_A$  with  $S$ 's public key, and sends it to  $S$ .

R4.  $S$  decrypts  $D_{S_{pr}}(E_{S_{pu}}(UV, T'_s, U_{id}, K'_u, P', Auth_Q \oplus Auth_A))$  and derives  $UV, T'_s, U_{id}, K'_u, P'$ , and  $Auth_Q \oplus Auth_A$ .  $S$  computes  $h(K'_u \| P') \oplus K'_u$ .  $S$  then compares  $h(K'_u \| P') \oplus K'_u$  and  $T'_s$  with  $PV$  and  $T_s$ , respectively, that were stored and sent in step R2. If both are equal, then  $S$  stores the sealed-verifier  $SV = h(K'_u \| P' \| U_{id}) \oplus K_{pr}$ ,  $PV$ ,  $UKP = E_{S_{pu}}(U_{id}, K'_u, P')$ , and  $QAK = Auth_Q \oplus Auth_A \oplus K_{pr}$  in his password file, where  $K_{pr}$  is the server's private key.

#### 4.2 Login Protocol

L1.  $U \rightarrow S: PV' = h(K'_u \| P') \oplus K'_u$ .

$U$  inputs his ID, password, and private key into the client system. The client system computes the user's password verifier  $PV' = h(K'_u \| P') \oplus K'_u$ , and sends it to  $S$  for a login request.

L2.  $S \rightarrow U: PV, r_s$ .

$S$  compares  $PV'$  with the  $PV$  that was stored in R2. If they are equal, then  $S$  generates a random nonce  $r_s$ , and then sends  $PV$  and  $r_s$  to  $U$ .

L3.  $U \rightarrow S: L$ .

$U$  compares  $PV'$  with  $PV$ . If they are equal, then  $U$  computes  $h(K_u \| P \| U_{id})$ . Next,  $U$  computes  $L = h(h(K_u \| P \| U_{id}) \oplus PV') \oplus h(K_u \| P \| U_{id}) \oplus PV' \oplus R_s$ , and sends it to  $S$ .

L4.  $S$  derives  $C_1 = h(h(K_u \| P \| U_{id}) \oplus PV') \oplus h(K_u \| P \| U_{id}) \oplus PV'$  by XORing  $L$  with  $r_s$ .  $S$  then computes  $C_2 = SV \oplus K_{pr} = h(K_u \| P \| U_{id})$  using the stored  $SV$  and  $K_{pr}$  in step R4 and  $C_3 = h(C_2 \oplus PV) \oplus C_2 \oplus PV$ . Next,  $S$  checks  $C_1 = C_3$ . If they are equal,  $S$  authenticates  $U$ .

#### 4.3 "Forget Password" Protocol

FP1.  $U \rightarrow S$ : "forget password" request.

FP2.  $S \rightarrow U: Auth'_Q, R_F$ .

$S$  generates a random nonce  $R_F$ , and then sends  $Auth'_Q$  and  $R_F$  to  $U$ .

FP3.  $U \rightarrow S: E_{S_{pu}}(FP, U'_{id})$ .

$U$  computes  $FP = Auth'_Q \oplus Auth'_A \oplus R_F$ , and encrypts it and  $U'_{id}$  with  $S$ 's public key. Next,  $U$  sends  $E_{S_{pu}}(FP, U'_{id})$  to  $S$ .

FP4.  $S \rightarrow U: Auth_A \oplus K'_u, Auth_A \oplus P'$ .

$S$  decrypts  $D_{S_{pr}}(E_{S_{pu}}(FP, U'_{id}))$ , and derives  $D_1 = Auth'_Q \oplus Auth'_A$  by XORing  $FP$  with  $R_F$ .  $S$  then derives  $D_2 = Auth_Q \oplus Auth_A$  by XORing  $QAK$  with  $K_{pr}$  that was stored in step R4. After that,  $S$  checks  $D_1 = D_2$ . If they are equal,  $S$  decrypts  $UKP$ ,  $D_{S_{pr}}(E_{S_{pu}}(U_{id}, K'_u, P'))$  that was stored in step R4. Otherwise,  $S$  rejects this request. Next,  $S$  derives  $K'_u$ ,  $U_{id}$  and  $P'$ , and then checks  $U'_{id} = U_{id}$ . If they are equal,  $S$  computes  $Auth_A \oplus K'_u$  and  $Auth_A \oplus P'$ , and then sends these values to  $U$ . If not,  $S$  terminates this session.

FP5.  $U$  obtains the former password  $P$  and private key  $K_u$  by XORing  $Auth_A \oplus K'_u$  and  $Auth_A \oplus P'$  with  $Auth_A$ .

#### 4.4 Password/Verifier Change Protocol

PC1.  $U \rightarrow S$ : password-change request.

PC2.  $S \rightarrow U$ :  $Auth'_Q, R_C$ .

$S$  generates a random nonce  $R_C$ , and sends  $Auth'_Q$  and  $R_C$  to  $U$ .

PC3.  $U \rightarrow S$ :  $E_{S_{pu}}(W_1, P_{new}, K_{u_{new}}, U_{id_{new}})$ .

$U$  computes  $W_1 = Auth'_Q \oplus Auth'_A \oplus R_C \oplus h(K_u || P || U_{id})$ , and encrypts  $W_1$  and the new values of  $K_{u_{new}}$ ,  $P_{new}$ , and  $U_{id_{new}}$  with  $S$ 's public key. Next,  $U$  sends  $(E_{S_{pu}}(Auth'_Q \oplus Auth'_A \oplus R_C \oplus h(K_u || P || U_{id}), K_{u_{new}}, P_{new}, U_{id_{new}}))$  to  $S$ .

PC4.  $S$  decrypts  $D_{S_{pr}}(E_{S_{pu}}(W_1, K_{u_{new}}, P_{new}, U_{id_{new}}))$ , and obtains  $W_1$ ,  $K_{u_{new}}$ ,  $P_{new}$ , and  $U_{id_{new}}$ .  $S$  then computes  $P_2 = Auth_Q \oplus Auth_A$  by XORing  $QAK$  with  $K_{pr}$  that was stored in step R4 and  $W_3 = SV \oplus K_{pr} = h(K_u || P || U_{id})$  using the stored  $SV$  and  $K_{pr}$  in step R4. Next,  $S$  computes  $W_4 = W_2 \oplus R_C \oplus W_3$  and checks  $W_1 = W_4$ . If they are equal, then  $S$  stores a new  $SV' = h(K_{u_{new}} || P_{new} || U_{id_{new}}) \oplus K_{pr}$ , a new  $PV' = h(K_{u_{new}} || P_{new}) \oplus K_{u_{new}}$ , a new  $UKP' = (E_{S_{pu}}(U_{id_{new}}, K_{u_{new}}, P_{new}))$ , and  $QAK = Auth_Q \oplus Auth_A \oplus K_{pr}$  in his password file.

## 5. SECURITY ANALYSIS

We will briefly demonstrate that the proposed scheme is secure against guessing, stolen-verifier, replay, denial-of-service, and impersonation attacks.

### 5.1 Guessing Attack

By nature, due to the use of a strong password, this scheme is able to resist the off-line guessing attack. Additionally, the user's password is always secretly concealed with the private key  $K_u$  within the hash function, since it is hard to find  $P$  and  $K_u$  from  $h(K_u || P) \oplus K_u$ . Therefore, no one can reveal the user's password  $P$  without  $U$ 's permission.

### 5.2 Stolen-Verifier Attack

The server stores the verifier of user's password instead of the clear text of the password. In the stolen-verifier attack, the adversary who has stolen the password verifier from the server uses it directly to masquerade as a legitimate user. If the adversary

obtains a copy of the password verifier  $h(K_u \| P) \oplus K_u$  in step R1, he also can obtain  $T_s$  by copying  $h(K_u \| P) \oplus K_u$  and computing  $PV \oplus T_s$  with the previous stolen verifier  $PV$  in step R2. However, the adversary can neither get any information nor manipulate after step R3 without the server's private key. Since it is hard to find  $P$  and  $K_u$  in  $h(K_u \| P) \oplus K_u$ , the adversary cannot derive  $h(K_u \| P \| U_{id}) \oplus K_{pr}$  to obtain the sealed-verifier  $SV$  from  $h(K_u \| P) \oplus K_u$ . Even though the adversary intercepts password verifiers in steps R1 and R2, the adversary cannot use them since there is no way to derive  $U_{id}$  before step R2 for registration. In step R3, since  $U_{id}$  is encrypted with the server's public key, the attacker cannot obtain  $U_{id}$  without the server's private key  $K_{pr}$ . The adversary also cannot obtain  $h(h(K_u \| P \| U_{id}) \oplus PV)$  for login from  $h(K_u \| P) \oplus K_u$ . Even if the adversary steals  $SV = h(K_u \| P \| U_{id}) \oplus K_{pr}$  and  $Auth_Q \oplus Auth_A \oplus K_{pr}$  from the server, he cannot open them without the server's private key. If the adversary obtains the server's private key, he is able to get any information. However, we assumed the server's private key  $K_{pr}$  is kept as a top secret on the server. If  $K_{pr}$  is released, not only does the server's private key need to be changed, but all users should be re-registered too. Since we use the verifier and other unknown values (e.g.  $K_{pr}$  or  $K_u$ ) together, even if the attacker steals the verifier, he will not use it anywhere without knowing  $K_{pr}$  or  $K_u$ . Thus, our scheme can resist any stolen-verifier attacks.

### 5.3 Replay Attack

The replay attack is an offensive action in which the adversary impersonates or deceives another legitimate participant through the reuse of information obtained in protocols. It indicates an attempt by an unauthorized third party to record exchanged messages. In step L3, since  $U_{id}$  is hashed with two other unknown values  $K_u$  and  $P$ , the attacker cannot obtain  $U_{id}$  without the knowledge of  $K_u$  and  $P$ . The adversary is able to steal  $PV$  in step R1 and  $r_s$  in step L2, and then obtain  $h(h(K_u^* \| P^* \| U_{id}) \oplus PV') \oplus h(K_u^* \| P^* \| U_{id})$ . However, he cannot get any information for login, "forget password" and change password protocols. After that, the adversary will try to change  $C_1 = h(h(K_u^* \| P^* \| U_{id}^*) \oplus PV')$  with his own values  $P^*$ ,  $U_{id}^*$  and  $PV'$ . However, the server will detect it as modified (i.e.  $C_1 \neq C_2$ ) in step L4, since the attacker needs the encrypted values with  $K_{pr}$  such as  $K_u$ ,  $P$ , and  $Auth_A$ . The adversary can steal  $PV$  and  $Auth_Q$  in steps R1 and R2, respectively. After that, he will use them for the replay attack. However, this attack cannot be successful, since the adversary needs to know the sealed-verifier  $SV$  and the server's private key  $K_{pr}$  for this attack. Consequently, our proposed scheme can resist a replay attack.

### 5.4 Denial-of-Service Attack

This attack is characterized by the explicit attempt of an attacker to prevent legitimate users of a service from using that service. This attempt includes several different flavors: disrupting service to a specific system or user, preventing a particular user from accessing a service, or denying requests issued by a legitimate user. The adversary, however, is unable to change the user's password without the user's permission in our scheme, since it is hard to find  $P$  and  $K_u$  not only in  $h(K_u \| P) \oplus K_u$ , but also in  $h(h(K_u \| P \| U_{id}) \oplus K_{pr}) \oplus h(K_u \| P \| U_{id}) \oplus K_{pr}$ . There is no chance to change the password or veri-

fier in step R3 or L3. Therefore, our improved scheme can resist a denial-of-service attack.

### 5.5 Impersonation Attack

This attack deceives the identity of one of the legitimate parties. An attacker inserts or changes a message and claims that it originated from a real sender. If the adversary impersonates  $U$  and wants to get the user's former password  $P$ , he should attack the "forget password" protocol. Since he does not know  $Auth_A$ , he cannot obtain the password. If the attacker wants to get the password, he needs to know  $K_{pr}$  for decryption. Moreover, in our protocol,  $Auth_A$  is always protected with a  $S$ 's private key  $K_{pr}$  and other unknown values such as  $R_C$ ,  $R_F$ , and  $h(K_u || P || U_{id})$ . If the adversary logs in the system successfully, he could try to change it with his own password  $P^*$ . However, to change the password, the adversary would need to attack the password/verifier change protocol and know  $K_u$ ,  $Auth_A$ , and  $SV$ , which is impossible. Thus, our proposed scheme can also resist a impersonation attack.

## 6. CONCLUSIONS

In this paper, we have proposed a secure hash-based strong-password authentication protocol using one-time public-key cryptography that includes not only secure registration and login authentication, but secure "forget password" and password/verifier change protocols. It is more secure against guessing, stolen-verifier, replay, denial-of-service, and impersonation attacks.

## REFERENCES

1. C. M. Chen and W. C. Ku, "Stolen-verifier attack on two new strong-password authentication protocols," *IEICE Transactions on Communications*, Vol. E85-B, 2002, pp. 2519-2521.
2. N. M. Haller, "The S/KEY(TM) one-time password system," *Internet Society Symposium on Network and Distributed System Security*, 1994, pp. 151-158.
3. W. C. Ku, "A hash-based strong-password authentication scheme without using smart cards," *ACM Operating System Review*, Vol. 38, 2004, pp. 29-34.
4. W. C. Ku, M. H. Chiang, and S. T. Chang, "Weaknesses of Yoon-Ryu-Yoo's hash-based password authentication scheme," *ACM Operating System Review*, Vol. 39, 2005, pp. 85-89.
5. W. C. Ku, C. M. Chen, and H. L. Lee, "Cryptoanalysis of a variant of Peyravian-Zunic's password authentication scheme," *IEICE Transactions on Communications*, Vol. E86-B, 2003, pp. 1682-1684.
6. M. Kim and Ç. K. Koç, "A simple attack on a recently introduced hash-based strong-password authentication scheme," *International Journal of Network Security*, Vol. 1, 2005, pp. 77-80.
7. M. Kim and Ç. K. Koç, "A simple attack on a recently introduced hash-based secure

- user authentication scheme,” *International Journal of Computer Science and Network Security*, Vol. 6, 2006, pp. 157-160.
8. W. C. Ku, H. C. Tsai, and S. M. Chen, “Two simple attacks on Lin-Shen-Hwang’s strong-password authentication protocol,” *ACM Operating System Review*, Vol. 37, 2003, pp. 26-31.
  9. L. Lamport, “Password authentication with insecure communication,” *Communications of the ACM*, Vol. 24, 1981, pp. 770-772.
  10. C. C. Lee, L. H. Li, and M. S. Hwang, “A remote user authentication scheme using hash functions,” *ACM Operating System Review*, Vol. 36, 2002, pp. 23-29.
  11. C. L. Lin, H. M. Sun, and T. Hwang, “Attacks and solutions on strong-password authentication,” *IEICE Transactions on Communications*, Vol. E84-B, 2001, pp. 2622-2627.
  12. C. W. Lin, J. J. Shen, and M. S. Hwang, “Security enhancement for optimal strong-password authentication protocol,” *ACM Operating System Review*, Vol. 37, 2003, pp. 7-12.
  13. M. Matsui, “New block encryption algorithm *MISTY*,” *Fast Software Encryption*, LNCS 1267, Springer-Verlag, 1997, pp. 54-68.
  14. National Institute for Standards and Technology, “Data encryption standard (DES),” *FIPS 46-3*, Oct 1999.
  15. M. Peyravian and N. Zunic, “Methods for protecting password transmission,” *Computers and Security*, Vol. 19, 2000, pp. 466-469.
  16. M. Sandirigama, A. Shimizu, and M. Noda, “Simple and secure password authentication protocol (SAS),” *IEICE Transactions on Communications*, Vol. E83-B, 2000, pp. 1363-1365.
  17. A. Shimizu, “A dynamic password authentication method by one-way function,” *IEICE Transactions on Information and Systems*, Vol. E73-DI, 1990, pp. 630-636.
  18. A. Shimizu, T. Horioka, and H. Inagaki, “A password authentication method for contents communication on the internet,” *IEICE Transactions on Communications*, Vol. E81-B, 1998, pp. 1666-1673.
  19. A. Shimizu and S. Miyaguchi, “Fast data encipherment algorithm FEAL,” *IEICE Transactions*, Vol. J70-D, 1987, pp. 1413-1423.
  20. T. Tsuji and A. Shimizu, “An implementation attack on one-time password authentication protocol OSPA,” *IEICE Transactions on Communications*, Vol. E86-B, 2003, pp. 2182-2185.
  21. E. J. Yoon, E. K. Ryu, and K. Y. Yoo, “A secure user authentication scheme using hash functions,” *ACM Operating System Review*, Vol. 38, 2004, pp. 62-68.

**Minho Kim** is currently an assistant professor of Computer Science at Korea Air Force Academy and has also been working in Electrical Engineering and Computer Science at Oregon State University. He received his Ph.D. degree from Oregon State University. Dr. Kim’s research interests include algorithms and protocols for cryptography, computer arithmetic, computer and network security, and wireless communications.

**Çetin Kaya Koç** received his Ph.D. (1988) degree in Electrical and Computer Engineering from University of California, Santa Barbara. After working for the University of Houston from 1988-1992 as an Assistant Professor, he moved to Oregon State University, where is a full professor. Dr. Koç has founded Information Security Laboratory at Oregon State University to coordinate faculty and graduate student research efforts concentrated on cryptography, information security, and electronic commerce. In September 2001, he received OSU College of Engineering Research Award for Outstanding and Sustained Research Leadership. his research interests are in cryptographic engineering, algorithms and architectures for cryptography, computer arithmetic and finite fields, parallel algebraic computation, and network security. He has co-founded the Workshop on Cryptographic Hardware and Embedded Systems (CHES) in 1999 and has been the program co-chair and proceedings editor from 1999 to 2003. He is currently a permanent member of the steering committee of CHES. The Proceedings of CHES Workshops are published by Springer in the Lecture Notes in Computer Science (LNCS) series. He was the Guest Co-Editor of the special issue in April 2003 of IEEE Transactions on Computers on cryptographic hardware and embedded software development. He is also in the editorial boards of IEEE Transactions on Computers and IEEE Transactions on Mobile Computing. He is a member of IEEE and IEEE Computer Society. He has become an IEEE Fellow (effective January 1, 2007) for his contributions to cryptographic engineering. He has been working as a consulting engineer with research and development interests in cryptographic engineering and embedded systems for several companies including Intel, RSA Security, Samsung Electronics, and Texas Instruments.