A Security Analysis of Zhao and Gu’s Key Exchange Protocol

Junghyun Nam*, Juryon Paik**, Youngsook Lee***, Dongho Won**

Abstract

Key exchange protocols are essential for building a secure communication channel over an insecure open network. In particular, password-based key exchange protocols are designed to work when user authentication is done via the use of passwords. But, passwords are easy for human beings to remember, but are low entropy and thus are subject to dictionary attacks. Recently, Zhao
and Gu proposed a new server-aided protocol for password-based key exchange. Zhao and Gu’s protocol was claimed to be provably secure in a formal adversarial model which captures the notion of leakage of ephemeral secret keys. In this paper, we mount a replay attack on Zhao and Gu’s protocol and thereby show that unlike the claim of provable security, the protocol is not secure against leakage of ephemeral secret keys. Our result implies that Zhao and Gu’s proof of security for the protocol is invalid.

Keywords : Security, Key exchange protocol, Password, Attack

I. Introduction

Key exchange protocols are designed to allow two or more parties to establish a common secret key over a public network. This secret key, commonly called a session key, is then typically used to build confidential or integrity-protected communication channel between the parties. The highest priority in designing a key exchange protocol is placed on ensuring the security of session keys to be established by the protocol. Roughly speaking, establishing a session key securely means that the key is being known only to the intended parties at the end of the protocol run. But unfortunately, the experience has shown that the design of secure key exchange protocols is notoriously difficult. Thus, key exchange protocols must be subjected to a thorough and systematic scrutiny before they are deployed into a public network, which might be controlled by an adversary.

Secure session-key generation requires an authentication mechanism to be integrated into key exchange protocols. In turn, achieving any form of authentication inevitably requires some secret information to be established between users in advance of the authentication stage. Cryptographic keys, either secret keys for symmetric cryptography or private/public keys for asymmetric cryptography, may be one form of the underlying secret information pre-established between users. However, these high-entropy cryptographic keys are random in appearance and thus are difficult for humans to remember, entailing a significant amount of administrative work and costs. Eventually, it is this drawback that password-based authentication has come to be widely used in reality. Passwords are drawn from a relatively small space like a dictionary, and are easier for humans to remember than cryptographic keys with high entropy.

Bellovin and Merritt [1] was the first to consider how two parties, who only share a weak, low-entropy password, and who are communicating over a public network, authenticate each other and agree on a high-entropy cryptographic key to be used for protecting their subsequent communication. Their protocol, known as encrypted key exchange, or EKE, was a great success in showing how one can exchange password authenticated information while protecting poorly-chosen passwords from the notorious password guessing attacks. Due in large part to the practical significance of password-based authentication, this initial work has been followed by a number of two-party protocols (e.g., [2][3][4][5][6][7]) offering various levels of security and complexity.

While two-party protocols for password-authenticated key exchange (PAKE) are well suited for client-server architectures, they are inconvenient and costly for use in large scale peer-to-peer systems. Since two-party PAKE protocols require each pair of potential communication parties to share a password, a large number of parties result in an even larger number of passwords to be shared. It is due to this problem that three-party models have been often used in
A, B : The clients
S : The authentication server
\[ ID_2 \overset{\text{def}}{=} ID_A, ID_B \]
\[ ID_3 \overset{\text{def}}{=} ID_A, ID_B, ID_S \]

\[ a_1, a_2 \in \mathbb{Z}_q^* \]
\[ A_1 = g^{a_1}, A_2 = g^{a_2} \]
\[ x = H_1(PW_A, a_1, a_2), X = g^x \]
\[ k_A = H_2(R^{a_1}, R^{a_2}, X, PW_A, ID_2) \]
\[ c_A = \text{Enc}_{k_A}(PW_A, \omega_A) \]
\[ M_A = \langle X, A_1, A_2, c_A, \omega_A, ID_2 \rangle \]

\[ b_1, b_2 \in \mathbb{Z}_q^* \]
\[ B_1 = g^{b_1}, B_2 = g^{b_2} \]
\[ y = H_1(PW_B, b_1, b_2), Y = g^y \]
\[ k_B = H_2(R^{b_1}, R^{b_2}, Y, PW_B, ID_2) \]
\[ c_B = \text{Enc}_{k_B}(PW_B, \omega_B) \]
\[ M_B = \langle Y, B_1, B_2, c_B, \omega_B, ID_2 \rangle \]

\[ k_A' = H_2(A_1', A_2', X, PW_A, ID_2) \]
\[ k_B' = H_2(B_1', B_2', Y, PW_B, ID_2) \]
\[ (PW_A', \omega_A') = \text{Dec}_{k_A'}(c_A) \]
\[ (PW_B', \omega_B') = \text{Dec}_{k_B'}(c_B) \]
\[ PW_A' \overset{\text{?}}{=} PW_A, PW_B' \overset{\text{?}}{=} PW_B \]
\[ V_A = H_2(k_B', \omega_{SA}, X, A_1, A_2, ID_3) \]
\[ V_B = H_2(k_A', \omega_{SB}, Y, B_1, B_2, ID_3) \]
\[ M_{SA} = \langle Y, B_1, B_2, V_B, \omega_{SB}, ID_3 \rangle \]
\[ M_{SB} = \langle X, A_1, A_2, V_A, \omega_{SA}, ID_3 \rangle \]

\[ V_B \overset{\text{?}}{=} H_2(k_A, \omega_{SB}, Y, B_1, B_2, ID_3) \]
\[ \text{sid} = \langle X, Y, A_1, A_2, B_1, B_2, ID_3 \rangle \]
\[ Z_1 = (YB_1)^y, Z_2 = (YB_2)^y \]
\[ Z_3 = Y^{x+b_1}, Z_4 = Y^{x+b_2} \]
\[ SK = H(Z_1, Z_2, Z_3, Z_4, \text{sid}) \]

Fig. 1. Zhao and Gu's Three-Party PAKE Protocol
designing PAKE protocols (e.g., [8][9][10][11][12]). In a typical three-party setting, each party (often called client) does not need to remember and manage multiple passwords, but shares only a single password with a trusted server who then assists clients in establishing a session key by providing authentication services to them. However, this convenience comes at the price of clients’ trust in the server. Despite this drawback, the three-party model offers an effective, realistic solution to the problem of session key exchange in large peer-to-peer systems, and in fact is assumed by the popular Kerberos authentication system [13].

Recently, Zhao and Gu [14] proposed a three-party PAKE protocol making use of the trapdoor test technique introduced by Cash, Kiltz, and Shoup [15]. Zhao and Gu’s protocol was claimed to be provably secure under the assumption that the hash functions used in the protocol are random oracles. The adversarial model, where security of the protocol is proven, captures the notion of strong corruption by allowing the adversary to ask EphemeralKeyReveal queries. An EphemeralKeyReveal query against a user instance outputs all the ephemeral secrets used by the instance during the protocol execution. Allowing an adversary to ask EphemeralKeyReveal queries models the adversary’s capability to embed a Trojan horse or other form of malicious code into a user’s machine and then obtain all the session-specific information of the victim. Since Zhao and Gu’s protocol is proven secure in a model that allows EphemeralKeyReveal queries, it should be secure against strong corruption. But what we found is the opposite: Zhao and Gu’s protocol does not exhibit resistance against strong corruption. Indeed, Zhao and Gu’s protocol is vulnerable to a replay attack where the adversary asks an EphemeralKeyReveal query in its attack. We here reveal this security vulnerability of Zhao and Gu’s protocol. Our result invalidates the claimed proof of security for the protocol.

II. Review of Zhao and Gu’s Protocol

This section describes the three-party PAKE protocol proposed by Zhao and Gu [14]. The protocol participants consist of a single server and two clients. The clients wish to establish a session key between them while the server exists to provide the clients with authentication services. We denote by $ID_A$, $ID_B$ and $ID_S$ the identities of $A$, $B$ and $S$, respectively. Let $PW_A$ and $PW_B$ be the passwords of $A$ and $B$, respectively. Each client’s password is assumed to be shared with the authentication server via a secure channel. The followings are the public system parameters used in the protocol.

Two large primes $p$ and $q$ with $q | (p - 1)$, and a generator $g$ of group $G$ of order $q$. A pair of symmetric encryption/decryption algorithms $(Enc, Dec)$ modeled as an ideal cipher [2].

Three hash functions $H_1$, $H_2$ and $H$ modeled as random oracles [16]. $H_1$ and $H_2$ map $\{0,1\}^*$ to $Z_q^*$ while $H$ maps $\{0,1\}^*$ to $\{0,1\}^\lambda$, where $\lambda$ is a security parameter representing the length of session keys.

Once $p$, $q$ and $g$ are fixed, the server $S$ generates its long-term private/public keys $(r, R)$ such that $r \in Z_q^*$ and $R = g^r \mod p$. A high-level depiction of the protocol is given in Fig. 1, and a more detailed description follows:

Client $A$ chooses random $a_1, a_2 \in Z_q^*$ and computes $A_1 = g^{a_1}$ and $A_2 = g^{a_2}$. Then $A$ verifies if $R$ lies in $G$. If not, $A$ aborts. Otherwise, $A$ computes $x = H_1(PW_A, a_1, a_2), \ k_A = H_2(R^{a_1}, R^{a_2}, X, PW_A, ID_A, ID_B), \ c_A = Enc_{k_A}(PW_A, \omega_A)$, where $\omega_A$ is a random value. Finally, $A$ deletes the ephemeral secret $x$ and sends $M_A = \langle X, A_1, A_2, c_A \rangle$. 


if computes value. Finally, and aborts. Otherwise, and sends computes and verifies that is a random value. Similarly, is a random value chosen by deletes the session-specific information: is in lies in , where . If not, aborts. Otherwise, computes and verifies if all of and lie in . If any of these are untrue, aborts. Otherwise, computes and sends , and computes the session key: Similarly, on receiving , checks if (1) and (2) If any of these are untrue, aborts. Otherwise, computes and defines the session ID: and computes the session key: 

III. Adversarial Model

Zhao and Gu’s protocol comes along with a claimed proof of its security in a formal model of adversarial capabilities. The adversarial model that they used is the one of Yoneyama [12] and captures security against strong corruption [2][17][18]. Here we provide an overview of the adversarial model as a preliminary step towards mounting an ephemeral-key reveal attack against the protocol.

1. Participants

Each participant in a three-party key exchange is either a client or the trusted server . Each may run the protocol multiple times either serially or concurrently, with possibly different participants. Thus, at a given time, there could be many instances of a single client and the server. denotes instance of a participant . An instance is said to accept when it computes a valid session key . During the initialization phase of the protocol, the server generates its long-term private/public key pair and each client chooses a password and shares it with .
Passwords are drawn from a dictionary $D$.

2. Adversary

The adversary is in complete control of every aspect of all communications between participants, and may ask, at any time, them to open up access to their long-term secret keys. These capabilities and others of the adversary are modeled via various oracles to which the adversary is allowed to make queries.

Execute($\Pi_C^i, \Pi_C^{i'}, \Pi_S^k$): This query prompts an honest execution of the protocol among the client instances $\Pi_C^i$ and $\Pi_C^{i'}$ and the server instance $\Pi_S^k$. The transcript of the honest execution is returned to the adversary as the output of the query. This oracle call represents passive eavesdropping of a protocol execution.

SendClient($\Pi_C^i, msg$): This query sends message $msg$ to the client instance $\Pi_C^i$. The instance $\Pi_C^i$ proceeds as it would in the protocol upon receiving $msg$. The response message generated by $\Pi_C^i$, if any, is the output of this query and is returned to the adversary. A query of the form SendClient($\Pi_C^i$, start:$C'$) prompts $\Pi_C^i$ to initiate the protocol with a client $C'$ ($\neq C$).

SendServer($\Pi_S^i, msg$): This query sends message $msg$ to the server instance $\Pi_S^i$. The instance $\Pi_S^i$ proceeds as it would in the protocol upon receiving $msg$. The response message generated by $\Pi_S^i$, if any, is the output of this query and is returned to the adversary.

Long-termKeyReveal($U$): This query outputs the long-term secret key of $U$. This oracle call captures the idea that damage due to loss of $U$’s long-term key should be restricted to those sessions where $U$ will participate in the future.

EphemeralKeyReveal($\Pi_U^i$): This query returns all short-term secrets used by instance $\Pi_U^i$. This models the adversary’s capability to embed a Trojan horse or other form of malicious code into a user’s machine and then obtain all the session-specific information of the victim.

SessionKeyReveal($\Pi_C^i$): This query returns the session key $SK_C^i$ held by instance $\Pi_C^i$, modeling leakage of session keys. This oracle call captures the idea that exposure of some session keys should not affect the security of other session keys.

EstablishParty($C, S, PW_C$): This query models the adversary to register a password $PW_C$ on behalf of a client $C$. In this way the adversary totally controls that client. Clients against whom the adversary did not issue this query are called honest.

Test($\Pi_C^i$): This query provides a means of defining security of session keys. The output of this query depends on the hidden bit $b$ chosen uniformly at random from $\{0,1\}$. The Test oracle returns the real session key held by $\Pi_C^i$ if $b = 1$, or returns a random key drawn from the session-key space if $b = 0$. The adversary is allowed to access the Test oracle only once.

TestPassword($C, PW_C$): This query provides a means of defining security of passwords. If the password guess $PW_C'$ is the same as the client $C$’s real password $PW_C$, then return 1. Otherwise, return 0. The adversary can make TestPassword query only once.

3. Partnership

Loosely stated, two instances are partners of each other if they participate together in a protocol execution and share a session key as a result of the execution. The notion of partners is used in the definition of security to disallow the adversary to
ask the Test query against an instance whose partner instance has already been asked for the session key (with a SessionKeyReveal query), ephemeral keys (with an EphemeralKeyReveal query), or long-term keys (with a Long-termKeyReveal query). It is thus important to define partnership correctly. An error in the partnership definition may render a protocol insecure (in the proof model used) when there is no known attack on the protocol (for a concrete example, see the work by Choo et al. [19]).

Let the session identifier ($\text{sid}$) of an instance be a function of the messages sent and received by the instance during its execution. Zhao and Gu follow the recent practice of relying on the notion of $\text{sid}$ to define partnership between instances. According to their definition of partnership, two instances $\Pi^i_C$ and $\Pi^i_{C'}$ (with $C \neq C'$) are said to be partnered if the following conditions hold: (1) both $\Pi^i_C$ and $\Pi^i_{C'}$ have accepted, (2) $\Pi^i_C$ and $\Pi^i_{C'}$ have computed the same $\text{sid}$, (3) the partner identifier for $\Pi^i_C$ is $\Pi^i_{C'}$ and vice versa, and (4) no instance besides $\Pi^i_C$ and $\Pi^i_{C'}$ has accepted with a partner identifier equal to $\Pi^i_C$ and $\Pi^i_{C'}$. When an instance $\Pi^i_C$ accepts, it holds a session key, a session identifier, and a partner identifier.

4. Security Definition

Definition of security is based on the notion of freshness. Intuitively, a fresh instance is an instance which holds a session key about which the adversary should not know. More precisely:

Definition 1 (freshness). Let $\Pi^i_C$ be an instance who has accepted and let $\Pi^i_{C'}$ be $\Pi^i_C$’s partner instance (if it exists). An instance $\Pi^i_C$ is considered fresh if none of the following conditions hold:

- The adversary asks neither SendClient($\Pi^i_C$, $\text{msg}$) nor SendClient($\Pi^i_{C'}$, $\text{msg}'$) query. Then the adversary either makes queries:
  - EphemeralKeyReveal($\Pi^i_C$) or EphemeralKeyReveal($\Pi^i_{C'}$)
  - The adversary asks SendClient($\Pi^i_{C'}$, $\text{msg}'$) query. Then the adversary either makes queries:
    - Long-termKeyReveal($C$),
    - Long-termKeyReveal($S$),
    - EphemeralKeyReveal($\Pi^i_C$) for any session $l$ or EphemeralKeyReveal($\Pi^i_{C'}$)
  - The adversary asks SendClient($\Pi^i_C$, $\text{msg}$) query. Then the adversary either makes queries:
    - Long-termKeyReveal($C'$),
    - Long-termKeyReveal($S$),
    - EphemeralKeyReveal($\Pi^i_C$) or EphemeralKeyReveal($\Pi^i_{C'}$) for any session $l$.

In this definition of freshness, all the queries for $\Pi^i_{C'}$ are defined if $\Pi^i_{C'}$ exists.

The security of a protocol $\pi$ against an adversary $C$ is defined in terms of the probability that $C$ succeeds in distinguishing a real session key established in an execution of $\pi$ from a random session key. That is, the adversary $C$ is considered successful in attacking $\pi$ if it breaks the semantic security of session keys generated by $\pi$. More precisely, the security is defined in the following context. The adversary $C$ executes the protocol exploiting as much parallelism as possible and asking any queries allowed in the adversarial model. During executions of the protocol, the adversary $C$, at any time, asks a Test query to a fresh instance, gets back a key as the response to this query, and at some later point in time, outputs a bit $b'$ as a guess for the value of the hidden bit $b$ used by the Test oracle. Then the advantage of $C$ in attacking protocol
$\pi$ is denoted by $Adv_{\pi}(C)$, and is defined as

$$Adv_{\pi}(C) = |2\Pr[b = b'] - 1|.$$  

Let $Adv_{\pi}(t, Q)$ denote the maximum value of $Adv_{\pi}(C)$ over all $C$ with time complexity at most $t$ and asking at most $Q$ queries. Then, protocol $\pi$ is said to be AKE-secure if $Adv_{\pi}(t, Q)$ is only negligibly larger than $nq_{\text{send}}/|D|$, where $n$ is a constant and $q_{\text{send}}$ is the number of SendClient/SendServer queries. This notion of security is commonly termed as "AKE security".

IV. Breaking AKE Security

In this section, we break the AKE security of Zhao and Gu's key exchange protocol. The security model described in the previous section allows the adversary to ask EphemeralKeyReveal queries. The EphemeralKeyReveal oracle is allowed to check that the protocol is secure against strong corruption. In other words, the EphemeralKeyReveal oracle call captures the idea that exposure of ephemeral secrets of a session should not affect the security of other sessions. Hence, a key exchange protocol proven secure in a model that allows EphemeralKeyReveal queries ought to be secure against an adversary who tries to break the security of a session by exploiting ephemeral secrets obtained from some other sessions. Zhao and Gu's protocol carries a claimed proof of its AKE security, but as we will see below, it does not provide security against strong corruption. This implies that their security proof is flawed.

The vulnerability of Zhao and Gu's protocol against strong corruption is attributed to the fact that clients' messages $\tau_1$ and $\tau_2$ can be replayed without being detected by the server. Our attack starts from this observation. Let $SES$ be an honest protocol session where $A$ and $B$ established a session key as per protocol specification. Suppose now that a malicious adversary $C$ eavesdropped the message $M_B$ sent by $B$ to $S$ in the protocol session $SES$. Suppose also that the adversary $C$ obtained the ephemeral secrets $- y, b_1$ and $b_2$ - which $B$ used in the session $SES$. As also stated in the definition of EphemeralKeyReveal oracle, this leakage of the ephemeral secrets can be justified under the assumption that $C$ has the capability to embed a Trojan horse or other form of malicious code into $B$'s machine and then log
all the session-specific information of $B$. With $y$, $b_1$ and $b_2$ in hand, $C$ can easily impersonate $B$ to $A$ as follows:

$C$ initiates a new session with $A$ as if the initiation message is from $B$.

Next, $C$ sends the message $M_B = \langle Y, B_1, B_2, \omega_B, ID_A, ID_B \rangle$ (eavesdropped in the previous session SES) to $S$ alleging that the message is from $B$.

$C$ then intercepts the message sent by $S$ to $B$ for this new session.

Finally, using $y$, $b_1$ and $b_2$, the adversary $C$ computes the same session key as that of $A$.

This allows $C$ to impersonate $B$ to $A$.

The above attack on Zhao and Gu’s protocol is well captured in the adversarial model. Let again $A$ and $B$ denote two registered clients and $M$ also be any registered client other than $A$ and $B$. Table 1 shows the sequence of oracle queries corresponding to the attack scenario described above. The goal of the adversary $C$ is to break the AKE security of Zhao and Gu’s protocol. $C$ begins by letting $A$ and $B$ execute the protocol together by asking Execute ($\Pi^1_A$, $\Pi^1_B$, $\Pi^1_S$). As a result, $C$ obtains the message $M_B = \langle Y, B_1, B_2, \omega_B, ID_A, ID_B \rangle$ from $B$ to $S$. Then $C$ asks EphemeralKeyReveal ($\Pi^1_B$) to obtain all the ephemeral secrets $\langle b_1, b_2, y, k_B \rangle$ used by instance $\Pi^1_B$. Now $C$ asks SendClient ($\Pi^2_A$, $\Pi^2_B$, $\Pi^2_S$) which prompts instance $\Pi^2_A$ to initiate the protocol with client $B$. In response to this query, $\Pi^2_A$ will output the message $M'_A = \langle X', A'_1, A'_2, \omega'_A, ID_A, ID_B \rangle$. The next queries $C$ makes correspond to an honest execution of the protocol among $\Pi^2_A$, $\Pi^2_B$ (impersonated by $C$) and $\Pi^2_S$. Hence, the rest of the queries are straightforward: $C$ asks SendServer ($\Pi^2_S$, $\langle X', A'_1, A'_2, \omega'_A, ID_A, ID_B \rangle$) and SendServer ($\Pi^2_S$, $\langle Y, B_1, B_2, \omega_B, ID_A, ID_B \rangle$), and then, as $\Pi^2_S$ responds to the queries, asks SendClient ($\Pi^2_A$, $M_B = \langle Y, B_1, B_2, V'_B, \omega'_S, ID_S \rangle$). Notice that $C$ replays the message $M_B$ obtained from $\Pi^1_B$. When $\Pi^2_A$ is sent the query SendClient ($\Pi^2_A$, $\langle Y, B_1, B_2, V'_B, \omega'_S, ID_S \rangle$), it accepts with the session key $SK'_A$ being computed as $SK'_A = H(Z_1, Z_2, Z_3, Z_4, sid)$, where $Z_1 = (Y B_1)^{x'}$, $Z_2 = Y B_1^{x'}$, $Z_3 = Y^{x'} + a_1$, and $Z_4 = Y_{\text{ps}}^{x'}$. It can be easily verified that the instance $\Pi^2_A$ is fresh under Definition 1: (1) no one in $\{A, B, S\}$ has been sent a Corrupt query, (2) no Reveal query has been made against any instance, and (3) the query EphemeralKeyReveal ($\Pi^1_B$) has been asked before the SendClient queries to $\Pi^2_A$ have been asked. Thus, $C$ may test (i.e., ask the Test query against) the instance $\Pi^2_A$. $C$ is able to compute $SK'_A$ on its own since it knows the values of the exponents $b_1$, $b_2$, $y$ used to compute $B_1$, $B_2$ and $Y$. This means that $Pr[b = b'] = 1$ and hence $Adv_a(C) = 1$. Therefore, $C$ achieves its goal of breaking the AKE security of Zhao and Gu’s protocol.

Generally speaking, it is desirable that ephemeral secrets exposed in a session should not jeopardize the session-key secrecy of any other sessions. For this reason, key exchange protocols proven AKE-secure in a model that allows strong corruption ought to be resistant against any attacks similar to ours. Our attack shows that the proof of security for Zhao and Gu’s protocol is invalid. The problem with the proof is that the result of EphemeralKeyReveal queries was not adequately considered in the simulation.
V. Conclusion

This work has considered the security of Zhao and Gu’s three-party protocol [14] for password authenticated key exchange. Although Zhao and Gu’s protocol comes along with a claimed proof of its security, we have shown that the protocol is not secure against strong corruption in the context of the proof model. This vulnerability, however, is not just a failure of Zhao and Gu’s protocol but it is an inherent limitation of all three-party protocols that do not require the server to verify the freshness of incoming messages. Thus, there is no quick tweak we can apply to make Zhao and Gu’s protocol resistant to strong corruption. Moreover, it is not clear how to make the protocol achieve any form of provable security.

참고문헌


저 자 소개

남 정 현
1997 : 성균관대학교 정보공학과 공학사
2002 : University of Louisiana, Lafayette, Computer Science, MS.
2006 : 성균관대학교 컴퓨터공학과 공학박사
현 재 : 건국대학교 컴퓨터공학부 부교수
 관심분야 : 코딩테크놀로지, 암호학
 Email : jhnam@kku.ac.kr

백 주 련
1997 : 성균관대학교 정보공학과 공학사
2005 : 성균관대학교 컴퓨터공학과 공학석사
2008 : 성균관대학교 컴퓨터공학과 공학박사
현 재 : 성균관대학교 컴퓨터공학부 연구교수
 관심분야 : 디지털미디어, 지능정보검색, 유사성 검색
 Email : wise96@ece.skku.ac.kr

이 영 숙
1987 : 성균관대학교 정보공학과 공학사
2005 : 성균관대학교 정보보호학과 공학석사
2008 : 성균관대학교 컴퓨터공학과 공학박사
2010~2011.6 : 호원대학교 기획조정처 경영평가실장
 현 재 : 호원대학교 사이버수사경찰학부 학부장
 관심분야 : 악호프로토콜, 네트워크 보안, SMART 연구, 디지털포렌식
 Email : ysooklee@howon.ac.kr

원 동 호
1976~1988 : 성균관대학교 전자공학과 학사, 석사, 박사
1978~1980 : 한국전자통신연구원 전임연구원
1985~1986 : 일본 동京공업대 객원연구원
1988~2003 : 성균관대학교 교학처장, 전기전자 및 컴퓨터공학부장, 정보통신대학원장, 정보통신기술연구소장, 연구처장
1996~1998 : 국무총리실 정보화추진위원회 자문위원
2002~2003 : 한국정보보호학회장
2002~2008 : 대검찰청 컴퓨터범죄수사단자문위원, 감사원 IT감사자문위원
 현 재 : 성균관대학교 컴퓨터공학과 교수, BK21 사업단장, 한국정보보호학회 명예회장
 관심분야 : 악호학, 정보이론, 정보보호
 Email : dhwon@security.re.kr