Security Weaknesses in Harn-Lin and Dutta-Barua Protocols for Group Key Establishment

Junghyun Nam[1], Moonseong Kim[2], Juryon Paik[3] and Dongho Won[3]
[1] Department of Computer Engineering, Konkuk University, Korea
[e-mail: jhnam@kku.ac.kr]
[2]Information and Communications Examination Bureau, Korean Intellectual Property Office, Korea
[e-mail: moonseong@kipo.go.kr]
[3] Department of Computer Engineering, Sungkyunkwan University, Korea
[e-mail: wise96@ece.skku.ac.kr, dhwon@security.re.kr]
*Corresponding author: Dongho Won

Received August 14, 2011; revised October 7, 2011; revised November 18, 2011; revised January 11, 2012; accepted January 19, 2012; published February 28, 2012

Abstract

Key establishment protocols are fundamental for establishing secure communication channels over public insecure networks. Security must be given the topmost priority in the design of a key establishment protocol. In this work, we provide a security analysis on two recent key establishment protocols: Harn and Lin's group key transfer protocol and Dutta and Barua's group key agreement protocol. Our analysis shows that both the Harn-Lin protocol and the Dutta-Barua protocol have a flaw in their design and can be easily attacked. The attack we mount on the Harn-Lin protocol is a replay attack whereby a malicious user can obtain the long-term secrets of any other users. The Dutta-Barua protocol is vulnerable to an unknown key-share attack. For each of the two protocols, we present how to eliminate their security vulnerabilities. We also improve Dutta and Barua's proof of security to make it valid against unknown key share attacks.

Keywords: Security, group key establishment, attack, secret sharing

This work was supported by Priority Research Centers Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2011-0018397).

DOI: 10.3837/tiis.2012.02.018

1. Introduction

Key establishment protocols allow two or more communicating parties to establish their common secret key called a *session key*. Establishment of session keys is one of the fundamental cryptographic operations and provides a typical way of building secure communication channels over insecure public networks. Traditionally, protocols which can be run by an arbitrary number of parties are called group (or conference) key establishment protocols [1][2][3][4][5], in contrast to protocols which can be run only by two or three parties. In the group setting, a session key is also called a *group key*. Key establishment protocols are often classified into two types: key agreement protocols and key transfer protocols. Key agreement protocols require each participant to contribute their parts to the final form of the session key, whereas key transfer protocols allow one trusted entity to generate the session key and then transfer it to all participants.

The first priority in designing a key establishment protocol is placed on ensuring the security of the protocol. Even if it is computationally infeasible to break the cryptographic algorithms used, the whole system becomes vulnerable to all manner of attacks if the keys are not securely established. But the experience shows that the design of secure key establishment protocols is notoriously difficult. Over the last decades, a number of protocols have been found to be insecure years after they were published [6][7][8][9][10]. Thus, key establishment protocols must be subjected to a thorough scrutiny before they can be deployed into a public network which might be controlled by an adversary.

The fundamental security attribute that a key establishment protocol is expected to achieve is *implicit key authentication*. Informally, this attribute means that no one other than the intended parties can compute the session key. A key establishment protocol achieving implicit key authentication is said to be *authenticated*, and is a primitive of crucial importance in much of modern cryptography and network security. Authenticated key establishment inevitably requires some secret information to be established between the communicating parties before the protocol is ever executed. The pre-established secrets are commonly known as *long-term keys*. Implicit key authentication can be achieved only when the secrecy of every long-term key is guaranteed. As soon as the long-term key of a party is disclosed, all the protocol sessions that the party participates become completely insecure. It is thus crucial that long-term keys must not be revealed under any circumstances.

Resistance to *unknown key-share (UKS) attacks* is among many desirable security attributes that key establishment protocols should achieve. An adversary U_A is said to succeed in an UKS attack if the attack results in two parties U_i and U_j such that: (1) U_i and U_j have computed the same session key; (2) U_i is unaware of the "key share" with U_j and falsely believes its key is shared with U_A; and (3) U_j correctly believes its key is shared with U_i. As implied by this definition, the adversary U_A need not obtain any session key to benefit from an UKS attack. The adversary U_A may be able to take advantage of U_i's false belief in various ways if subsequent messages are encrypted or authenticated with the established key [11]. UKS attacks were first discussed by Diffie et al. [12] and have been found against many key establishment protocols [7][8][13][14][11][15].

In this work, we are concerned with the security of two recent key establishment protocols, namely the group key transfer protocol due to Harn and Lin [16] and the group key agreement protocol due to Dutta and Barua [17]. The Harn-Lin protocol employs Shamir's secret sharing

[18] and assumes a trusted key generation center (KGC) who provides key distribution service to its registered users. During registration, KGC issues each user a long-term key which should be kept privately by the user. One of the security claims made for this protocol is that the long-term key of each user cannot be learned by other users. But, it turns out that this claim is not true. The truth is that the Harn-Lin protocol is vulnerable to a replay attack whereby a malicious user, who is registered with KGC, can readily obtain the long-term key of any other registered user. We reveal this security vulnerability of the Harn-Lin protocol and then suggest a countermeasure against the replay attack. The attack we mount on the Dutta-Barua protocol is an UKS attack. The Dutta-Barua protocol is based on the well-known protocol of Burmester and Desmedt [1] and requires 2 communication rounds to establish a session key among a group of users. This protocol carries a claimed proof of its security in an adversarial model which captures UKS attacks. But, the proof simply assumes non-concurrent executions of the protocol and so does not capture our UKS attack. Hence, we not only fix the protocol but also extend its proof to the concurrent case. The replay attack on the Harn-Lin protocol is given in Section 2 while the UKS attack on the Dutta-Barua protocol is given in Section 3.

2. Harn and Lin's Group Key Transfer Protocol

This section investigates the security of Harn and Lin's group key transfer protocol HL [16]. We first review the HL protocol and cryptanalyze it by mounting a replay attack. We then show how to fix the protocol by presenting a simple countermeasure against the attack.

2.1 Protocol Description

The protocol HL consists of three phases: system initialization, user registration, and key distribution.

System initialization. KGC randomly chooses two safe primes p and q (i.e., p and q are primes such that $p' = (p-1)/2$ and $q' = (q-1)/2$ are also primes) and computes $n = pq$. n is made publicly known.

User registration. Each user is required to register at KGC to subscribe the key distribution service. During registration, KGC shares a secret (x_i, y_i) with each user U_i where x_i, $y_i \in Z_n^*$.

Key distribution. This phase constitutes the core of the protocol and is performed whenever a group of users $U_1,...,U_t$ decide to establish a common session key.

Step 1. A designated user of the group, called the initiator, sends a key-distribution request to KGC. The request carries the list of participating users $\langle U_1,...,U_t \rangle$.

Step 2. KGC broadcasts the participant list $\langle U_1,...,U_t \rangle$ in response to the request.

Step 3. Each user U_i, for $i = 1,...,t$, sends a random challenge $r_i \in Z_n^*$ to KGC.

Step 4. KGC randomly selects a session key k and constructs by interpolation a t-th degree polynomial $f(x)$ passing through the $(t+1)$ points: $(x_1, y_1 \oplus r_1), \ldots, (x_t, y_t \oplus r_t)$ and $(0, k)$. Next, KGC selects t additional points $P_1, \ldots, P_t$ that lie on the polynomial $f(x)$. KGC then computes $\beta = h(k, U_1,...,U_t, r_1,...,r_t, P_1,...,P_t)$, where h is a one-way hash function, and broadcasts $\langle \beta, r_1,...,r_t, P_1,...,P_t \rangle$ to the users. All computations with respect to $f(x)$ are performed modulo n.

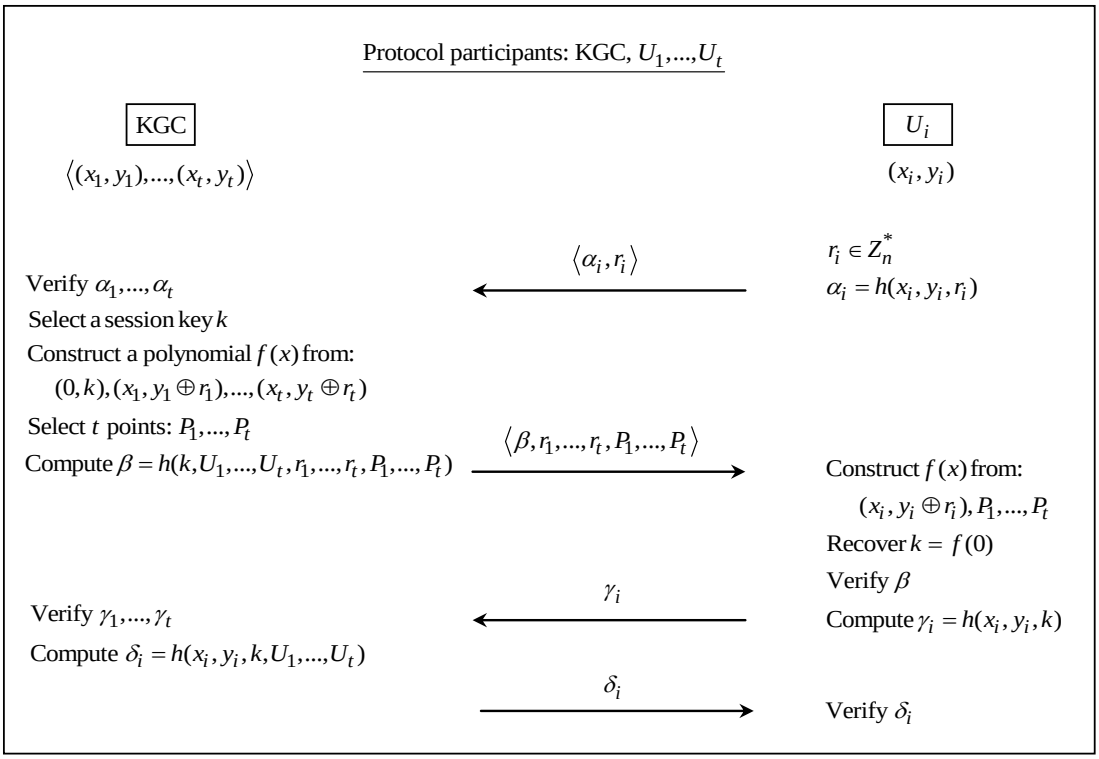


Fig. 1. The protocol HL$^+$ (described from Step 3)

Step 5. Each U_i constructs the polynomial $f(x)$ from the $(t+1)$ points: $P_1,\ldots,P_t$ and $(x_i, y_i \oplus r_i)$. Then U_i recovers the session key $k = f(0)$ and checks the correctness of β in the straightforward way. U_i aborts if the check fails.

Since the above protocol HL focuses on protecting the keying material broadcasted from KGC to users, Harn and Lin also present (in Remark 2 of [16]) how HL can be extended to provide user authentication and key confirmation. Let HL$^+$ be the extended version of HL. HL$^+$ is constructed from HL by revising Steps 3 and 4 to achieve user authentication and by adding Steps 6 and 7 to achieve key confirmation.

Step 3 (of HL$^+$). Each user U_i, for $i = 1,...,t$, selects a random challenge $r_i \in Z_n^*$, computes $\alpha_i = h(x_i, y_i, r_i)$, and sends $\langle \alpha_i, r_i\rangle$ to KGC.

Step 4 (of HL$^+$). KGC checks the correctness of each α_i in the straightforward way. KGC aborts if any of the checks fails. Otherwise, KGC continues with Step 4 of HL.

Step 6. Each U_i sends $\gamma_i = h(x_i, y_i, k)$ to KGC.

Step 7. After receiving all γ_i's, KGC sends $\delta_i = h(x_i, y_i, k, U_1, ..., U_t)$ to U_i for $i = 1,...,t$.

All other parts (including the phases of system initialization and user registration) remain unchanged between HL and HL$^+$. A high level description of HL$^+$ is given in **Fig. 1**.

2.2 Replay Attack

The fundamental security goal of a key establishment protocol is to ensure that no one other than the intended users can compute the session key. In the cases of HL and HL$^+$, this goal can

be achieved only when the secrecy of every (x_i, y_i) is guaranteed. As soon as (x_i, y_i) is disclosed, all the protocol sessions that U_i participates become completely insecure. Thus, it is important that x_i's and y_i's must not be revealed under any circumstances.

Harn and Lin claim that their protocols prevent the secret (x_i, y_i) of each U_i from being disclosed to other users, either insiders or outsiders (Theorem 3 of [16]). However, we found that this claim is wrong. Suppose that a malicious registered user U_j has a goal of finding out U_i's secret (x_i, y_i). Then U_j can achieve its goal by mounting the following attack against the protocol HL^+.

Step 0. As a preliminary step, the adversary U_j eavesdrops on a protocol session, where U_i participates, and stores the message $\langle \alpha_i, r_i \rangle$ sent by U_i in Step 3 of the session.

U_j then initiates two (either concurrent or sequential) sessions S and S' of the protocol alleging that the participants of both sessions are U_i and U_j. Once KGC responds with the participant list $\langle U_i, U_j \rangle$ in Step 2 of each session, U_j performs Step 3 of the sessions while playing dual roles of U_j itself and the victim U_i.

Step 3 of S. U_j sends the eavesdropped message $\langle \alpha_i, r_i \rangle$ to KGC as if the message is from U_i. But, U_j behaves honestly in sending its own message; U_j selects a random $r_j \in Z_n^*$, computes $\alpha_j = h(x_j, y_j, r_j)$, and sends $\langle \alpha_j, r_j \rangle$ to KGC.

Step 3 of S'. U_j replays the messages $\langle \alpha_i, r_i \rangle$ and $\langle \alpha_j, r_j \rangle$. That is, U_j sends $\langle \alpha_i, r_i \rangle$ as U_i's message and sends $\langle \alpha_j, r_j \rangle$ as its own message.

KGC cannot detect any discrepancy since α_i and α_j are both valid. Note that KGC does not check for message replays. Hence, KGC will distribute the keying materials for the sessions. Let $f(x) = a_2 x^2 + a_1 x + k$ and $f'(x) = a_2' x^2 + a_1' x + k'$ be the polynomials constructed by KGC respectively in sessions S and S'. As soon as receiving the keying materials, U_j derives these polynomials as specified in Step 5 of the protocol. Now let

$$\begin{aligned} g(x) &= f(x) - f'(x) \\ &= (a_2 - a_2')x^2 + (a_1 - a_1')x + k - k'. \end{aligned}$$

Then, $g(x_i) = 0$ and $g(x_j) = 0$ since $f(x_i) = f'(x_i) = y_i \oplus r_i$ and $f(x_j) = f'(x_j) = y_j \oplus r_j$. This implies that x_i and x_j are the two roots of the quadratic equation $(a_2 - a_2')x^2 + (a_1 - a_1')x + k - k' = 0$. It follows that

$$(a_2 - a_2')x^2 + (a_1 - a_1')x + k - k' = (a_2 - a_2')(x - x_i)(x - x_j).$$

Since $k - k' = (a_2 - a_2')x_i x_j$, we get

$$x_i = x_j^{-1}(a_2 - a_2')^{-1}(k - k'). \tag{1}$$

Here, the computations are done modulo n. Once x_i is obtained as in Eq. (1), y_i can be easily

computed from $f(x_i) = y_i \oplus r_i$. The value of y_i is different depending on whether $y_i \oplus r_i < n$ or $y_i \oplus r_i \geq n$.

$$y_i = \begin{cases} f(x_i) \oplus r_i & \text{if } y_i \oplus r_i < n \\ (f(x_i) + n) \oplus r_i & \text{otherwise.} \end{cases}$$

α_i can serve as a verifier for checking which of the two cases is true. Using (x_i, y_i) obtained as above, U_j is able to complete the protocol without the attack being noticed.

The above attack assumes, for ease of exposition, that KGC allows for the key establishment between two parties. But, this assumption is not necessary. If two-party key establishments are not allowed, U_j can collude with another malicious user U_k to mount a slight variant of the attack. Assume two sessions of the protocol, in both of which the participants are U_i, U_j and U_k. If U_j and U_k collude together and run the two sessions as in the attack above, they can construct a cubic polynomial $g(x) = (a_3 - a_3')x^3 + (a_2 - a_2')x^2 + (a_1 - a_1')x + k - k'$ such that $g(x_i) = g(x_j) = g(x_k) = 0$. Then, with x_j and x_k in hand, the adversaries can compute x_i as

$$x_i = (-1)x_j^{-1}x_k^{-1}(a_3 - a_3')^{-1}(k - k')$$

and thereby can determine y_i as above.

So far, we have seen the vulnerability of the protocol HL$^+$. As can be expected, the basic protocol HL also suffers from the same vulnerability. The attack against HL is essentially similar to the above attack, and its description is omitted due to the similarity.

Disclosure of a long-term key can easily lead to a devastating result. Suppose, for example, that the malicious user U_j gets unregistered with KGC after obtaining the long-term secret (x_i, y_i) of U_i. Then U_j, without reregistering, can completely compromise all the sessions that U_i will participate in the future.

2.3 Countermeasure

The security failure of HL$^+$ (and HL) is attributed to one obvious flaw in the protocol design: the messages sent by users in Step 3 can be replayed in different protocol sessions. This flaw allows our adversary U_j to send the same random challenges twice and thereby to construct a quadratic polynomial $g(x)$ such that $g(x_i) = g(x_j) = 0$. Fortunately, message replays can be effectively prevented if Steps 2 and 3 of the protocols are revised as follows:

Step 2 (revision). KGC selects a random $r_0 \in Z_n^*$ and broadcasts it along with the participant list $\langle U_1, ..., U_t \rangle$.

Step 3 (revision). Each user U_i, for $i = 1, ..., t$, selects a random $r_i \in Z_n^*$, computes $\alpha_i = h(x_i, y_i, r_i, r_0, U_1, ..., U_t)$, and sends $\langle \alpha_i, r_i \rangle$ to KGC.

The other steps of the protocols remain unchanged except that in Step 4 of HL, KGC has to check the correctness of α_i, for $i = 1, ..., t$, before starting to construct the polynomial $f(x)$. As a result of our modification, the protocols HL and HL$^+$ become identical except that HL$^+$ requires two additional steps (Steps 6 and 7) for key confirmation. With the modification

applied, the message $\langle \alpha_i, r_i \rangle$ eavesdropped in a protocol session can no longer be replayed in any other sessions. Hence, our attack is not valid against the improved protocols.

3. Dutta and Barua's Group Key Agreement Protocol

As previously mentioned, Dutta and Barua's group key agreement protocol DB [17] is vulnerable to an unknown key-share attack. We here reveal this security problem with the protocol DB and show how to address it. We also interpret our attack in the context of the formal proof model to invalidate the claimed proof of security for DB.

3.1 Protocol Description

Let U be a set of all users who can participate in the protocol DB. Any subset of U may decide at any point to establish a session key. Thus, a user may have several instances involved in distinct, possibly concurrent, protocol sessions. The protocol DB requires each user U_i to maintain a counter c_i whose value indicates the number of instances created by U_i. The counters of users are used in defining session identifiers, as specified in the protocol description below. The network where the users interact is assumed to be fully controlled by an active adversary who may read, intercept and fabricate any messages at will. The protocol is authenticated using signatures. Let $\sum = (\mathrm{KGen}, \mathrm{Sign}, \mathrm{Vrfy})$ be a signature scheme which is existentially unforgeable under an adaptive chosen message attack. Here, KGen is the key generation algorithm, Sign is the signature generation algorithm, and Vrfy is the signature verification algorithm. Before the protocol is ever executed, the following initialization should be performed to generate system parameters and long-term keys.

- The users in U choose a cyclic multiplicative group of prime order q and fix an arbitrary generator g of the cyclic group.
- Each user $U_i \in U$ generates its long-term verification/signing keys (VK_i, SK_i) by running KGen.
- Each $U_i \in U$ sets its counter c_i to 0. (c_i is incremented whenever a new instance of U_i is created.)

If there are n participants $U_1, ..., U_n$, DB proceeds as follows. (Throughout the protocol description, all indices are to be taken in a cycle, i.e., $U_{n+1} = U_1$, etc.)

Round 1: Each U_i chooses a random $x_i \in Z_q^*$, computes $y_i = g^{x_i}$ and $\alpha_i = \mathrm{Sign}_{SK_i}(U_i | 1 | y_i | c_i)$, and sends $\mathrm{Exp}_i = U_i | 1 | y_i | c_i | \alpha_i$ to U_{i-1} and U_{i+1}. Here, the symbol | denotes the string concatenation operation.

Round 2: Upon receiving Exp_{i-1} and Exp_{i+1}, U_i checks that $\mathrm{Vrfy}_{PK_{i-1}}(U_{i-1} | 1 | y_{i-1} | c_{i-1}, \alpha_{i-1}) = 1$ and $\mathrm{Vrfy}_{PK_{i+1}}(U_{i+1} | 1 | y_{i+1} | c_{i+1}, \alpha_{i+1}) = 1$. U_i aborts if either of two verifications fails. Otherwise, U_i computes $K_i^L = y_{i-1}^{x_i}$, $K_i^R = y_{i+1}^{x_i}$, $Y_i = K_i^R / K_i^L$, and $\beta_i = \mathrm{Sign}_{SK_i}(U_i | 2 | Y_i | c_i)$. Then U_i broadcasts $\mathrm{Div}_i = U_i | 2 | Y_i | c_i | \beta_i$.

Key computation: Each U_i checks that $\mathrm{Vrfy}_{PK_j}(U_j | 2 | Y_j | c_j, \beta_j) = 1$ for all $j \in \{1, ..., n\} \setminus \{i\}$. U_i aborts if any of the verifications fails. Otherwise, U_i computes

$$K_{i+1}^R = Y_{i+1}K_i^R,$$
$$K_{i+2}^R = Y_{i+2}K_{i+1}^R,$$
$$\cdots$$
$$K_{i+n-1}^R = Y_{i+n-1}K_{i+n-2}^R$$

U_i checks that K_{i+n-1}^R is equal to K_i^L. If not, U_i aborts. Otherwise, U_i computes the session key sk as

$$sk = K_1^R K_2^R \cdots K_n^R$$
$$= g^{x_1x_2+x_2x_3+\cdots+x_nx_1}$$

and defines the session identifier SID_i as $SID_i = \langle (U_1,c_1),\ldots,(U_n,c_n)\rangle$.

Fig. 2 shows an execution of the protocol DB when there are 4 participants U_1, U_2, U_3 and U_4.

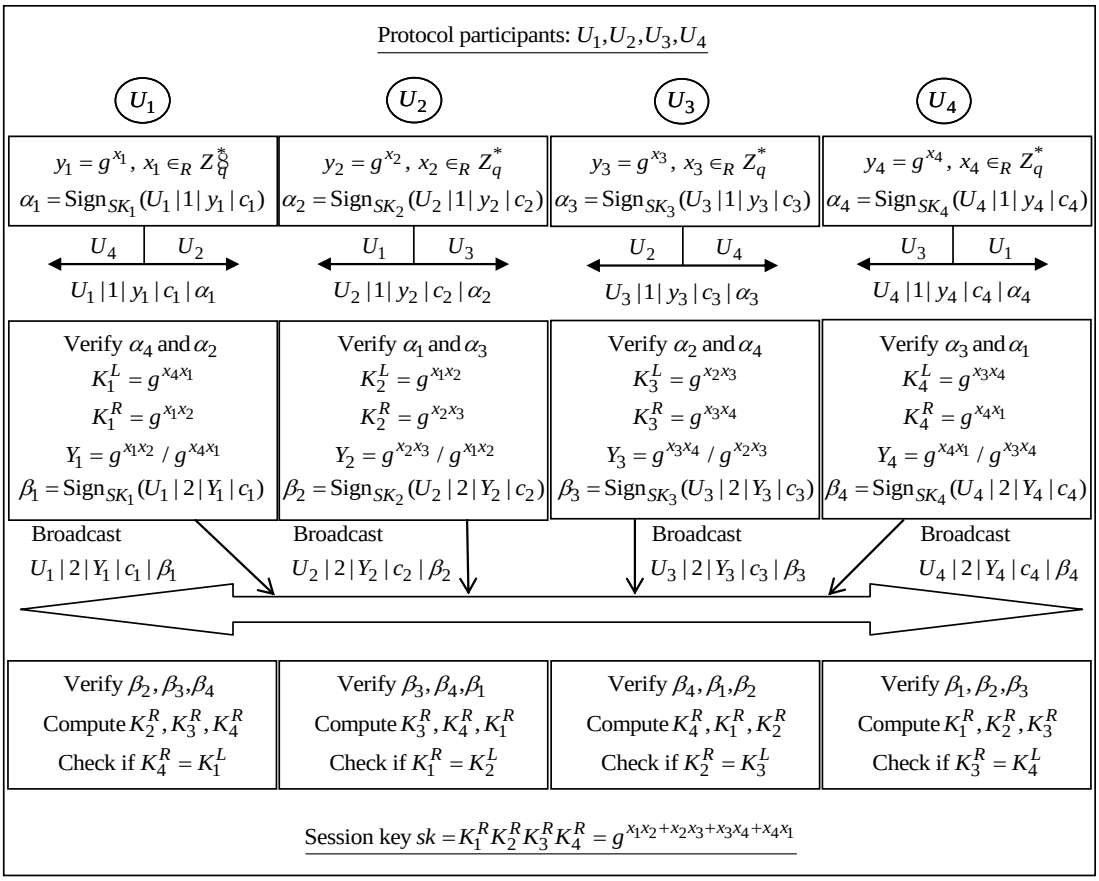


Fig. 2. An execution of the protocol DB

3.2 Unknown Key-Share Attack

We here mount an UKS attack against the protocol DB described above. Consider a protocol session S to be run by the users of group $G = \{U_1, U_2, U_3\}$ Now suppose that U_1 and U_2 accept the invitation by the malicious adversary U_A to participate in a new concurrent session

S', thus forming the group $G' = \{U_1, U_2, U_A\}$. Let U_i^S and $U_i^{S'}$ denote U_i's instances participating respectively in S and S'. Then the goal of the adversary U_A is to trick $U_1^{S'}$ and $U_2^{S'}$ into establishing a session key with U_3. The attack works as follows:

1. As session S starts, U_1^S, U_2^S and U_3 will send their first messages $\text{Exp}_1 = U_1 | 1 | y_1 | c_1 | \alpha_1$, $\text{Exp}_2 = U_2 | 1 | y_2 | c_2 | \alpha_2$ and $\text{Exp}_3 = U_3 | 1 | y_3 | c_3 | \alpha_3$, respectively. The adversary U_A intercepts Exp_3 while blocking Exp_1 and Exp_2 from reaching U_3. In other words, U_1^S and U_2^S never receive Exp_3 and U_3 receives neither Exp_1 nor Exp_2.
2. As a participant of session S', the adversary U_A sends the message $\text{Exp}_A = U_A | 1 | y_3 | c_A | \alpha_A$ to $U_1^{S'}$ and $U_2^{S'}$ and receives the messages $\text{Exp}_1' = U_1 | 1 | y_1' | c_1' | \alpha_1'$ and $\text{Exp}_2' = U_2 | 1 | y_2' | c_2' | \alpha_2'$ respectively from $U_1^{S'}$ and $U_2^{S'}$. Notice that Exp_A contains y_3 (which is generated by U_3 and is obtainable from Exp_3). We mean by this that U_A has computed its signature α_A as $\alpha_A = \text{Sign}_{SK_A}(U_A | 1 | y_3 | c_A)$.
3. U_A forwards the received messages Exp_1' and Exp_2' to U_3 as if they are sent by U_1^S and U_2^S, respectively. These messages will pass U_3's verification since α_1' (resp. α_2') is a valid signature on $U_1 | 1 | y_1' | c_1'$ (resp. $U_2 | 1 | y_2' | c_2'$) under the verification key VK_1 (resp. VK_2). Hence, U_3 will send out its second message $\text{Div}_3 = U_3 | 2 | Y_3 | c_3 | \beta_3$. If we let $y_1' = g^{x_1'}$ and $y_2' = g^{x_2'}$, then clearly
$$Y_3 = g^{x_3 x_1' - x_2' x_3}.$$
4. U_A intercepts Div_3, computes $\beta_A = \text{Sign}_{SK_A}(U_A | 2 | Y_3 | c_A)$ (using Y_3 from U_3), and sends $\text{Div}_A = U_A | 2 | Y_3 | c_A | \beta_A$ to $U_1^{S'}$ and $U_2^{S'}$. Meanwhile, $U_1^{S'}$ and $U_2^{S'}$ will send U_A their second messages $\text{Div}_1' = U_1 | 2 | Y_1' | c_1' | \beta_1'$ and $\text{Div}_2' = U_2 | 2 | Y_2' | c_2' | \beta_2'$ where
$$Y_1' = g^{x_1' x_2' - x_3 x_1'} \text{ and } Y_2' = g^{x_2' x_3 - x_1' x_2'}.$$
5. U_A forwards Div_1' and Div_2' to U_3 as if they are from U_1^S and U_2^S, respectively. These messages will pass U_3's verifications since the signatures β_1' and β_2' are both valid and K_3^L is equal to $Y_2' Y_1' K_3^R$, where $K_3^L = g^{x_2' x_3}$ and $K_3^R = g^{x_3 x_1'}$.
6. Consequently, $U_1^{S'}$, $U_2^{S'}$ and U_3 will compute the same session key
$$sk = g^{x_1' x_2' + x_2' x_3 + x_3 x_1'}.$$

At the end of the attack: (1) U_1, U_2 and U_3 have computed the same session key sk; (2) U_1 and U_2 believe that sk is shared with U_A, while in fact it is shared with U_3; (3) U_3 believes that sk is shared with U_1 and U_2. This shows that the protocol DB is vulnerable to an UKS attack when two protocol sessions are running concurrently with some joint participants.

3.3 Attacking in the Model

The protocol DB carries a claimed proof of security in a formal model of communication and adversarial capabilities. The proof model used for DB is a typical one [3] and allows the

adversary A to access all the standard oracles: Send, Execute, Reveal, Corrupt and Test. Any key establishment protocol proven secure in such a model should be resistant to UKS attacks [8]. But as we have shown, the DB protocol is vulnerable to an UKS attack. The existence of our attack means, in the context of the proof model, that there exists an adversary A whose advantage in attacking protocol DB is 1, or in other words, there exists an adversary A who can distinguish, with probability 1, random keys from real session keys. The construction of such an A is rather straightforward from the attack above, and its brief description follows:

Signing-key disclosure: First, A obtains U_A's long-term signing key SK_A by querying Corrupt(U_A).

Initiation: Next, A asks Send queries required to initiate two protocol sessions S: $G=\{U_1,U_2,U_3\}$ and S': $G'=\{U_1,U_2,U_A\}$. For example, a query of the form Send(U_1, *, $\langle U_2,U_A\rangle$) prompts an unused instance * of U_1 to initiate the protocol with U_2 and U_A. But, no instance of U_A needs to be asked this form of Send query because A will simulate by itself the actions of U_A.

Run: Now, A runs the two sessions in the exact same way as U_A did in the above-described attack. Note that A can perfectly simulate U_A's attack by asking Send queries and by using the disclosed long-term signing key SK_A. Let U_i^S and U_i^S be U_i's instances participating respectively in S and S'. Then, as in the attack above, the instances $U_1^{S'}$, $U_2^{S'}$ and U_3^S will eventually accept the same session key sk.

Session-key disclosure: A obtains the session key sk by querying either Reveal($U_1^{S'}$) or Reveal($U_2^{S'}$).

Test: The instance U_3^S is *fresh* because (1) no Corrupt query has been asked for any of the users U_1, U_2 and U_3 and (2) no Reveal query has been made for any of the instances U_1^S, U_2^S and U_3^S. Thus, A may test (i.e., ask a Test query against) the instance U_3^S. Since A knows the value of sk, the probability that A guesses correctly the bit b used by the Test oracle is 1 and so is the advantage of A in attacking DB.

3.4 Countermeasure

The vulnerability of DB to the UKS attack is attributed to the fact that U_3 cannot distinguish between the signatures generated by U_1^S (resp. U_2^S) and those generated by $U_1^{S'}$ (resp. $U_2^{S'}$). Given this observation, it is not hard to figure out how to fix the protocol. As a simple countermeasure against the attack, we recommend to change the computations of the signatures α_i and β_i to

$$\alpha_i = \mathrm{Sign}_{SK_i}(U_i \,|\, 1 \,|\, y_i \,|\, c_i \,|\, U_1 \,|\cdots|\, U_n),$$
$$\beta_i = \mathrm{Sign}_{SK_i}(U_i \,|\, 2 \,|\, Y_i \,|\, c_i \,|\, U_1 \,|\cdots|\, U_n).$$

The identities of all protocol participants are now included as part of the messages to be signed. With this modification applied, the signatures from $U_1^{S'}$ and $U_2^{S'}$ can no longer pass U_3's verification since the participants are different between the two sessions S and S'. Hence, the UKS attack is not valid against the improved protocol.

3.5 Security Proof

Since our UKS attack can be simulated in the proof model used for DB, there must be some problems with the security proof given by Dutta and Barua [17]. The problem with Dutta and Barua's proof is that it simply assumes non-concurrent executions of the protocol. We here prove the security of our improved protocol DB$^+$ by extending Dutta and Barua's proof to the concurrent case. As in [17], we use UP to denote the unauthenticated version of the protocol DB (or DB$^+$). The following theorem presents our result on the security of DB$^+$. It claims that DB$^+$ is secure against active adversaries under the security of UP against passive adversaries. (All the notations that are not defined here are taken over from [17].)

Theorem 1. For any adversary who asks q_E Execute queries and q_S Send queries with time complexity t, its advantage in breaking the security of the protocol DB$^+$ is upper bounded by

$$\mathsf{Adv}_{\mathsf{DB}^+}(t, q_E, q_S) \le Q \cdot \mathsf{Adv}_{\mathsf{UP}}(t,1) + |P| \cdot \mathsf{Adv}_{\Sigma}(t)$$

where $Q = q_E + q_S$ and P is a polynomial-sized set of potential participants.

Proof. We prove the theorem by finding a reduction from the security of protocol DB$^+$ to the security of protocol UP. As shown in [17], the unauthenticated protocol UP is provably secure against a passive adversary. Assuming an active adversary A^+ who attacks protocol DB$^+$, we construct a passive adversary A that uses A^+ in its attack on UP. As in a typical reductionist approach, the adversary A simply runs A^+ as a subroutine and answers the oracle queries of A^+ on its own. The idea in constructing A is to use the fact that in attacking DB$^+$, the adversary A^+ is able to relay messages only between user instances with the same set of participants and counters. Based on this idea, the adversary A obtains a transcript T of UP for each unique combination of participants and nonces by calling its own Execute oracle, and generates a transcript T^+ of DB$^+$ by patching T with appropriate signatures. A then uses the messages of T^+ in answering A^+'s Send queries directed to user instances which have the same participants and nonces as used in generating T^+. In this way, A^+ is limited to sending messages already contained in T^+, unless signature forgery occurs. In essence, A is ensuring that A^+'s capability of attacking protocol DB$^+$ is demonstrated only on the session key associated with the patched transcript T^+ and thus is translated directly into the capability of attacking protocol UP.

However, there exists a difficulty in constructing the passive adversary A from the active adversary A^+. Since A^+ can obtain a private signing key at any time by calling the Corrupt oracle, it may send arbitrary - but still valid - messages of its choice (i.e., messages that are not contained in the patched transcript T^+) to an instance. The problem in this case is that A cannot simulate the actions of the instance because it does not have appropriate internal data used by the instance at earlier stage. The exact same problem arises in proving security for the compiler of Katz and Yung [3]. The proof for the Katz-Yung compiler circumvents this simulation problem by letting A guess the session in which A^+ will take advantage of its only chance to access the Test oracle. For the guessed session, A handles the queries of the active adversary by calling its own Execute oracle as described above, and for all other sessions, A honestly responds by directly executing protocol DB$^+$ (i.e., without accessing the Execute oracle). Our proof follows this approach in extending Dutta and Barua's proof to the concurrent case.

The passive adversary A begins by choosing a random $\chi \in \{1,...,Q\}$ which represents a guess of the session for which A^+ will ask its Test query. A simulates the queries of the active dversary A^+ as follows:

Corrupt Queries. These queries are answered in the obvious way. Namely, A returns the long-term signing key SK_i in response to the query Corrupt(U_i).

Execute Queries. If an Execute query is not the χ-th Send/Execute query of A^+, then A simply generates by itself a transcript of an execution of DB^+ and returns this to A^+. A can do this because it knows all the signing keys of users. If an Execute query is the χ-th Send/Execute query of A^+, A proceeds exactly as in Dutta and Barua's simulation of Execute queries.

Send Queries. If a Send query is not the χ-th Send/Execute query of A^+, then A simulates on its own the actions of the instance and returns a response as needed. A can do this because it knows all the signing keys of users. If a Send query is the χ-th Send/Execute query of A^+, A proceeds exactly as in Dutta and Barua's simulation of Send queries.

Reveal Queries. If a Reveal query is asked to an instance simulated by A itself, then the appropriate session key can be computed/returned. Otherwise, A aborts and outputs a random bit since its guess χ was incorrect.

Test Queries. If the Test query is asked to an instance for which A has asked its single Execute query, then A asks its own Test query and returns the result to A^+. Otherwise, A aborts and outputs a random bit since its guess χ was incorrect.

As long as Forge does not occur and A correctly guesses χ, the above simulation for A^+ is perfect. Let Guess denote the event that A correctly guesses χ. If Forge occurs, A aborts and outputs a random bit. Let $\mathsf{Win} = \mathsf{Guess} \wedge \overline{\mathsf{Forge}}$. Then, clearly, $\Pr[\mathsf{Succ} \mid \overline{\mathsf{Win}}] = 1/2$. Now, to derive the statement of Theorem 1, we apply a series of simple modifications to the definitional equation $\mathsf{Adv}_{A,\mathsf{UP}} = \left|2\Pr_{A,\mathsf{UP}}[\mathsf{Succ}] - 1\right|$ as follows:

$$
\begin{aligned}
\mathsf{Adv}_{A,\mathsf{UP}} &= \left|2\Pr_{A,\mathsf{UP}}[\mathsf{Succ}] - 1\right| \\
&= \left|2\Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ} \wedge \mathsf{Win}] + 2\Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ} \wedge \overline{\mathsf{Win}}] - 1\right| \\
&= \left|2\Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ} \wedge \mathsf{Win}] + 2\Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ} \mid \overline{\mathsf{Win}}]\Pr_{A^+,\mathsf{DB}^+}[\overline{\mathsf{Win}}] - 1\right| \\
&= \left|2\Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ} \wedge \mathsf{Win}] + \Pr_{A^+,\mathsf{DB}^+}[\overline{\mathsf{Win}}] - 1\right| \\
&= \left|2/Q \cdot \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \Pr_{A^+,\mathsf{DB}^+}[\overline{\mathsf{Guess}} \vee \mathsf{Forge}] - 1\right| \\
&= \left|2/Q \cdot \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ}] - 2/Q \cdot \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ} \wedge \mathsf{Forge}] + \Pr_{A^+,\mathsf{DB}^+}[\overline{\mathsf{Guess}} \vee \mathsf{Forge}] - 1\right| \\
&\geq \left|2/Q \cdot \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ}] - 1/Q \cdot \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Forge}] - 1\right| \\
&\geq 1/Q \mid 2\Pr_{A^+,\mathsf{DB}^+}[\mathsf{Succ}] - 1 \mid - 1/Q \cdot \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Forge}] \\
&= 1/Q \cdot \mathsf{Adv}_{A^+,\mathsf{DB}^+} - 1/Q \cdot \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Forge}]
\end{aligned}
$$

It follows that $Q \cdot \mathsf{Adv}_{A,\mathsf{UP}} \geq \mathsf{Adv}_{A^+,\mathsf{DB}^+} - \Pr_{A^+,\mathsf{DB}^+}[\mathsf{Forge}]$. Since $\Pr_{A^+,\mathsf{DB}^+}[\mathsf{Forge}] \leq |P| \mathsf{Adv}_{\Sigma}(t)$

(see [3]), this yields the statement of the theorem.

4. Conclusion

This work has revealed the security weaknesses in two key establishment protocols: Harn and Lin's group key transfer protocol and Dutta and Barua's group key agreement protocol. The Harn-Lin protocol cannot protect the long-term keys of users while the Dutta-Barua protocol is vulnerable to an unknown key share attack. We have also suggested how the weaknesses can be eliminated. One implication of our result is that the claimed proof of security for the Dutta-Barua protocol is not rigorous enough to capture unknown key share attacks. The problem we found with Dutta and Barua's proof is that it fails to consider concurrent executions of the protocol. We have addressed this problem by extending Dutta and Barua's proof to the concurrent case.

References

[1] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," *Advances in Cryptology – EUROCRYPT 1994*, vol.950, pp.275-286, 1995. Article (CrossRef Link)

[2] E. Bresson, O. Chevassut, D. Pointcheval and J.-J. Quisquater, "Provably authenticated group Diffie-Hellman key exchange," in *Proc. of 8th ACM Conference on Computer and Communications Security*, pp.255-264, 2001. Article (CrossRef Link)

[3] J. Katz and M. Yung, "Scalable protocols for authenticated group key exchange," in *Proc. of Advances in Cryptology – CRYPTO 2003*, vol.2729, pp.110-125, 2003. Article (CrossRef Link)

[4] J. Nam, S. Kim and D. Won, "Secure group communications over combined wired and wireless networks," in *Proc. of 2nd International Conference on Trust, Privacy, and Security in Digital Business*, vol.3592, pp.90-99, 2005. Article (CrossRef Link)

[5] M. Abdalla, E. Bresson, O. Chevassut and D. Pointcheval, "Password-based group key exchange in a constant number of rounds," in *Proc. of 9th International Workshop on Practice and Theory in Public Key Cryptography*, vol.3958, pp.427-442, 2006. Article (CrossRef Link)

[6] 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, 2001. Article (CrossRef Link)

[7] H. Krawczyk, "HMQV: a High-Performance secure Diffie-Hellman protocol," *Advances in Cryptology – CRYPTO 2005*, vol.3621, pp.546-566, 2005. Article (CrossRef Link)

[8] K.-K. Choo, C. Boyd and Y. Hitchcock, "Errors in computational complexity proofs for protocols," *Advances in Cryptology – ASIACRYPT 2005*, vol.3788, pp.624-643, 2005. Article (CrossRef Link)

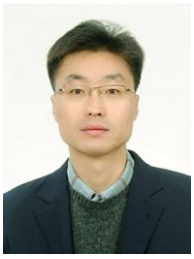
[9] J. Nam, S. Kim and D. Won, "Attack on the Sun-Chen-Hwang's three-party key agreement protocols using passwords," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E89-A, no.1, pp.209-212, 2006.

[10] J. Nam, J. Paik, U. Kim and D. Won, "Security enhancement to a password-authenticated group key exchange protocol for mobile ad-hoc networks," *IEEE Communications Letters*, vol.12, no.2, pp.127-129, 2008. Article (CrossRef Link)

[11] B. S. Kaliski, "An Unknown Key-Share attack on the MQV key agreement Protocol," *ACM Transactions on Information and System Security*, vol.4, no.3, pp.275-288, 2001. Article (CrossRef Link)

[12] W. Diffie, P. Oorschot and M. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes, and Cryptography*, vol.2, no.2, pp.107-125, 1992. Article (CrossRef Link)

[13] S. Blake-Wilson and A. Menezes, "Unknown Key-Share attacks on the Station-to-Station (STS) protocol," in *Proc. of 2nd International Workshop on Practice and Theory in Public Key Cryptography*, vol.1560, pp.154-170, 1999. Article (CrossRef Link)
[14] J. Baek and K. Kim, "Remarks on the unknown Key-Share attacks," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E83-A, no.12, pp.2766-2769, 2000.
[15] K. Shim, "Cryptanalysis of mutual authentication and key exchange for low power wireless communications," *IEEE Communications Letters*, vol.7, no.5, pp.248-250, 2003. Article (CrossRef Link)
[16] L. Harn and C. Lin, "Authenticated group key transfer protocol based on secret sharing," *IEEE Transactions on Computers*, vol.59, no.6, pp.842-846, 2010. Article (CrossRef Link)
[17] R. Dutta and R. Barua, "Provably secure constant round contributory group key agreement in dynamic setting," *IEEE Transactions on Information Theory*, vol.54, no.5, pp.2007-2025, 2008. Article (CrossRef Link)
[18] A. Shamir, "How to share a secret," *Communications of the ACM*, vol.22, no.11,pp. 612-613, 1979. Article (CrossRef Link)



Junghyun Nam received the B.E. degree in Information Engineering from Sungkyunkwan University, Korea, in 1997. He received his M.S. degree in Computer Science from University of Louisiana, Lafayette, in 2002, and the Ph.D. degree in Computer Engineering from Sungkyunkwan University, Korea, in 2006. He is now an associate professor in Konkuk University, Korea. His research interests include cryptography and computer security.

Moonseong Kim received the M.S. degree in Mathematics, August 2002 and the Ph.D. degree in Electrical and Computer Engineering, February 2007 both from Sungkyunkwan University, Korea. He was a research professor at Sungkyunkwan University in 2007. From December 2007 to October 2009, he was a visiting scholar in ECE and CSE, Michigan State University, USA. Since October 2009, he has been a patent examiner in Information and Communication Examination Bureau, Korean Intellectual Property Office (KIPO), Korea. His research interests include wired/wireless networking, sensor networking, mobile computing, network security protocols, and simulations/numerical analysis.

Juryon Paik received the B.E. degree in Information Engineering from Sungkyunkwan University, Korea, in 1997. She received her M.E. and Ph.D. degrees in Computer Engineering from Sungkyunkwan University in 2005 and 2008, respectively. Currently, she is a research professor at the Department of Computer Engineering, Sungkyunkwan University. Her research interests include XML mining, semantic mining, and web search engines.

Dongho Won received his B.E., M.E., and Ph.D. degrees from Sungkyunkwan University in 1976, 1978, and 1988, respectively. After working at ETRI (Electronics & Telecommunications Research Institute) from 1978 to 1980, he joined Sungkyunkwan University in 1982, where he is currently Professor of School of Information and Communication Engineering. In the year 2002, he served as the President of KIISC (Korea Institute of Information Security & Cryptology). He was the Program Committee Chairman of the 8th International Conference on Information Security and Cryptology (ICISC 2005). His research interests are on cryptology and information security.