情報保護學會論文誌
第15券 第1號, 2005. 2

Bresson-Chevassut-Essiari-Pointcheval의 저전력 모바일 장치를 위한 키 동의 방식의 안전성 분석*

남 정 현,†‡ 이 영 교, 김 승 주, 원 동 호
성균관대학교 정보통신공학부

Cryptanalysis of Bresson-Chevassut-Essiari-Pointcheval's Key Agreement Scheme for Low-Power Mobile Devices

Junghyun Nam,†‡ Younggyo Lee, Seungjoo Kim, Dongho Won
School of Information and Communication Engineering,
Sungkyunkwan University

요 약

Bresson 등은 최근에 발표한 논문에서 무선 네트워크 환경에 적합한 그룹 키 동의 방식을 제안하였고 이의 안전성을 증명하였다. 하지만 본 논문에서는 Bresson 등이 제안한 그룹 키 동의 방식이 여러가지 공격에 취약함을 보임으로써 안전성 증명에 오류가 있음을 입증한다.

ABSTRACT

Bresson et al. have recently proposed an efficient group key agreement scheme well suited for a wireless network environment. Although it is claimed that the proposed scheme is provably secure under certain intractability assumptions, we show in this paper that this claim is unfounded, breaking the allegedly secure scheme in various ways.

Keywords : *Group key agreement, key authentication, forward secrecy, known key security, collusion attack, interleaving attack.*

Ⅰ. Introduction

In many ways, security risks for mobile computing are similar to those for other computing platforms. There are the usual concerns of protecting privileged information, authenticating users and devices. However, the devices' strict limitations on hardware and software characteristics add to the challenges. Due to serious resource constraints on mobile devices, security solutions targeted for more traditional networks are often not directly applicable to wireless networks. In this sense it has been of particular interest to devise an efficient protocol for secure wireless communications.

Though secure key distribution is of vital concern to anyone interested in communicating securely over a public net-

접수일 : 2004년 11월 3일 ; 채택일 : 2005년 2월 2일
* 본 연구는 정보통신부 및 정보통신진흥원의 대학 IT연구센터 육성·지원사업의 연구결과로 수행되었음.
† 주저자, ‡ 교신저자 : jhnam@dosan.skku.ac.kr

work, the design of key exchange protocols that are secure against an active adversary is not an easy task to do, especially in a multi-party setting: there is a long history of protocols for this domain being proposed and subsequently broken by some active attacks (e.g., [1-2]). Consequently, key exchange protocols must be subjected to the strictest scrutiny possible before they can be deployed into an adversarially controlled network.

The recent work of Bresson et al.[3] proposed a very efficient group key agreement scheme well suited for unbalanced networks consisting of devices with strict power consumption restrictions and wireless gateways with less stringent restrictions. The proposed scheme consists of three protocols: the setup protocol GKE.Setup, the remove protocol GKE.Remove, and the join protocol GKE.Join. The main GKE.Setup protocol allows a set of mobile devices (also called *clients*) and a wireless gateway (also called *server*) to agree on a common secret key called a session key. To meet the efficiency needs of clients, the protocol shifts most of computational burden to the gateway and provides mobile devices with the ability to perform public-key cryptographic operations off-line. The other protocols of the scheme allow the server to efficiently handle dynamic membership changes of clients in one wireless domain.

In this paper we show that the Bresson et al.'s group key agreement scheme is completely insecure, presenting various attacks against the three protocols of the scheme.

II. Review of Bresson et al.'s Scheme

Let $\mathbb{G}$ be a finite cyclic group of l-bit prime order q, where l is a security parameter, and let g be an arbitrary generator of $\mathbb{G}$. Both $\mathbb{G}$ and g are known to all parties in the network. There are three hash functions $H:\{0,1\}^* \rightarrow \{0,1\}^l$, $H_0:\{0,1\}^* \rightarrow \{0,1\}^{l_0}$, where l_0 needs not be equal to l, and $H_1:\{0,1\}^{l_1} \times \mathbb{G} \rightarrow \{0,1\}^{l_0}$, where l_1 is the maximal bit-length of a counter c used in the scheme.

2.1 Long-Term Key Generation

We denote by S the server and by C the set of all clients that can participate in the protocols. Before the setup protocol is run for the first time, an initialization phase occurs during which:

1. The server S sets its private/public keys to be $(SK_S, PK_S) = (x, y)$, where $x \in_R \mathbb{Z}_q^*$ and $y = g^x$.
2. Each client $U_i \in C$ generates a pair (SK_i, PK_i) of signing/verifying keys by running the key generation algorithm of a signature scheme.

2.2 The GKE.Setup Protocol

Let $G_c \subseteq C$ be a set of *clients* who wish to establish a session key with the server S. In the protocol, each client $U_i \in G_c$ generates an ephemeral Diffie-Hellman pair $(x_i, y_i = g^{x_i})$, which then leads to a secret value $\alpha_i = g^{xx_i}$ shared between the client U_i and the server S. Signatures are used for authenticating the clients. Let I_c be the set of indices of the clients in G_c. The actual protocol executes in two rounds as follows:

Round 1. Each client $U_i \in G_c$ chooses a random $x_i \in \mathbb{Z}_q$, and precomputes $y_i = g^{x_i}$,

$\alpha_i = y^r$ and a signature σ_i of y_i under the signing key SK_i. Each client U_i then sends (y_i, σ_i) to the server S.

Round 2. For each message (y_i, σ_i), the server S verifies the signature σ_i using PK_i and if the verification succeeds, computes the value $\alpha_i = y_i^r$. Then S initializes the counter c to 0, computes the shared secret value

$$K = H_0(c \| \{\alpha_i\}_{i \in I_c}),$$

and sends to each client U_i the values c and $K_i = K \oplus H_1(c \| \alpha_i)$.

Key computation. After recovering the shared secret value K as

$$K = K_i \oplus H_1(c \| \alpha_i),$$

each client U_i (and S) computes their session key as:

$$sk = H(K \| G_c \| S).$$

2.3 The GKE.Remove Protocol

Let R be a set of clients leaving an existing client group G_c. Then, the GKE.Remove protocol is run to provide the sever S and the remaining clients with a new session key sk. After the client group G_c is updated to be $G_c \setminus R$, the protocol proceeds as follows:

Round 1. The server S increases the counter c and computes the common secret value

$$K = H_0(c \| \{\alpha_i\}_{i \in I_c}).$$

S then sends to each client $U_i \in G_c$ the values c and $K_i = K \oplus H_1(c \| \alpha_i)$.

Key computation. After receiving the values c and K_i, each client $U_i \in G_c$ first checks that the new counter is greater than the old one, and simply recovers the common secret value K and the session key sk as follows:

$$K = K_i \oplus H_1(c \| \alpha_i) \text{ and } sk = H(K \| G_c \| S).$$

2.4 The GKE.Join Protocol

Let J be a set of new clients who want to join an existing client group G_c. Then, the client group G_c is updated to be $G_c \cup J$ and the GKE.Join protocol is run to provide S and each client $U_i \in G_c$ with a new session key sk. The protocol proceeds as follows:

Round 1. Each new client $U_j \in J$ chooses a random $x_j \in \mathbb{Z}_q$, and precomputes $y_j = g^{x_j}$, $\alpha_j = y^{x_j}$ and a signature σ_j of y_j under the signing key SK_j. Each client $U_j \in J$ then sends (y_j, σ_j) to the server S.

Round 2. The server S verifies the incoming signatures, and if correct, operates as in the setup protocol, with an increased counter c: it computes the shared secret value

$$K = H_0(c \| \{\alpha_i\}_{i \in I_c}),$$

and sends to each client $U_i \in G_c$ the values c and $K_i = K \oplus H_1(c \| \alpha_i)$.

Key computation. Each client $U_i \in G_c$ already holds the value $\alpha_i = y^{r_i}$ and the old counter value (set to zero for the new ones). So it first checks that the new counter is greater than the old one, and simply recovers the shared secret value K and the session key sk as follows:

$K=K_i \oplus H_1(c \| \alpha_i)$ and $sk=H(K \| G_c \| S)$.

Ⅲ. Attacks on the GKE.Setup Protocol

In this section we show that the GKE.Setup protocol does not meet the main security properties: implicit key authentication, forward secrecy, and known key security.

3.1 Implicit Key Authentication

The fundamental security property for a key exchange protocol KEP to achieve is implicit key authentication, which is defined in the following context[4.1]. Let sk_i be the session key computed and accepted by U_i as a result of protocol KEP, and let U be the set of the intended participants of KEP. Then we say that KEP provides implicit key authentication if each $U_i \in U$ is assured that no party $U_k \notin U$ can learn the key sk_i unless helped by a dishonest $U_j \in U$.

To show that the GKE.Setup protocol does not provide implicit key authentication, we consider two runs of the protocol which are executed in an either concurrent or non-concurrent manner. We denote by G_c and G'_c the sets of clients with respect to the first and second runs, respectively. Assume that the adversary A participates as a client in the first run of the protocol (i.e., $A \in G_c$), but is intended to be excluded from the second run (i.e., $A \notin G'_c \cup \{S\}$). Also assume that two client sets G_c and G'_c are non-disjoint. The goal of adversary A is to share the same key with the participants of the second run. To do so, the adversary A gathers some information during the first run and uses that information to impersonate some client in the second run. The detailed attack scenario is as follows:

1. In the first run of the protocol, the adversary A computes the shared secret value K participating as a normal client. A then obtains $H_1(c \| \alpha_i)$ for all $i \in I_c$ by computing

$$H_1(c \| \alpha_i) = K \oplus K_i,$$

which can be done without knowing α_i. The adversary A records $H_1(c \| \alpha_i)$ and (y_i, σ_i) for all $i \in I_c$.

2. In the first round of the second run, the adversary A (pretending to be U_j for some $U_j \in G_c \cap G'_c$) replaces the message (y'_j, σ'_j) sent by U_j with (y_j, σ_j) stored in the previous step of this scenario. Because the server S thinks that (y_j, σ_j) is from U_j, it will compute the shared secret value K' as per protocol specification and will send to client U_j the values $c=0$ and

$$K'_j = K' \oplus H_1(c \| \alpha'_j),$$

with α'_j computed as $\alpha'_j = y_j^x$.

3. In the second round of the second run, the adversary A reads K'_j which is transmitted through an open channel. Now, from the values K'_j and $H_1(c \| \alpha_j)$ (obtained in the first step of this scenario), the adversary A can recover the shared secret value K' as follows:

$$K' = K'_j \oplus H_1(c \| \alpha_j).$$

This equation holds, since $\alpha'_j = \alpha_j$ and

thus $H_1(c \| \alpha'_j) = H_1(c \| \alpha_j)$. Finally, the adversary A can share the same session key $sk' = H(K' \| G'_c \| S)$ with all the participants of the second run except U_j.

Consequently, there seems to be little reason to expect that the GKE.Setup protocol provides implicit key authentication, as soon as the adversary participates as a client in a protocol execution and is intended to be excluded from another protocol execution with a non-disjoint set of clients.

3.2 Forward Secrecy

The perfect forward secrecy property says that earlier session keys are protected against loss of some underlying information at the present time. As noted by the authors themselves, the GKE.Setup protocol does not provide perfect forward secrecy: as soon as the long-term private key x of the server is leaked, all the past session keys can be recovered since every α_i can easily be computed from y_i and x.

However, it is claimed by the authors that the protocol achieves partial forward secrecy: disclosure of the private signing keys of clients does not reveal anything about previous session keys. In support of this claim, they argue that the long-term keys of the clients are used for implicit authentication only, and not for hiding the session key. But, this claim is flawed. The attack below shows that if some client's signing key is ever revealed, then any previous session key can be computed by an active adversary. As a simple scenario, we consider two runs of the protocol with the first one being completed before the second one begins. Similarly as before, we denote the client groups with respect to the first and second runs of the protocol by G_c and G'_c respectively. Assume that the adversary A wants to recover the session key established in the first run of the protocol in which she has not participated. The attack is launched as follows:

1. In the first run of the protocol, the adversary A eavesdrops on the session recording the transmitted messages (y_i, σ_i) and

$$K_i = K \oplus H_1(c \| \alpha_i)$$

for some $i \in I_c$.

2. Now, the adversary A participates as a client in the second run of the protocol. Because we consider forward secrecy, we will assume that the private signing key SK_j of some other client $U_j \in G'_c$ is exposed to A.
3. In the first round of the second run, the adversary A proceeds much like a normal client by sending the message (y'_A, σ'_A) to the server. In the same time period, the adversary A (pretending to be the client U_j) replaces the message (y'_j, σ'_j) sent by U_j with (y_i, σ''_j), where σ''_j is the signature of y_i (stored in the first step) under the private key SK_j. Note that the adversary A can sign any message of its choice on behalf of U_j.
4. Because the verification of signature σ''_j will succeed, the server S will compute α'_j as

$$\alpha'_j = y_i^x$$

and the shared secret value K' as

per protocol specification. Then S will send to client U_j the values $c=0$ and

$$K'_j = K' \oplus H_1(c \| \alpha'_j).$$

5. Now, in the second round of the second run, the adversary A should be able to compute the shared secret value K' since it participates as a group member. From K' and K'_j, the adversary A can recover $H_1(c \| \alpha'_j)$ as follows:

$$H_1(c \| \alpha'_j) = K' \oplus K'_j .$$

6. With this information $H_1(c \| \alpha'_j)$, the adversary A can recover the shared secret value K of the first run of the protocol as follows:

$$K = K_i \oplus H_1(c \| \alpha'_j),$$

since $\alpha_i = \alpha'_j$ and thus $H_1(c \| \alpha_i) = H_1(c \| \alpha'_j)$. Finally, A can compute the session key $sk = H(K \| G_c \| S)$ of the first run.

Therefore, once an underlying key is exposed, there is nothing to prevent an adversary with the key from accessing privileged information communicated in earlier sessions.

3.3 Known Key Security

A protocol is said to provide known key security if compromising some session keys does not allow a passive adversary to compromise keys of other sessions, nor an active adversary to impersonate one of the protocol parties. In this subsection, we will extend our security analysis of the protocol by presenting an active known key attack. We will assume two sessions of the protocol with the same participants. Then the following attack is possible:

1. In the first run of the protocol, the adversary A eavesdrops on the session recording the transmitted messages (y_i, σ_i) and (c, K_i) for all $i \in I_c$. Since we consider known key attack, we will assume that the session key K of this run is revealed to A.
2. In the first round of the second run, the adversary A replaces the message (y'_i, σ'_i) sent by each U_i with (y_i, σ_i) obtained in the previous step.
3. Since all the y_i's are replayed, the server S will compute the same session key as computed in the first run of the protocol.

Therefore, at the end of this scenario, the server S will share with the adversary A the key that has been shared by all participants of the first session of the protocol.

Ⅳ. Collusion Attack on the GKE.Remove Protocol

Let A_1 and A_2 denote two colluding adversaries. First, consider two existing sessions of the GKE.Setup protocol with the client groups G_c $(A_1, A_2 \notin G_c)$ and G'_c $(A_1, A_2 \in G'_c)$, respectively. Now, assume that the adversaries A_1 and A_2 move from G'_c to G_c, and thus two new sessions of the protocols are opened: in the first session, the GKE.Join protocol is executed with the client group $G_c = G_c \cup \{A_1, A_2\}$; while in the second session, the

GKE.Remove protocol is executed with the client group $G'_c = G'_c \setminus \{A_1, A_2\}$. We also assume that $G_c \cap G'_c \neq \emptyset$. Then, the following attack on the second session, where the GKE.Remove protocol is executed, is possible:

1. Before moving from G'_c to G_c, in the second run of the setup protocol, one of two adversaries, say A_2, records the message (y'_k, σ'_k) sent to S by some client $U_k \in G_c \cap G'_c$.
2. Now, after moving from G'_c to G_c, the adversaries A_1 and A_2 participate in the run of the join protocol. In the first round of this run, the adversary A_1 sends to S the message (y_{A_1}, σ_{A_1}) exactly following the protocol specification. In the same time period, the adversary A_2 generates the signature σ_{A_2} of y'_k (obtained in the previous step of this scenario) under her signing key SK_{A_2} and sends the message (y'_k, σ_{A_2}) to S. In the second round, S operates as specified in the protocol, verifying the received signatures, increasing the counter c, computing α_{A_1}, $\alpha_{A_2} = y'^{r}_{k}$ and K, and sending out the keying material to the clients in G_c.
3. Now that the two colluding adversaries A_1 and A_2 receive from S the messages (c, K_{A_1}) and (c, K_{A_2}) respectively, they can easily obtain the value $H_1(c \| \alpha_{A_2})$ without knowing α_{A_2}. The adversaries first compute K as

$$K = K_{A_1} \oplus H_1(c \| \alpha_{A_1})$$

using α_{A_1}, and then, from K, recover $H_1(c \| \alpha_{A_2})$ as

$$H_1(c \| \alpha_{A_2}) = K \oplus K_{A_2}.$$

4. Now, in the run of the GKE.Remove protocol with the client group G'_c ($U_k \in G'_c$), the adversaries eavesdrop on the message (c', K'_k) sent by S to the client U_k. Meanwhile, after receiving the messages from S, the clients in G'_c operates as specified in the protocol, checking that the new counter is greater than the old one, recovering the common secret value K', computing the session key sk' as:

$$sk' = H(K' \| G'_c \| S)$$

But, this session key sk' can be also obtained by the adversaries. Since $c = c'$ and $\alpha_{A_2} = \alpha'_k$, and thus $H_1(c \| \alpha_{A_2}) = H_1(c' \| \alpha'_k)$, the adversaries can compute K' as

$$K' = K'_k \oplus H_1(c \| \alpha_{A_2})$$

using $H_1(c \| \alpha_{A_2})$ obtained in the run of the join protocol, and hence the session key sk'.

Consequently, all the users taking part in the remove protocol share a session key with A_1 and A_2.

V. Interleaving Attack on the GKE.Join Protocol

We now show that the GKE.Join protocol does not satisfy implicit key authentication. Let's assume that a set of new clients, J, wants to join two existing

sessions of the GKE.Setup protocol with the client groups G_c ($A \in G_c$) and G'_c ($A \notin G'_c$), respectively. Assume further that the clients in J are permitted to join, and thus two concurrent runs of the GKE.Join protocol are opened with the new client groups $G_c = G_c \cup J$ and $G'_c = G'_c \cup J$, respectively. Then, an interleaving attack can be launched against the clients in $J \subset G'_c$. As can be observed in the following attack, increasing the counter (in the second round of the GKE.Join protocol) does not play any role to prevent relaying messages between two sessions from leading to a successful attack.

1. In the first round of the second run, the adversary intercepts all the messages (y'_j, σ'_j) sent to S by the clients in $J \subset G'_c$.
2. In the first round of the first run, the adversary A replaces the message (y_j, σ_j) sent to S by each client U_j in $J \subset G_c$ with (y'_j, σ'_j) obtained in the previous step of this scenario.
3. In the second round of the first run, the server S verifies all the received signatures. All these signature verifications should be passed since they have been honestly generated by the clients themselves. After the verifications are complete, S operates as specified in the protocol, computing α_j as $\alpha_j = y_j^x$ for each new client U_j in $J \subset G_c$ and increasing the counter c. The server S then sends to each client U_i in G_c the values c and

$$K_i = K \oplus H_1(c \| \alpha_i),$$

and to the adversary A the values c and

$$K_A = K \oplus H_1(c \| \alpha_A).$$

Now, the adversary A records all the messages sent by S to the other clients while recovering the shared secret value K from K_A.

4. In the second round of the second run, the adversary A (pretending to be the server S) sends to each client U_j in $J \subset G'_c$ the message (c, K_j) obtained in the second round of the first run. After receiving this message from A, each client in $J \subset G'_c$ first checks that the newly received counter is greater than the old one; this verification will succeed since the server S increased the counter for the first run of the protocol. Then, each client in $J \subset G'_c$ recovers K from K_j and compute the session key as:

$$sk' = H(K \| G'_c \| S).$$

At the end of this attack, the clients in $J \subset G'_c$ believe that they have established a secure session with S sharing a secret key sk', while in fact they have shared it with A.

Ⅵ. Improvement

To frustrate the attacks, we must somehow prevent information obtained in some sessions from being used for a successful attack on any other session. At first glance, it seems like that the weakness of the scheme is solely because the counter c is always initialized to the same value 0 in the setup protocol. However, we note that increasing the counter monotonically with each new session of the protocol is

not enough by itself to solve the security problems. Indeed, with this modification alone, the protocol can still be broken by an interleaving attack in a scenario where two sessions of the protocol are executed concurrently: note that the interleaving attack described in the previous section does not require that two counter values used in two concurrent sessions be the same. But fortunately, it turned out that we can easily solve the security problems without compromising the efficiency of the original scheme, by modifying the counter management scheme and the computation of K_i to the following:

1. During the initialization phase, the server S and all clients in C initialize their own counter c to 0.
2. The server S increases the counter c monotonically with each new session of the protocols, and computes each K_i as

$$K_i = K \oplus H_0(c \| \alpha_i \| G_r \| S).$$

3. Upon receiving the message (c, K_i) from the server S, each client $U_i \in G_r$ checks that the newly received counter is greater than the old one, recovers the shared secret value K as

$$K = K_i \oplus H_0(c \| \alpha_i \| G_r \| S),$$

and updates its counter to the new value.

Our improved version is as efficient as the original scheme, meeting the efficiency needs of clients: it provides low-power mobile devices with the ability to perform all of the public-key cryptographic operations off-line.

Ⅶ. Conclusion

The recent work of Bresson et al.[3] proposed an efficient group key agreement scheme which consists of three protocols, the setup protocol GKE.Setup, the remove protocol GKE.Remove, and the join protocol GKE.Join. But unfortunately, this group key agreement scheme is completely insecure since:

1. The GKE.Setup protocol does not meet the main security properties, namely, implicit key authentication, forward secrecy, and known key security.
2. The GKE.Remove protocol can be broken by a collusion attack in a scenario where it is executed concurrently with the GKE.Join protocol.
3. The GKE.Join protocol is also vulnerable to an interleaving attack.

References

[1] O. Pereira and J.-J. Quisquater, "A security analysis of the Cliques protocols suites," In Proc. of 14th IEEE Computer Security Foundations Workshop, pp. 73-81, June 2001.

[2] C. Boyd and J.M.G. Nieto, "Round-optimal contributory conference key agreement," In Proc. of PKC 2003, LNCS 2567, pp. 161-174, January 2003.

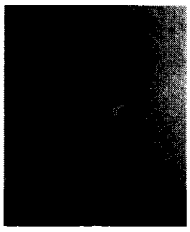
[3] E. Bresson, O. Chevassut, A. Essiari, and D. Pointcheval, "Mutual authentication and group key agreement for low-power mobile devices," Computer Communications, vol.27, no.17, pp. 1730-1737, November 2004. A preliminary version appeared in Proc. of the 5th IFIP-TC6/IEEE International Conference on Mobile and Wireless Communications Networks (MWCN 2003).

[4] G. Ateniese, M. Steiner, and G. Tsu-

dik, "New multiparty authentication services and key agreement protocols," IEEE Journal on Selected Areas in Communications, vol.18, no.4, pp. 628-639, April 2000.

〈著者紹介〉

남 정 현 (Junghyun Nam) 학생회원
1997년 2월 : 성균관대학교 정보공학과(공학사)
2002년 5월 : Computer Science, University of Louisiana, Lafayette(M.S.)
2003년 3월~현재 : 성균관대학교 정보통신공학부 박사과정
〈관심분야〉 암호 프로토콜, 암호이론, 네트워크 보안



이 영 교 (Younggyo Lee) 학생회원
1986년 2월 : 한양대학교 전자공학과(공학사)
1991년 6월 : 한양대학교 전자공학과(공학석사)
2002년 3월~현재 : 성균관대학교 정보통신공학부 박사과정
1993년 3월~1998년 9월 : 대우통신 종합연구소 선임연구원
1999년 2월~2001년 6월 : LG 전자/정보통신 중앙연구소 선임연구원
2002년 3월~현재 : 인하공업대학 정보통신과 초빙전임강사
〈관심분야〉 암호 프로토콜, 정보통신 보안, 네트워크 이론

김 승 주 (Seungjoo Kim) 종신회원
1994년 2월 : 성균관대학교 정보공학과(공학사)
1996년 2월 : 성균관대학교 대학원 정보공학과(공학석사)
1999년 2월 : 성균관대학교 대학원 정보공학과(공학박사)
1998년 12월~2004년 2월: 한국정보보호진흥원(KISA) 팀장
2001년 1월~현재 : 한국정보보호학회 논문지편집위원
2002년 4월~현재 : 한국정보통신기술협회(TTA) IT 국제표준화 전문가
2004년 3월~현재 : 성균관대학교 정보통신공학부 교수
〈관심분야〉 암호이론, 정보보호표준, 정보보호제품 및 스마트카드 보안성 평가, PET

원 동 호 (Dongho Won) 종신회원
1976년~1988년 : 성균관대학교 전자공학과(학사, 석사, 박사)
1978년~1980년 : 한국전자통신연구원 전임연구원
1985년~1986년 : 일본 동경공업대 객원연구원
1988년~2003년 : 성균관대학교 교학처장, 전기전자 및 컴퓨터공학부장, 정보통신대학원장, 정보통신기술연구소장, 연구처장.
1996년~1998년 : 국무총리실 정보화추진위원회 자문위원
2002년~2003년 : 한국정보보호학회장
현재 : 성균관대학교 정보통신공학부 교수, 한국정보보호학회 명예회장, 정통부지정 정보보호인증기술연구센터 센터장
〈관심분야〉 암호이론, 정보이론, 정보보호