JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 28, 829-857 (2012)

Scalable Group Key Exchange for Securing Distributed Operating Systems*

JUNGHYUN NAM[1], MINKYU PARK[1,+], SANGCHUL HAN[1],
JURYON PAIK[2] AND DONGHO WON[2]
[1]*Department of Computer Engineering*
Konkuk University
Chungju-si, 380-701 Korea
[2]*Department of Computer Engineering*
Sungkyunkwan University
Suwon-si, 440-746 Korea

Distributed operating systems are designed to run over a collection of computing nodes that are spatially disseminated and communicate through a computer network. The computing nodes interact collaboratively with each other in order to pursue a common purpose. Protecting group communication between the collaborating nodes against potential attacks is one of the major challenges faced by the designers of distributed operating systems. This challenge can be effectively addressed by a group key exchange (GKE) protocol which allows a group of communicating parties to build a secure multicast channel over an insecure network. In this work, we propose a scalable GKE protocol that achieves both constant round complexity and logarithmic computation complexity. Our GKE protocol achieves its scalability by employing a complete binary tree structure combined with a so-called "nonce-chained authentication technique". Besides the scalability, our protocol features provable security against active adversaries in a well-defined communication model.

Keywords: distributed operating system, security, group key exchange, session key, binary tree

1. INTRODUCTION

With the increasing ubiquity of computer networks, distributed computing is gaining tremendous popularity and its applications continue to proliferate in a large number of domains. In general, distributed computing is a type of computing in which different components comprising an application software run on different machines connected to a communication network. Typical distributed applications include video/audio teleconferencing, multiplayer online games, grid computing, and replicated databases. A collection of interconnected computers working collaboratively on tasks, called a distributed system, often requires a distributed operating system (DOS) to provide services for execution of various application software. DOSs must provide all the usual services expected from any operating systems. But the inherent nature of distribution requires a DOS to offer many additional services including distributed system security. DOSs operate over a group of computers which usually communicate through a public network rather than a private

Received May 31, 2011; accepted March 31, 2012.
Communicated by Jiman Hong, Junyoung Heo and Tei-Wei Kuo.
* 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) and by Ministry of Culture, Sports and Tourism (MCST) and from Korea Copyright Commission in 2011.
+ Corresponding author.

network. Confidential and sensitive information transmitted over a public network may be sniffed, intercepted and exploited by a potential adversary. It is thus required that a DOS should be equipped with a security tool for protecting group communication between distributed computers. This requirement can be met effectively by employing a group key exchange (GKE) protocol. As a cryptographic primitive, a GKE protocol specifies how a group of communicating parties can establish a common secret key (called a *session key*). Any set of parties, who desire confidential and integrity-preserving communication, can first generate a common session key by running a GKE protocol. Once a session key has been established, the parties can use this key to encrypt and/or authenticate their subsequent multicast messages.

The first priority in designing a key exchange protocol is placed on ensuring the security of session keys to be established by 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 unfortunately, the experience has shown that the design of secure key exchange protocols is notoriously difficult; there is a long history of protocols for this domain being proposed and later found to be flawed. 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. This concern has prompted active research on formal models for security analysis of key exchange protocols, and highlighted the importance of proofs of protocol security in a well-defined model. Although rigorously proving a protocol secure can often be a lengthy and complicated task, proofs are advocated as invaluable tools for obtaining a high level of assurance in the security of key exchange protocols [1-8].

Efficiency is another important consideration in designing key exchange protocols. In particular, it may become a critical practical issue in the group setting where quite a large number of parties are likely to get involved in session key generation. The efficiency of a GKE protocol is typically measured with respect to communication cost as well as computation cost incurred by the protocol. Three common measures for gauging the communication cost of a protocol are (1) the *round complexity*, the number of rounds until the protocol terminates; (2) the *message complexity*, the maximum number of messages both sent and received per user in the protocol; and (3) the *bit complexity*, the maximum number of bits (*i.e.*, the maximum combined length of messages) both sent and received per user in the protocol. In order for a GKE protocol to be scalable, it is desirable in many real-life applications that the protocol be able to complete in a constant number of rounds. The computation cost of a protocol is directly related to the *computation complexity* which we define as the maximum amount of computation done by a single user in the protocol. By computation, we do not mean simple traverses of the identities of the protocol participants, but mean any kinds of cryptographic operations such as public-key and symmetric-key operations, modular arithmetic operations, hash function evaluations, *etc*. Although the above definitions of various complexities are largely based on those given in [5], there is a noteworthy difference in defining message and bit complexities. Our definitions for these complexities count both the sent and received traffics whereas those in [5] consider only the sent one. We believe this modification provides a more accurate way to measure the communication efficiency of any distributed protocols.

1.1 Motivation

Efficient and secure generation of session keys for large groups is a difficult problem that needs more work to solve it. The difficulty of the problem is well indicated by the fact that it took nearly two decades before we got the first provably-authenticated GKE protocol [1] even with round complexity $O(n)$ in a group of size n. Still up to now, there are only a very limited number of constant-round protocols [2, 5, 6, 9] carrying a claimed proof of security against active adversaries in a formal model. However, all these constant-round protocols suffer from the number of public-key operations that scales linearly in group size, and thus exhibit $O(n)$ computation complexity under the definition above. These best-known protocols are categorized as key agreement protocols, but the situation is not much different for authenticated key transport protocols [10-12]. Indeed, we are unaware of any, provably secure or not, authenticated GKE protocols achieving both constant round complexity and logarithmic computation complexity. The protocols of [2, 10-12] requires one distinct user to perform $O(n)$ modular exponentiations or public-key encryptions. The other protocols from [5, 6, 9] is all a novel extension of the protocol (*i.e.*, protocol 3) by Burmester and Desmedt [13], but commonly require each user to perform $O(n)$ signature verifications. For moderate size groups, these previous solutions are clearly appealing. But for large groups, many applications will likely demand a protocol whose computation complexity scales logarithmically with group size. It is this observation that prompted the present work aimed at designing an authenticated GKE protocol which scales very well for large groups.

1.2 Contribution

The result of this work is the first forward-secure authenticated GKE protocol that achieves $O(1)$ round complexity and $O(\log n)$ computation complexity. In Tables 1 and 2, we summarize the computation and communication requirements of our protocol and other authenticated GKE protocols [2, 5, 10, 11] (By the tables, we are not arguing that one is overall superior to another, but meant to provide an asymptotic analysis for comparing scalability of different protocols.) Like the protocols of [10, 11], our GKE protocol is categorized as a key transport protocol. The protocol of [2] features optimal round complexity [14], but lacks perfect forward secrecy. As Table 1 shows, the maximum computation rate per user is bounded by $O(\log n)$ in our protocol, whereas this rate per user rises up to $O(n)$ in the other protocols. Thus from a theoretical point of view, our main contribution is to show the possibility of achieving logarithmic computation complexity in constructing forward-secure constant-round protocols for authenticated group key exchange. However, it is also important from a practical viewpoint to notice that for reasonable values of n, the actual computation in our protocol can be heavier than that in the other protocols.

Our result can be even stronger in a broadcast network model, where each message sent is assumed to be received by all parties in the network. In the broadcast model, our protocol distinguishes itself from the other protocols in that it achieves $O(\log n)$ message and bit complexities as shown in Table 2. (Recall that both the sent and received traffics are considered for estimating message and bit complexities.) Thus if we assume a broadcast network, our protocol can be regarded as the first forward-secure authenticated GKE

Table 1. Computation requirements of authenticated GKE protocols.

	Exp	Sig/Dec	Ver/Enc	Div	Mul
Boyd-Nieto [2]		$O(1)$/	/$O(n)$		
Katz-Yung [5]	$O(1)$	$O(1)$/	$O(n)$/	$O(1)$	$O(n\log n)$
Hirose-Yoshida [10]	$O(n)$	$O(n)$/	$O(n)$/		$O(n)$
Mayer-Yung [11]		$O(1)$/	$O(n)$/$O(n)$		
Here	$O(\log n)$	$O(1)$/	$O(\log n)$/	$O(1)$	$O(\log n)$

Note: "Mayer-Yung [11]" refers to *a-MKT with Consistency* 1 of [11].
Exp: the maximum number of modular exponentiations performed per user.
Sig/Dec: the maximum numbers of signature generations and public-key decryptions performed per user.
Ver/Enc: the maximum numbers of signature verifications and public-key encryptions performed per user.
Div: the maximum number of modular divisions performed per user.
Mul: the maximum number of modular multiplications performed per user.

Table 2. Communication requirements of authenticated GKE protocols.

	Rounds	Messages		Bits	
		PtP	Broadcast	PtP	Broadcast
Boyd-Nieto [2]	1	$O(n)$	$O(n)$	$O(n^2)$	$O(n)$
Katz-Yung [5]	3	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Hirose-Yoshida [10]	3	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Mayer-Yung [11]	4	$O(n)$	$O(n)$	$O(n^2)$	$O(n)$
Here	3	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$

Note: "Mayer-Yung [11]" refers to *a-MKT with Consistency* 1 of [11].
Rounds: the number of communication rounds required to complete the protocol.
Messages: the maximum number of messages both sent and received per user.
Bits: the maximum number of bits both sent and received per user.
PtP: the point-to-point network model.
Broadcast: the broadcast network model.

protocol that not only achieves $O(1)$ round complexity but also bounds all other complexities (*i.e.*, bit, message, and computation complexities) by $O(\log n)$.

Furthermore, our protocol is provably secure against a powerful active adversary under the decisional Diffie-Hellman assumption. We provide a rigorous proof of security for the protocol in a refinement of the standard security model [1, 2, 5, 6, 9]. From the standpoint of the adversary's capabilities, our security model is a unique combination of previous results from [1, 15-17], which are in turn based on earlier work of Bellare, Pointcheval, and Rogaway [18]. In particular, our model maximizes the overall attacking ability of the adversary in two ways. Firstly, we allow the adversary to query the Test oracle as many times as it wants [15]. Secondly, we incorporate *strong corruption* [18] into the model by allowing the adversary to ask users to release any short-term and long-term secret information (see section 2 of [19] for a detailed discussion on this). Our security proof of course captures important security notions of perfect forward secrecy and known key security. In addition since security is proved in the strong corruption model, our protocol also guarantees that the release of short-term secrets used in some sessions does not jeopardize the security of other sessions.

1.3 Tree-Based Protocols

A number of GKE protocols, including ours, have leveraged a tree structure in order to provide better scalability. As is widely known, the protocols of Wallner *et al.* [20] and Wong *et al.* [21] are based on a logical tree of key encryption keys. These protocols make substantial progress towards scalable key management in very large groups, by reducing the cost of rekeying operations associated with group updates from $O(n)$ to $O(\log n)$. But, these group rekeying methods (and their many optimizations and extensions) fail to provide (perfect) forward secrecy, requiring long-term pairwise secure channels between a key server and all users.

The approach using logical key trees has been extended by Kim *et al.* [22, 23] to the forward-secure case. Their protocols require no secure channels of any kind and offer distributed functionality. Later, Lee *et al.* [24] present a pairing-based variant of the TGDH protocol of [22]. All these works [22-24], however, provide no explicit treatment of key exchange for initial group formation, focusing only on key updates upon group membership changes.

Ren *et al.* [25] make use of a binary key tree in their generic construction where an authenticated GKE protocol is built upon any authenticated protocol for two-party key exchange. Barua *et al.* [26] and Dutta *et al.* [27] construct their protocols by combining a ternary tree structure with the one-round tripartite protocol of Joux [28]. Back in 1994, Burmester and Desmedt [13] also proposed a tree-based GKE protocol. This protocol (*i.e.*, protocol 2 of [13]) seems to be the first GKE protocol utilizing a binary tree structure, and differs from all other protocols mentioned here in that there exists a bijective mapping between protocol participants and tree nodes. But, this protocol, in common with other protocols from [25-27], has round complexity $O(\log n)$, in contrast to $O(1)$ in our protocol.

After the first version of this paper was written, we became aware that in 1996, Burmester and Desmedt [29] presented a graph-based protocol called CKDS. The CKDS protocol (more precisely, the multicast version of CKDS) has a potential to achieve the same level of complexities as our protocol, in the sense that the minimum spanning tree of the graph it used could have a height of $O(\log n)$. But unlike our provably-authenticated protocol, this protocol assumes a passive adversary and justifies its security on purely heuristic grounds without providing no formal analysis of security.

1.4 Organization

The remainder of this paper is organized as follows. Section 2 presents our two-round GKE protocol secure against passive adversaries. This unauthenticated protocol is then transformed into a three-round authenticated protocol in section 4. The transformation is done by using a modified version of the compiler presented by Katz and Yung [5]. Concluding this work, section 6 poses a challenging open problem. Proofs of security for the unauthenticated and authenticated protocols are provided in sections 3 and 5, respectively.

2. A SCALABLE GKE PROTOCOL

This section presents a new group key exchange protocol called SKE (Scalable Key Exchange). Let $\mathcal{G} = \{U_1, \ldots, U_n\}$ be a set of n users wishing to establish a session key

among themselves. As stated in the Introduction, our goal is to design a forward-secure GKE protocol with round complexity $O(1)$ and computation complexity $O(\log n)$. Towards the goal, we arrange the users in a complete binary tree where all the levels, except perhaps the last, are full; while on the last level, any missing nodes are to the right of all the nodes that are present. Fig. 1 shows an example of a complete binary tree of height 3 with 6 leaves and 6 internal nodes. Users in $\mathcal{G}$ are placed at nodes in a straightforward way that U_i has U_{2i} as its left child and U_{2i+1} as its right child. Let N_i denote the node at which U_i is positioned and let $\mathcal{G}_i$ denote the subgroup consisting of all users located in the subtree rooted at node N_i. Each internal node N_i is associated with a node key k_i. In the protocol, the node key k_i is first generated by U_i and then shared as the subgroup key among the users in $\mathcal{G}_i$. Accordingly, k_1 serves as the group key (*i.e.*, session key) shared by all users in $\mathcal{G}$.

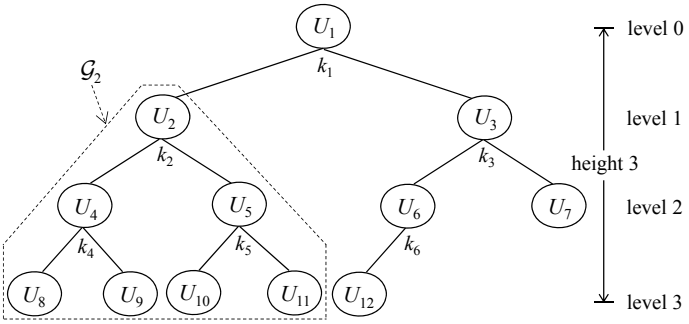


Fig. 1. A complete binary tree for $\mathcal{G} = \{U_1, \ldots, U_{12}\}$.

2.1 Decisional Diffie-Hellman (DDH) Assumption

Let $\mathbb{G}$ be a cyclic (multiplicative) group of prime order q. Since the order of $\mathbb{G}$ is prime, all the elements of $\mathbb{G}$, except 1, are generators of $\mathbb{G}$. Let g be a random fixed generator of $\mathbb{G}$ and let x, y, z be randomly chosen elements in $\mathbb{Z}_q^*$ where $z \neq xy$. Informally stated, the DDH problem for $\mathbb{G}$ is to distinguish between the distributions of g^x, g^y, g^{xy} and g^x, g^y, g^z, and the DDH assumption is said to hold in $\mathbb{G}$ if it is computationally infeasible to solve the DDH problem for $\mathbb{G}$. More formally, we define the advantage of $\mathcal{D}$ in solving the DDH problem for $\mathbb{G}$ as $\mathrm{Adv}_{\mathbb{G}}^{\mathrm{ddh}}(\mathcal{D}) = |\Pr[\mathcal{D}(\mathbb{G}, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{D}(\mathbb{G}, g, g^x, g^y, g^z) = 1]|$. We say that the DDH assumption holds in $\mathbb{G}$ (or equivalently, the DDH problem is hard in $\mathbb{G}$) if $\mathrm{Adv}_{\mathbb{G}}^{\mathrm{ddh}}(\mathcal{D})$ is negligible for all probabilistic polynomial time algorithms $\mathcal{D}$. We denote by $\mathrm{Adv}_{\mathbb{G}}^{\mathrm{ddh}}(t)$ the maximum value of $\mathrm{Adv}_{\mathbb{G}}^{\mathrm{ddh}}(\mathcal{D})$ over all $\mathcal{D}$ running in time at most t.

2.2 Description of SKE

In describing the protocol, we assume that the following public information has been fixed in advance and is known to all parties in the network: (1) the structure of the tree and the users' positions within the tree, (2) a cyclic multiplicative group $\mathbb{G}$ of prime order q, where the DDH assumption holds, and a generator g of $\mathbb{G}$, and (3) a function I

mapping elements of $\mathbb{G}$ to elements of $\mathbb{Z}_q$. A standard way of generating $\mathbb{G}$ where the DDH assumption is assumed to hold is to choose two primes p, q such that $p = kq + 1$ for some small $k \in \mathbb{N}$ (*e.g.*, $k = 2$) and let $\mathbb{G}$ be the subgroup of order q in $\mathbb{Z}_p^*$. For our purpose, we require that I: $\mathbb{G} \to \mathbb{Z}_q$ be bijective and (for any element in $\mathbb{G}$) efficiently computable. Whether there are appropriate bijections from $\mathbb{G}$ into $\mathbb{Z}_q$ depends on the group $\mathbb{G}$. If p is a safe prime (*i.e.*, $p = 2q + 1$), then such a bijection I can be constructed as follows,

$$I(x) = \begin{cases} x & \text{if } x \le q \\ p - x & \text{if } q < x < p \end{cases}.$$

The protocol SKE runs in two communication rounds.

Round 1: All users, except U_1 at the root, send a message to their parent as follows,

– Each user U_i at a leaf node chooses a random $r_i \in \mathbb{Z}_q$, computes $z_i = g^{r_i}$, and sends $M_i^1 = U_i \,|\, 1 \,|\, z_i$ to its parent.
– Each user U_i at an internal node chooses two random s_i, $t_i \in \mathbb{Z}_q$, computes $k_i = g^{s_i t_i}$, $r_i = I(k_i)$ and $z_i = g^{r_i}$, and sends $M_i^1 = U_i \,|\, 1 \,|\, z_i$ to its parent.

Meanwhile, U_1 chooses two random s_1, $t_1 \in \mathbb{Z}_q$ and computes $k_1 = g^{s_1 t_1}$.

Round 2: Each internal user U_i (including U_1) sends a message to its descendants (*i.e.*, the users in $\mathcal{G}_i \setminus \{U_i\}$) as follows,

- First, U_i computes $x_{2i} = z_{2i}^{s_i}$ and $y_{2i} = k_i x_{2i}^{-1}$. If U_i has the right child (this is the case for all internal users, except possibly for the last one), it also computes $x_{2i+1} = z_{2i+1}^{s_i}$ and $y_{2i+1} = k_i x_{2i+1}^{-1}$.
- Then, U_i computes $w_i = g^{s_i}$ and sends $M_i^2 = U_i \,|\, 2 \,|\, w_i \,|\, y_{2i} \,|\, y_{2i+1}$ (or $M_i^2 = U_i \,|\, 2 \,|\, w_i \,|\, y_{2i}$ if U_i has only the left child) to its descendants.

Key computation: Using messages from ancestors, each user $U_i \neq U_1$ computes every node key k_j on the path from the parent to the root as follows,

$$\left.\begin{array}{l} \text{while } i \ge 2 \\ \quad \text{do } j \leftarrow \lfloor i/2 \rfloor \\ \qquad k_j = y_i \cdot w_j^{r_i} \\ \qquad \text{if } j > 1 \\ \qquad\quad \text{then } r_j = I(k_j) \\ \qquad i \leftarrow j \end{array}\right\}.$$

Having derived the root node key k_1, all users in $\mathcal{G}$ simply set the session key K equal to k_1.

Consider, for example, the user U_{11} in Fig. 1. (For simplicity, let us exclude user identities and sequence numbers from consideration.) U_{11} sends $z_{11} = g^{r_{11}}$ to U_5 in the first round and receives $w_5 \,|\, y_{10} \,|\, y_{11}$, $w_2 \,|\, y_4 \,|\, y_5$ and $w_1 \,|\, y_2 \,|\, y_3$ respectively from U_5, U_2 and U_1 in the second round. U_{11} then computes, in sequence, $k_5 = y_{11} \cdot w_5^{r_{11}}$, $r_5 = I(k_5)$, $k_2 = y_5 \cdot w_2^{r_5}$, r_2

$= I(k_2)$ and $k_1 = y_2 \cdot w_1^{r_2}$. Finally, U_{11} sets its session key to k_1. Meanwhile, the internal user U_5 sends $z_5 = g^{r_5}$, where $r_5 = I(k_5)$ and $k_5 = g^{s_5 t_5}$, to U_2 in the first round. Then in the second round, U_5 sends $w_5 | y_{10} | y_{11}$ to U_{10} and U_{11}, and receives $w_2 | y_4 | y_5$ and $w_1 | y_2 | y_3$ respectively from U_2 and U_1. Finally, U_5 computes $k_2 = y_5 \cdot w_2^{r_5}$, $r_2 = I(k_2)$ and $k_1 = y_2 \cdot w_1^{r_2}$ and sets its session key to k_1.

It can be easily verified that the SKE protocol achieves the complexity bounds claimed in section 1. Notice in SKE that the users at level ℓ perform about ℓ operations of any kind. This means that the maximum amount of computation done by a user scales linearly with the height of the tree, *i.e.*, logarithmically with the number of users in $\mathcal{G}$. Hence, the computation complexity of SKE is $O(\log n)$ as claimed. The message and bit complexities of SKE in a broadcast network are also $O(\log n)$, since the maximum numbers of messages and bits both sent and received by a user increase linearly as the tree height grows. In a point-to-point network, the message and bit complexities rise up to $O(n)$ because the root user has to send a same message $n - 1$ times. (Hereafter, for brevity of exposition, all statements regarding message and bit complexities assumes a broadcast network.)

Of course, the SKE protocol is not authenticated, and is categorized as a key transport protocol because the session key is generated by one user (*i.e.*, U_1) and then transferred to all other users. In section 4, we will show how to convert this unauthenticated protocol into an authenticated one without compromising the protocol's scalability.

2.3 Security Result for SKE

The following theorem presents our result on the security of protocol SKE. It says, roughly, that the group key exchange protocol SKE is secure against passive adversaries under the DDH assumption for $\mathbb{G}$. (All the security claims made in this paper are proved in the context of the adversarial model of [19]. Thus, see section 2 of [19] for all the definitions and notations that are not given in this paper.)

Theorem 1 Let $Q = (q_{\text{exec}}, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$. Then for any adversary with time complexity at most t and query complexity at most Q, its advantage in breaking the security of protocol SKE is upper bounded by:

$$\text{Adv}_{\text{SKE}}(t, Q) = q_{\text{test}} q_{\text{exec}} (2^{\lfloor \log|\mathcal{U}|+1 \rfloor} - 1)\text{Adv}_{\mathbb{G}}^{\text{ddh}}(t')$$

where $t' = t + O(|\mathcal{U}| q_{\text{exec}} t_{\text{SKE}})$ and t_{SKE} is the time required for execution of protocol SKE by any party.

At a high level, the proof proceeds by a mathematical induction on the height of the binary tree used in protocol SKE. The complete proof of Theorem 1 is provided in section 3.

3. SECURITY PROOF FOR SKE

3.1 Single vs. Multiple Test Queries

Our security model allows the adversary to call the Test oracle multiple times, as

long as the tested instances are fresh and no two of them are partnered together. Here we show that in attacking any key exchange protocol, the maximum advantage obtainable by a passive adversary asking q_{test}. Test queries is at most q_{test} times the maximum advantage that a passive adversary can obtain when it is restricted to access the Test oracle only once.

Lemma 1 For any key exchange protocol P,

$$\text{Adv}_P(t, Q) \le q_{\text{test}} \cdot \text{Adv}_P(t, Q')$$

where $Q = (q_{\text{exec}}, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$ and $Q' = (q_{\text{exec}}, 0, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$.

Proof: Let $\mathcal{A}$ be an adversary attacking a key exchange protocol P, with time complexity t and query complexity $Q = (q_{\text{exec}}, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$. Recall that the advantage of $\mathcal{A}$ in breaking the security of protocol P is the probability that $\mathcal{A}$ outputs 1 in the real experiment $\text{Exp}_P^{\text{real}}(\mathcal{A})$ minus the probability that $\mathcal{A}$ outputs 1 in the random experiment $\text{Exp}_P^{\text{rand}}(\mathcal{A})$. Namely,

$$\text{Adv}_P(\mathcal{A}) = |\Pr[\text{Exp}_P^{\text{real}}(\mathcal{A}) = 1] - \Pr[\text{Exp}_P^{\text{rand}}(\mathcal{A}) = 1]|.$$

The proof proceeds by a standard hybrid argument [30]. Consider a sequence of $q_{\text{test}} + 1$ hybrid experiments $\text{Exp}_P^{\xi}(\mathcal{A})$, $0 \le \xi \le q_{\text{test}}$, where each $\text{Exp}_P^{\xi}(\mathcal{A})$ is defined as follows.

Experiment $\text{Exp}_P^{\xi}(\mathcal{A})$:
(1) The adversary $\mathcal{A}$ interacts with the oracles, asking queries as many times as it wants. The interaction proceeds as specified in the model except that $\mathcal{A}$'s queries to the Test oracle are handled differently, as follows:
The first ξ queries to the Test oracle are answered with a random session key and all remaining Test queries are answered with the real session key.
(2) Some time after $\mathcal{A}$ asked all its queries, it outputs 0 or 1 as the outcome of the experiment.

Clearly, the experiments $\text{Exp}_P^{0}(\mathcal{A})$ and $\text{Exp}_P^{q_{\text{text}}}(\mathcal{A})$ at the extremes of the sequence are identical to $\text{Exp}_P^{\text{real}}(\mathcal{A})$ and $\text{Exp}_P^{\text{rand}}(\mathcal{A})$, respectively. Notice that as we move from $\text{Exp}_P^{\xi-1}(\mathcal{A})$ to $\text{Exp}_P^{\xi}(\mathcal{A})$ in the sequence, we change the response of ξth Test query from the real session key to a random session key. Since there are q_{test} such moves from $\text{Exp}_P^{\text{real}}(\mathcal{A})$ to $\text{Exp}_P^{\text{rand}}(\mathcal{A})$, the inequality of the lemma follows immediately if we prove that the difference between the probabilities that $\mathcal{A}$ outputs 1 in any two neighboring experiments $\text{Exp}_P^{\xi-1}(\mathcal{A})$ and $\text{Exp}_P^{\xi}(\mathcal{A})$ is at most $\text{Adv}_P(t, Q')$ where $Q' = (q_{\text{exec}}, 0, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$. That is, to complete the proof, it suffices to show that for every $1 \le \xi \le q_{\text{test}}$,

$$|\Pr[\text{Exp}_P^{\xi-1}(\mathcal{A}) = 1] - \Pr[\text{Exp}_P^{\xi}(\mathcal{A}) = 1]| \le \text{Adv}_P(t, Q'). \quad (1)$$

For this purpose, let $\varepsilon = |\Pr[\text{Exp}_P^{\xi-1}(\mathcal{A}) = 1] - \Pr[\text{Exp}_P^{\xi}(\mathcal{A}) = 1]|$. Then, using the adversary

$\mathcal{A}$, we construct an adversary $\mathcal{A}_\xi$ attacking the protocol P, with advantage ε, time complexity t, and query complexity $Q' = (q_{\text{exec}}, 0, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$. $\mathcal{A}_\xi$ begins by invoking adversary $\mathcal{A}$, then proceeds to answer the oracle queries of $\mathcal{A}$ using its own oracle queries, and finally ends by outputting whatever bit $\mathcal{A}$ eventually outputs. $\mathcal{A}_\xi$ answers the oracle queries of $\mathcal{A}$ as follows,

- When $\mathcal{A}$ asks a query to the Execute, Reveal, Corrupt, or Dump oracle, $\mathcal{A}_\xi$ answers it in a straightforward way by sending the same query to its own corresponding oracle and then simply forwarding to $\mathcal{A}$ the outcome of its oracle query.
- If $\mathcal{A}$ queries the Test oracle, then there are three cases to handle:
  - For the first $\xi - 1$ Test queries, $\mathcal{A}_\xi$ answers them with a random session key.
  - On the ξth Test query, $\mathcal{A}_\xi$ asks a query to its own Test oracle and returns the result it receives.
  - For all the remaining Test queries, $\mathcal{A}_\xi$ answers them with the real session key by accessing its own Reveal oracle.

It is easy to see that $\mathcal{A}_\xi$ has time complexity t and query complexity at most $Q' = (q_{\text{exec}}, 0, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$. Moreover, the following is clear from the simulation above.

- The probability that $\mathcal{A}_\xi$ outputs 1 when its Test oracle returns the real session key is exactly $\Pr[\text{Exp}_P^{\xi-1}(\mathcal{A}) = 1]$.
- The probability that $\mathcal{A}_\xi$ outputs 1 when its Test oracle returns a random session key is identical to $\Pr[\text{Exp}_P^{\xi}(\mathcal{A}) = 1]$.

The advantage of $\mathcal{A}_\xi$ in attacking protocol P, $\text{Adv}_P(\mathcal{A}_\xi)$, is therefore exactly $\varepsilon = |\Pr[\text{Exp}_P^{\xi-1}(\mathcal{A}) = 1] - \Pr[\text{Exp}_P^{\xi}(\mathcal{A}) = 1]|$. Since $\text{Adv}_P(\mathcal{A}_\xi) \le \text{Adv}_P(t, Q')$ by definition, we obtain Eq. (1) above. This gives the desired result of the lemma. ❑

Corollary 1 For any key exchange protocol P,

$$\text{Adv}_P(t, Q) \le q_{\text{test}} \cdot \text{Adv}_P(t, Q')$$

where $Q = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$ and $Q' = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$.

Proof: The proof is straightforward from the proof of Lemma 1 and is omitted. ❑

3.2 The Basis Step

We prove here the induction basis for the proof of Theorem 1. Let SKE_i denote the protocol SKE but with the height of its input tree restricted to some fixed value $i > 0$. Namely, SKE_i is exactly the same as SKE, except that it can be run only for those groups such that $2^i \le n < 2^{i+1}$. Then the following corollary serves as the induction basis: the key exchange protocol SKE_1 is secure against a passive adversary (asking multiple Test queries), under the DDH assumption for $\mathbb{G}$.

Corollary 2 Let $Q = (q_{\text{exec}}, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$. Then we have:

$$\text{Adv}_{\text{SKE}_1}(t, Q) \le 2q_{\text{test}}q_{\text{exec}} \cdot \text{Adv}^{\text{ddh}}_{\mathbb{G}}(t')$$

where $t' = t + O(q_{\text{exec}}t_{\text{SKE1}})$ and t_{SKE_1} is the time required for execution of SKE_1 by any party.

The first step towards proving the corollary has already been taken with the proof of Lemma 1. Recall that by Lemma 1, we showed that the security of a protocol against passive adversaries asking multiple Test queries can be reduced to the security of the same protocol against passive adversaries asking only a single Test query. So to prove Corollary 2, we are left with proving the following lemma which claims that as long as the DDH assumption holds in $\mathbb{G}$, the protocol SKE_1 is secure against passive adversaries who query the Test oracle only once.

Lemma 2 Let $Q = (q_{\text{exec}}, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, 1)$. Then we have

$$\text{Adv}_{\text{SKE}_1}(t, Q) \le 2q_{\text{exec}} \cdot \text{Adv}^{\text{ddh}}_{\mathbb{G}}(t')$$

where t' is as in Corollary 2.

Proof: Let $\mathcal{A}_1$ be a passive adversary attacking protocol SKE_1, with time complexity t and query complexity $Q = (q_{\text{exec}}, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, 1)$. Assume that the probability that $\mathcal{A}_1$ correctly guesses the value of the hidden bit b used by the Test oracle is $1/2 + \varepsilon$. Then we construct from $\mathcal{A}_1$ a distinguisher $\mathcal{D}$ that solves the DDH problem for $\mathbb{G}$ with probability $\varepsilon/q_{\text{exec}}$.

To construct the distinguisher $\mathcal{D}$, we first need to consider the following two distributions:

$$\text{Real}_1 = \left\{ (T, K) \,\middle|\, \begin{array}{l} s_1, t_1 \in_R \mathbb{Z}_q; r_2, \ldots, r_n \in_R \mathbb{Z}_q; \\ w_1 = g^{s_1}; z_2 = g^{r_2}, \ldots, z_n = g^{r_n}; \\ k_1 = g^{s_1 t_1}; x_2 = g^{s_1 r_2}, \ldots, x_n = g^{s_1 r_n}; \\ y_2 = k_1 \cdot x_2^{-1}, \ldots, y_n = k_1 \cdot x_n^{-1}; \\ T = (w_1, z_2, \ldots, z_n, y_2, \ldots, y_n); \\ K = k_1 \end{array} \right\}$$

and

$$\text{Fake}_1 = \left\{ (T, K) \,\middle|\, \begin{array}{l} s_1 \in_R \mathbb{Z}_q; r_2, \ldots, r_n, a_1, \ldots, a_n \in_R \mathbb{Z}_q; \\ w_1 = g^{s_1}; z_2 = g^{r_2}, \ldots, z_n = g^{r_n}; \\ k_1 = g^{a_1}; x_2 = g^{a_2}, \ldots, x_n = g^{a_n}; \\ y_2 = k_1 \cdot x_2^{-1}, \ldots, y_n = k_1 \cdot x_n^{-1}; \\ T = (w_1, z_2, \ldots, z_n, y_2, \ldots, y_n); \\ K = k_1 \end{array} \right\}.$$

The distribution Real_1 with $n \in \{2, 3\}$ represents the distribution of protocol transcript T and session key K in the real execution of SKE_1. (For ease of exposition, we describe the proof for an arbitrary $n \in \{2, 3, \ldots\}$; however, all security results are stated for $n \in \{2, 3\}$.) Notice that we have omitted for brevity user identities and sequence numbers (*i.e.*, $U_i|1$ and $U_1|2$) in the transcript T. The distribution Fake_1 is obtained from Real_1 by changing the way of computing k_1 and x_i's; these values are now computed independently of w_1 and z_i's.

With the above in mind, we now claim that distinguishing between two distributions Real_1 and Fake_1 is at least as difficult as solving the DDH problem for $\mathbb{G}$.

Claim 1 Let $\mathcal{D}'$ be a distinguisher that given as input (T, K) coming from one of two distributions Real_1 and Fake_1, runs in time t and outputs 0 or 1. Then we have:

$$|\Pr[\mathcal{D}'(T,K)=1 \mid (T,K) \leftarrow \text{Real}_1] - \Pr[\mathcal{D}'(T,K)=1 \mid (T,K) \leftarrow \text{Fake}_1]| \leq \text{Adv}_{\mathbb{G}}^{\text{ddh}}(t')$$

where $t' = t + O(t_{\text{exp}})$ and t_{exp} is the time required to perform an exponentiation in $\mathbb{G}$.

Proof: In order to prove the claim, we show how to build from $\mathcal{D}'$ a distinguisher $\mathcal{D}''$ that solves the DDH problem in $\mathbb{G}$. Let $(g^{s_1}, g^{r_2}, g^{s_1'r_2}) \in \mathbb{G}^3$ be an instance of the DDH problem given as input to $\mathcal{D}''$. Using the triple $(g^{s_1}, g^{r_2}, g^{s_1'r_2})$, $\mathcal{D}''$ first generates (T, K) according to the distribution Dist_1 given below. Then $\mathcal{D}''$ runs $\mathcal{D}'(T, K)$ and outputs whatever bit $\mathcal{D}'$ eventually outputs.

The running time of $\mathcal{D}''$ is the running time of $\mathcal{D}'$ added to the time to generate (T, K) according to Dist_1. Note that if $n \in \{2, 3\}$, then generating (T, K) according to Dist_1 requires $\mathcal{D}''$ to perform only a constant number of exponentiations in $\mathbb{G}$.

$$\text{Dist}_1 = \left\{ (T,K) \;\middle|\; \begin{array}{l} t_1 \in_R \mathbb{Z}_q;\ \alpha_3, \beta_3, \ldots, \alpha_n, \beta_n \in_R \mathbb{Z}_q; \\ w_1 = g^{s_1}; \\ z_2 = g^{r_2}, z_3 = g^{t_1\alpha_3 + r_2\beta_3}, \ldots, z_n = g^{t_1\alpha_n + r_2\beta_n}; \\ k_1 = g^{s_1 t_1}; \\ x_2 = g^{s_1'r_2}, x_3 = g^{s_1 t_1 \alpha_3 + s_1' r_2 \beta_3}, \ldots, x_n = g^{s_1 t_1 \alpha_n + s_1' r_2 \beta_n}; \\ y_2 = k_1 \cdot x_2^{-1}, \ldots, y_n = k_1 \cdot x_n^{-1}; \\ T = (w_1, z_2, \ldots, z_n, y_2, \ldots, y_n); \\ K = k_1 \end{array} \right\}$$

If $(g^{s_1}, g^{s_2}, g^{s_1'r_2})$ is a true Diffie-Hellman triple (*i.e.*, $s_1 = s_1'$), then we have $\text{Dist}_1 \equiv \text{Real}_1$ since $k_1 = w_1^{t_1}$ and $x_i = z_i^{s_1}$ for all $i \in [2, n]$. If instead $(g^{s_1}, g^{r_2}, g^{s_1'r_2})$ is a random triple, then it is clear that $\text{Dist}_1 \equiv \text{Fake}_1$. This means that:

(1) The probability that $\mathcal{D}''$ outputs 1 on a true Diffie-Hellman triple is exactly the probability that $\mathcal{D}'$ outputs 1 on (T, K) generated according to the distribution Real_1.
(2) The probability that $\mathcal{D}''$ outputs 1 on a random triple is identical to the probability that $\mathcal{D}'$ outputs 1 on (T, K) generated according to the distribution Fake_1.

So the claim follows. ❑

We now make the following observation about the Fake_1 distribution.

Claim 2 For any (computationally unbounded) adversary $\mathcal{A}$, we have:

$$\Pr[\mathcal{A}(T, K_{(b)}) = b \mid (T, K_{(1)}) \leftarrow \text{Fake}_1; K_{(0)} \leftarrow \mathbb{G}, b \leftarrow \{0, 1\}] = 1/2.$$

Proof: When $\mu = g^{\nu}$, let us write $\log_g \mu$ to denote the exponent ν. Then in distribution Fake_1, the transcript T constrains the exponents a_i only by the following $n - 1$ equations:

$$\begin{aligned}
\log_g y_2 &= a_1 - a_2, \\
\log_g y_2 &= a_1 - a_3, \\
&\vdots \\
\log_g y_n &= a_1 - a_n.
\end{aligned}$$

Since the equation $\log_g K = a_1$ is linearly independent of the set of $n - 1$ equations above, the session key K is independent of the transcript T. This implies the claim. ❑

We are now ready to describe the construction of the distinguisher $\mathcal{D}$. Assume without loss of generality that $\mathcal{A}_1$ makes its Test query to an instance activated by the γth Execute query. The distinguisher $\mathcal{D}$ begins by choosing a random $\delta \in \{1, \ldots, q_{\text{exec}}\}$ as a guess for the value of γ and by choosing a bit b uniformly at random from $\{0, 1\}$. $\mathcal{D}$ then invokes $\mathcal{A}_1$ as a subroutine and proceeds to simulate the oracles. Since $\mathcal{A}_1$ is a passive adversary, $\mathcal{D}$ does not need to simulate the Send oracle. Moreover, $\mathcal{D}$ may ignore Corrupt queries of $\mathcal{A}_1$ because there is no long-term secret information used in the protocol SKE_1. For all other queries of $\mathcal{A}_1$, except the δth Execute query, $\mathcal{D}$ answers them in the natural way by executing protocol SKE_1 on its own. When $\mathcal{A}_1$ asks the δth Execute query, $\mathcal{D}$ slightly deviates from the protocol, embedding an instance of the DDH problem given as input into the transcript as follows: using the input $(g^{s_1}, g^{r_2}, g^{s'_1 s_2}) \in \mathbb{G}^3$, $\mathcal{D}$ generates (T, K) according to the distribution Dist_1 and answers the δth Execute query of $\mathcal{A}_1$ with T. If $\delta \neq \gamma$, then $\mathcal{D}$ aborts and outputs a random bit. Otherwise, $\mathcal{D}$ answers the Test query of $\mathcal{A}_1$ with K if $b = 1$, and with a random key otherwise. Now at some point in time, when $\mathcal{A}_1$ terminates and outputs its guess b', $\mathcal{D}$ outputs 1 if $b = b'$, and 0 otherwise. Let t be the running time of $\mathcal{A}_1$. Then from the simulation above, it is straightforward to see that $\mathcal{D}$ takes at most time $t' = t + O(q_{\text{exec}} q_{\text{SKE}_1})$.

We now analyze the advantage of $\mathcal{D}$ in solving the DDH problem for $\mathbb{G}$. Suppose that $\mathcal{A}_1$ asked its Test query to an instance activated by the δth Execute query; this happens with probability $1/q_{\text{exec}}$. If $(g^{s_1}, g^{r_2}, g^{s'_1 r_2})$ is a true Diffie-Hellman triple, then, by Claim 1, $\text{Dist}_1 \equiv \text{Real}_1$ and thus, by assumption, $\Pr[b = b'] = 1/2 + \varepsilon$. So, the probability that $\mathcal{D}$ outputs 1 on a true Diffie-Hellman triple is also $1/2 + \varepsilon$. If instead $(g^{s_1}, g^{r_2}, g^{s'_1 r_2})$ is a random triple, then $\text{Dist}_1 \equiv \text{Fake}_1$ and hence, $\Pr[b = b'] = 1/2$ by Claim 2. Therefore, the probability that $\mathcal{D}$ outputs 1 on a random triple is exactly 1/2. Now since $\Pr[\delta = \gamma] = 1/q_{\text{exec}}$, we obtain $\text{Adv}^{\text{ddh}}_{\mathbb{G}}(\mathcal{D}) = \varepsilon/q_{\text{exec}}$. Finally, since $\text{Adv}_{\text{SKE}_1}(\mathcal{A}_1) = 2\varepsilon$ and $\text{Adv}^{\text{ddh}}_{\mathbb{G}}(\mathcal{D}) \leq \text{Adv}^{\text{ddh}}_{\mathbb{G}}(t')$

by definitions, it follows that $\mathrm{Adv}_{\mathrm{SKE}_1}(\mathcal{A}_1) \le 2q_{\mathrm{exec}} \cdot \mathrm{Adv}^{\mathrm{ddh}}_{\mathbb{G}}(t')$. This completes the proof of Lemma 2. ❑

3.3 The Induction Step

We now claim that for each $h \ge 1$, if the key exchange protocol SKE_h is secure against passive adversaries, then so is protocol SKE_{h+1}. This claim is formalized by the following corollary.

Corollary 3 Let $Q_{h+1} = (q_{\mathrm{exec}}, 0, q_{\mathrm{reve}}, q_{\mathrm{corr}}, q_{\mathrm{dump}}, q_{\mathrm{test}})$ and $Q_h = (2, 0, 0, 0, 0, 2)$. Let n be the number of users participating in protocol SKE_{h+1} (*i.e.*, $2^{h+1} \le n < 2^{h+2}$). Then we have:

$$\mathrm{Adv}_{\mathrm{SKE}_{h+1}}(t, Q_{h+1}) \le 2q_{\mathrm{test}}q_{\mathrm{exec}} \cdot \mathrm{Adv}_{\mathrm{SKE}_h}(t', Q_h) + q_{\mathrm{test}} \cdot \mathrm{Adv}_{\mathrm{SKE}_1}(t, Q_{h+1})$$

where $t' = t + O(nq_{\mathrm{exec}}t_{\mathrm{SKE}_{h+1}})$ and $t_{\mathrm{SKE}_{h+1}}$ is the time required for execution of SKE_{h+1} by any party.

By Lemma 1, we know that to prove Corollary 3, it suffices to prove the claim that protocol SKE_{h+1} is secure against passive adversaries accessing the Test oracle only once. A precise formulation of this claim is given by the following Lemma 3. The proof of the lemma proceeds very much along the lines of that of Lemma 2, extending the techniques used there to this more interesting case.

Lemma 3 Let $Q_{h+1} = (q_{\mathrm{exec}}, 0, q_{\mathrm{reve}}, q_{\mathrm{corr}}, q_{\mathrm{dump}}, 1)$ and $Q_h = (2, 0, 0, 0, 0, 2)$. Then we have:

$$\mathrm{Adv}_{\mathrm{SKE}_{h+1}}(t, Q_{h+1}) \le 2q_{\mathrm{exec}} \cdot \mathrm{Adv}_{\mathrm{SKE}_h}(t', Q_h) + q_{\mathrm{test}} \cdot \mathrm{Adv}_{\mathrm{SKE}_1}(t, Q_{h+1})$$

where t' is as in Corollary 3.

Proof: Let $\mathcal{A}_{h+1}$ be a passive adversary attacking protocol SKE_{h+1}, with time complexity t and query complexity $Q_{h+1} = (q_{\mathrm{exec}}, 0, q_{\mathrm{reve}}, q_{\mathrm{corr}}, q_{\mathrm{dump}}, 1)$. Given the adversary $\mathcal{A}_{h+1}$, we construct a passive adversary $\mathcal{A}_h$ attacking protocol SKE_h, with time complexity t' and query complexity $Q_h = (2, 0, 0, 0, 0, 2)$.

For ease of exposition, we first introduce some additional notations. Consider the tree structure shown in Fig. 1 of section 2 and recall that $\mathcal{G}_i$ denotes the subgroup consisting of the users in the subtree rooted at the node hosting U_i. In protocol SKE_{h+1}, there are two cases depending on whether $n > 5$ or $n \in \{4, 5\}$. If $n > 5$, then both of two subgroup keys k_2 and k_3 exist, whereas in the case of $n \in \{4, 5\}$, the node N_3 is a leaf node and thus only k_2 exist (*i.e.*, k_3 does not exist). Notice that each of k_2 and k_3 (if it ever exists) is generated by executing protocol SKE_h. Let $T_{h,i}$ denote the transcript of protocol SKE_h executed by subgroup $\mathcal{G}_i$ to generate the subgroup key k_i. Then we write $(T_{h,i}, k_i) \leftarrow \mathrm{Real}_h$ to denote the generation of a transcript/key pair $(T_{h,i}, k_i)$ through a real execution of SKE_h. We also write $(T_{h,i}, a_i) \leftarrow \mathrm{Rand}_h$ to denote the generation of $(T_{h,i}, a_i)$ where $T_{h,i}$ is generated by a real execution of SKE_h and a_i is a random key chosen independently of $T_{h,i}$ but chosen uniformly from $\mathbb{G}$.

Using these notations, we now consider two distributions Real_{h+1} and Rand_{h+1} defined below. The distribution Real_{h+1}, when either $l = m = 3$ (*i.e.*, $n > 5$) or $l = 2$ and $m = 3$ (*i.e.*, $\{4, 5\}$), matches exactly the real execution of protocol SKE_{h+1}. (As in the basis step, we describe the proof for the general case where $2 \le l \le m$ and $m \in \{3, 4, \ldots\}$, but for description purpose only.) The distribution Rand_{h+1} is obtained from Real_{h+1} by replacing each subgroup key k_i with a random key a_i.

We claim that distinguishing between the two distributions Real_{h+1} and Rand_{h+1} is no easier than breaking the security of protocol SKE_h.

$$\text{Real}_{h+1} = \left\{ (T, K) \,|\, \begin{array}{l} (T_{h,2}, k_2), \ldots, (T_{h,l}, k_l) \leftarrow \text{Real}_h; \\ s_1, t_1 \in_R \mathbb{Z}_q; r_2 = I(k_2), \ldots, r_l = I(k_l); \\ r_{l+1}, \ldots, r_m \in_R \mathbb{Z}_q; \\ w_1 = g^{s_1}; z_2 = g^{r_2}, \ldots, z_m = g^{r_m}; \\ k_1 = g^{s_1 t_1}; x_2 = g^{s_1 r_2}, \ldots, x_m = g^{s_1 r_m}; \\ y_2 = k_1 \cdot x_2^{-1}, \ldots, y_m = k_1 \cdot x_m^{-1}; \\ T = (T_{h,2}, \ldots, T_{h,l}, w_1, z_2, \ldots, z_m, y_2, \ldots, y_m); \\ K = k_1 \end{array} \right\}$$

$$\text{Rand}_{h+1} = \left\{ (T, K) \,|\, \begin{array}{l} (T_{h,2}, a_2), \ldots, (T_{h,l}, a_l) \leftarrow \text{Rand}_h; \\ s_1, t_1 \in_R \mathbb{Z}_q; r_2 = I(a_2), \ldots, r_l = I(a_l); \\ r_{l+1}, \ldots, r_m \in_R \mathbb{Z}_q; \\ w_1 = g^{s_1}; z_2 = g^{r_2}, \ldots, z_m = g^{r_m}; \\ k_1 = g^{s_1 t_1}; x_2 = g^{s_1 r_2}, \ldots, x_m = g^{s_1 r_m}; \\ y_2 = k_1 \cdot x_2^{-1}, \ldots, y_m = k_1 \cdot x_m^{-1}; \\ T = (T_{h,2}, \ldots, T_{h,l}, w_1, z_2, \ldots, z_m, y_2, \ldots, y_m); \\ K = k_1 \end{array} \right\}$$

Claim 3 Let $\mathcal{D}$ be a distinguisher that given as input (T, K) coming from one of two distributions Real_{h+1} and Rand_{h+1}, runs in time t and outputs 0 or 1. Let $Q_h = (2, 0, 0, 0, 0, 2)$. Then we have:

$$\begin{aligned} &|\Pr[\mathcal{D}(T, K) = 1 \,|\, (T, K) \leftarrow \text{Real}_{h+1}] - \Pr[\mathcal{D}(T, K) = 1 \,|\, (T, K) \leftarrow \text{Rand}_{h+1}]| \\ &\le \text{Adv}_{\text{SKE}_h}(t', Q_h) \end{aligned}$$

where $t' = t + O(t_{\text{exp}})$ and t_{exp} is the time required to perform an exponentiation in $\mathbb{G}$.

Proof: Suppose that μ and ν are the probabilities that $\mathcal{D}$ outputs 1 on (T, K) generated according to Real_{h+1} and Rand_{h+1}, respectively. Then we prove the claim by constructing from $\mathcal{D}$ an adversary $\mathcal{A}_h'$ attacking protocol SKE_h with advantage $|\mu - \nu|$.

First, $\mathcal{A}_h'$ obtains $l - 1$ transcripts $T_{h,2}$, $T_{h,3}$, …, $T_{h,l}$ by making an Execute query for each of the subgroups $\mathcal{G}_2$, $\mathcal{G}_3$, …, $\mathcal{G}_l$. Let $\prod_{U \in \mathcal{G}_i}$ denote any instance activated by the Exe-

cute query directed to $\mathcal{G}_i$. Next, $\mathcal{A}_h'$ asks $l-1$ Test queries Test($\prod_{U\in\mathcal{G}_2}$), Test(($\prod_{U\in\mathcal{G}_3}$), …, Test($\prod_{U\in\mathcal{G}_l}$). Let k_i' be either the real session key or a random session key returned in response to the query Test($\prod_{U\in\mathcal{G}_i}$). We then write $(T_{h,i}, k_i') \leftarrow \text{Test}_h$ to denote the above way of generating a transcript/key pair $(T_{h,i}, k_i')$.

Having made the queries and received the results as above, $\mathcal{A}_h'$ generates (T, K) according to the distribution Dist_{h+1} (defined below), runs $\mathcal{D}(T, K)$, and outputs whatever bit $\mathcal{D}$ outputs. Distribution Dist_{h+1} is defined as follows,

$$\text{Real}_{h+1} = \left\{ (T, K) \,|\, \begin{array}{l} (T_{h,2}, k_2'), \ldots, (T_{h,l}, k_l') \leftarrow \text{Test}_h; \\ s_1, t_1 \in_R \mathbb{Z}_q; \\ r_2 = I(k_2'), \ldots, r_l = I(k_l'); r_{l+1}, \ldots, r_m \in_R \mathbb{Z}_q; \\ w_1 = g^{s_1}; z_2 = g^{r_2}, \ldots, z_m = g^{r_m}; \\ k_1 = g^{s_1 t_1}; x_2 = g^{s_1 r_2}, \ldots, x_m = g^{s_1 r_m}; \\ y_2 = k_1 \cdot x_2^{-1}, \ldots, y_m = k_1 \cdot x_m^{-1}; \\ T = (T_{h,2}, \ldots, T_{h,l}, w_1, z_2, \ldots, z_m, y_2, \ldots, y_m); \\ K = k_1 \end{array} \right\}.$$

To generate (T, K) according to Dist_{h+1}, $\mathcal{A}_h'$ performs $O(m)$ exponentiations in $\mathbb{G}$ and makes $l-1$ Execute queries and $l-1$ Test queries. If we instantiate both l and m with 3, $\mathcal{A}_h'$ has time complexity $t' = t + O(t_{\text{exp}})$ and query complexity $Q_h = (2, 0, 0, 0, 0, 2)$.

The only possible difference between the distribution Dist_{h+1} and the other two distributions Real_{h+1} and Rand_{h+1} is in the way of generating the subgroup keys. If each k_i' is the real session key, clearly we have $\text{Dist}_{h+1} \equiv \text{Real}_{h+1}$. On the other hand, if each k_i' is a random session key chosen independently of the transcript $T_{h,i}$, then $\text{Dist}_{h+1} \equiv \text{Real}_{h+1}$. This means that:

(1) The probability that $\mathcal{A}_h'$ outputs 1 when k_2', …, k_l' are real session keys is exactly μ, the probability that $\mathcal{D}$ outputs 1 on (T, K) generated according to the distribution Real_{h+1}.
(2) The probability that $\mathcal{A}_h'$ outputs 1 when k_2', …, k_l' are random session keys is exactly ν, the probability that $\mathcal{D}$ outputs 1 on (T, K) generated according to the distribution Real_{h+1}.

Thus, $\text{Adv}_{\text{SKE}^h}(\mathcal{A}_h') = |\mu - \nu|$. Since $\text{Adv}_{\text{SKE}^h}(\mathcal{A}_h') \le \text{Adv}_{\text{SKE}^h}(t', Q_h)$, we obtain the statement of Claim 3. ❑

Letting Real_1 be as defined in the proof of Lemma 2, we continue with the following claim.

Claim 4 For any (computationally unbounded) adversary $\mathcal{A}$, we have:

$$\Pr[\mathcal{A}(T, K_{(b)}) = b \,|\, (T, K_{(1)}) \leftarrow \text{Rand}_{h+1}; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0, 1\}] =$$
$$\Pr[\mathcal{A}(T, K_{(b)}) = b \,|\, (T, K_{(1)}) \leftarrow \text{Real}_1; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0, 1\}].$$

Proof: In distribution Rand_{h+1}, the session key K is completely independent of the set of l

– 1 transcripts $\{T_{h,i} \mid i \in [2, l]\}$ because each $a_i \in \mathbb{G}$ is chosen at random independently of $T_{h,i}$. Therefore, if we define Rand'_{h+1} as the distribution derived from Rand_{h+1} by eliminateing all the transcripts $T_{h,2}, T_{h,3}, \ldots, T_{h,l}$, it is clear that:

$$\begin{aligned}&\Pr[\mathcal{A}(T, K_{(b)}) = b \mid (T, K_{(1)}) \leftarrow \text{Rand}'_{h+1}; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0,1\}] = \\ &\Pr[\mathcal{A}(T, K_{(b)}) = b \mid (T, K_{(1)}) \leftarrow \text{Rand}_{h+1}; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0,1\}].\end{aligned} \tag{2}$$

Because now $\text{Rand}'_{h+1} \equiv \text{Real}_1$, it is also immediate that:

$$\begin{aligned}&\Pr[\mathcal{A}(T, K_{(b)}) = b \mid (T, K_{(1)}) \leftarrow \text{Rand}'_{h+1}; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0,1\}] = \\ &\Pr[\mathcal{A}(T, K_{(b)}) = b \mid (T, K_{(1)}) \leftarrow \text{Real}_1; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0,1\}].\end{aligned} \tag{3}$$

Combining Eqs. (2) and (3) yields the result of Claim 4. ❑

Before continuing further, let us define

$$\text{SuccPr}_1(\mathcal{A}_{h+1}) = \Pr[\mathcal{A}_{h+1}(T, K_{(b)}) = b \mid (T, K_{(1)}) \leftarrow \text{Real}_1; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0, 1\}]$$

and

$$\text{SuccPr}_{h+1}(\mathcal{A}_{h+1}) = \Pr[\mathcal{A}_{h+1}(T, K_{(b)}) = b \mid (T, K_{(1)}) \leftarrow \text{Real}_{h+1}; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0, 1\}].$$

Armed with Claims 3 and 4, we now give the details of the construction of the adversary $\mathcal{A}_h$. Assume without loss of generality that $\mathcal{A}_{h+1}$ makes its Test query to an instance activated by the γth Execute query. The adversary $\mathcal{A}_h$ begins by choosing a random $\delta \in \{1, \ldots, q_{\text{exec}}\}$ as a guess for the value of γ and by choosing a bit b uniformly at random from $\{0, 1\}$. It then runs $\mathcal{A}_{h+1}$ as a subroutine, answering the oracle queries of $\mathcal{A}_{h+1}$. For all queries of $\mathcal{A}_{h+1}$, except the δth Execute query, $\mathcal{A}_h$ answers them in the natural way by executing protocol SKE_{h+1} on its own. But when $\mathcal{A}_{h+1}$ asks the δth Execute query, $\mathcal{A}_h$ responds by calling its own Execute and Test oracles; namely, it generates (T, K) according to the distribution Dist_{h+1} and returns the transcript T in response to the query. If $\delta \neq \gamma$, $\mathcal{A}_h$ aborts and outputs a random bit. Otherwise, $\mathcal{A}_h$ answers the Test query of $\mathcal{A}_{h+1}$ with K if $b = 1$, and with a random key otherwise. Now when $\mathcal{A}_{h+1}$ terminates and outputs its guess b', $\mathcal{A}_h$ outputs 1 if $b = b'$, and 0 otherwise.

It is easy to see that $\mathcal{A}_h$ has query complexity $Q_h = (2, 0, 0, 0, 0, 2)$ and time complexity $t' = t + O(nq_{\text{exec}}t_{\text{SKE}_{h+1}})$, where $2^{h+1} \leq n \leq 2^{h+2}$.

To analyze the advantage of $\mathcal{A}_h$ in attacking SKE_h, assume that $\mathcal{A}_{h+1}$ asked its Test query to an instance activated by the δth Execute query. If $k_2', \ldots, k_l'$ in Dist_{h+1} are real session keys, then $\text{Dist}_{h+1} \equiv \text{Real}_{h+1}$ and thus, the probability that $\mathcal{A}_{h+1}$ correctly guesses the hidden bit b is $\text{SuccPr}_{h+1}(\mathcal{A}_{h+1})$. Hence, the probability that $\mathcal{A}_h$ outputs 1 when its Test oracle returns actual session keys is also $\text{SuccPr}_{h+1}(\mathcal{A}_{h+1})$. On the other hand, if $k_2', \ldots, k_l'$ in Dist_{h+1} are random session keys, then $\text{Dist}_{h+1} \equiv \text{Rand}_{h+1}$ and thus, by Claim 4, the probability that $\mathcal{A}_{h+1}$ correctly guesses the hidden bit b is $\text{SuccPr}_1(\mathcal{A}_{h+1})$. So, the probability that $\mathcal{A}_h$ outputs 1 when its Test oracle returns random session keys is also $\text{SuccPr}_1(\mathcal{A}_{h+1})$. Therefore since $\Pr[\delta = \gamma] = 1/q_{\text{exec}}$, we obtain:

$$\mathrm{Adv}_{\mathrm{SKE}_h}(\mathcal{A}_h) = \frac{1}{q_{\mathrm{exec}}} |\mathrm{SuccPr}_{h+1}(\mathcal{A}_{h+1}) - \mathrm{SuccPr}_1(\mathcal{A}_{h+1})|. \quad (4)$$

Note that this equation already implies that $|\mathrm{SuccPr}_{h+1}(\mathcal{A}_{h+1}) - \mathrm{SuccPr}_1(\mathcal{A}_{h+1})|$ is negligible and so $\mathrm{SuccPr}_{h+1}(\mathcal{A}_{h+1})$ is not much greater than 1/2.

It remains to bound the advantage of $\mathcal{A}_{h+1}$ in attacking protocol SKE_{h+1}. By applying Eq. (4) to the definitional equation $\mathrm{Adv}_{\mathrm{SKE}_{h+1}}(\mathcal{A}_{h+1}) = |2 \cdot \mathrm{SuccPr}_{h+1}(\mathcal{A}_{h+1}) - 1|$, we easily have:

$$\mathrm{Adv}_{\mathrm{SKE}_{h+1}}(\mathcal{A}_{h+1}) \le |2q_{\mathrm{exec}} \cdot \mathrm{Adv}_{\mathrm{SKE}_h}(\mathcal{A}_h) + 2 \cdot \mathrm{SuccPr}_1(\mathcal{A}_{h+1}) - 1|.$$

From this, the following is immediate:

$$\begin{aligned} \mathrm{Adv}_{\mathrm{SKE}_{h+1}}(\mathcal{A}_{h+1}) &\le 2q_{\mathrm{exec}} \cdot \mathrm{Adv}_{\mathrm{SKE}_h}(\mathcal{A}_h) + \mathrm{Adv}_{\mathrm{SKE}_1}(\mathcal{A}_{h+1}) \\ &\le 2q_{\mathrm{exec}} \cdot \mathrm{Adv}_{\mathrm{SKE}_h}(t', Q_h) + \mathrm{Adv}_{\mathrm{SKE}_1}(t, Q_{h+1}). \end{aligned}$$

This completes the proof of Lemma 3. ❑

4. A SCALABLE AUTHENTICTED GKE PROTOCOL

Perhaps one of the most pleasing results of research on group key exchange is the one-round compiler presented by Katz and Yung [5] (in short, the KY compiler). The KY compiler shows how we can transform any group key exchange protocol secure against a passive adversary into one that is secure against an active adversary. It certainly is elegant in its scalability, usefulness, and proven security. The transformation itself is quite simple: it first adds an additional round for exchanging nonces among users and then lets all the messages of the original protocol be signed and verified with the nonces. In this section, we convert the unauthenticated protocol SKE into the authenticated protocol SKE^+ by using a modified version of the KY compiler.

4.1 Description of SKE^+

Let again $\mathcal{G}$ be the set of users wishing to establish a common session key. During the initialization phase of SKE^+, each user $U_i \in \mathcal{G}$ generates its long-term verification/signing keys (PK_i, SK_i) by running $\mathrm{Kgen}(1^\kappa)$ (see section 2 of [19] for the definition of signature schemes) and makes the verification key PK_i public. Recall that each user U_i receives as input a pair of session and group IDs (sid_i, gid_i) to start to run the protocol. Upon receiving (sid_i, gid_i), U_i verifies that (1) sid_i is currently not in use for some active instance of it and (2) there is a bijective mapping between users in gid_i and nodes of the tree to be used. By checking the first condition, U_i is ensuring that the session ID is unique for all its active instances. This means that as far as security is concerned, reusing a session ID previously assigned to a closed session is legal and thus session IDs can be erased once their corresponding sessions have ended. If either of both conditions above is untrue, then U_i declines to participate in the protocol run associated with (sid_i, gid_i). Otherwise, U_i performs the protocol SKE^+ as follows:

Round 1: Each user $U_i \in \mathcal{G}$ chooses a random nonce $\phi_i \in \{g^0, g^1, \ldots, g^{q-1}\}$ and sends $\tilde{M}_i^0 = U_i \,|\, 0 \,|\, \phi_i$ to its parent, sibling, descendants, and sibling's descendants. Let ncs_i be an ordered sequence defined as follows,

$$\text{ncs}_i = \begin{cases} ((U_i, \phi_i), (U_{2i}, \phi_{2i}), (U_{2i+1}, \phi_{2i+1})) & \text{if } U_i \text{ has two children} \\ ((U_i, \phi_i), (U_{2i}, \phi_{2i})) & \text{if } U_i \text{ has only the left child} \\ ((U_i, \phi_i)) & \text{otherwise} \end{cases}.$$

Let $\varphi_i = \lfloor i/2 \rfloor$. Then, after receiving all nonces (from its children, sibling, ancestors, and ancestors' siblings), each U_i computes the ordered sequences ncs_i, $\text{ncs}_{\varphi(i)}$, $\text{ncs}_{\varphi(\varphi(i))}$, …, ncs_1 as defined above. Notice that the maximum number of nonces received by any single user is at most $2\lfloor \log n \rfloor$.

Round 2: This round proceeds like the first round of protocol SKE, except that users have to sign their outgoing messages:

- Each user $U_i \neq U_1$ computes z_i as specified in SKE and generates a signature $\sigma_i^1 = \text{Sign}_{SK_i}(U_i \,|\, 1 \,|\, z_i \,|\, \text{sid}_i \,|\, \text{ncs}_{\varphi(i)} \,|\, \text{ncs}_{\varphi(\varphi(i))} \,|\, \ldots \,|\, \text{ncs}_1)$. Then U_i sends $\tilde{M}_i^1 = U_i \,|\, 1 \,|\, z_i \,|\, \sigma_i^1$ to its parent.
- The operation of U_1 is exactly the same as in SKE. That is, U_1 chooses two random $s_1, t_1 \in \mathbb{Z}_q$ and computes $k_1 = g^{s_1 t_1}$.

Round 3: All users operate as in Round 2 of SKE, but verifying the correctness of incoming messages and signing outgoing messages. We describe this round only for users who have both left and right children; users with left child only behave correspondingly.

- When U_i receives $\tilde{M}_j^1 = U_j \,|\, 1 \,|\, z_j \,|\, \sigma_j^1$ from U_j for $j = 2i$ and $j = 2i + 1$, it first checks that $\text{Vrfy}_{PK_j}(U_j \,|\, 1 \,|\, z_j \,|\, \text{sid}_i \,|\, \text{ncs}_i \,|\, \text{ncs}_{\varphi(i)} \,|\, \ldots \,|\, \text{ncs}_1, \sigma_j^1) = 1$. If any of the verifications fail, U_i aborts the protocol without accepting (*i.e.*, without computing a session key). Otherwise, U_i computes x_{2i}, y_{2i}, x_{2i+1}, y_{2i+1}, and w_i as in protocol SKE, generates a signature $\sigma_i^2 = \text{Sign}_{SK_i}(U_i \,|\, 2 \,|\, w_i \,|\, y_{2i} \,|\, y_{2i+1} \,|\, \text{sid}_i \,|\, \text{ncs}_i \,|\, \text{ncs}_{\varphi(i)} \,|\, \ldots \,|\, \text{ncs}_1)$, and sends $\tilde{M}_i^1 = U_i \,|\, 2 \,|\, w_i \,|\, y_{2i} \,|\, y_{2i+1} \,|\, \sigma_i^2$ to its descendants.

Key computation: Each user $U_i \neq U_1$, for all messages $\tilde{M}_j^2$ from its ancestors in the tree, checks that $\text{Vrfy}_{PK_j}(U_j \,|\, 2 \,|\, w_j \,|\, y_{2j} \,|\, y_{2j+1} \,|\, \text{sid}_i \,|\, \text{ncs}_j \,|\, \text{ncs}_{\varphi(j)} \,|\, \ldots \,|\, \text{ncs}_1, \sigma_j^2) = 1$. If any of the verifications fail, U_i terminates without accepting. Otherwise, U_i derives the root node key k_1 as in SKE and sets the session key K equal to k_1.

The above transformation from SKE to SKE$^+$ requires round complexity to be increased by a constant factor and the other (bit, message, and computation) complexities by a factor of $\log n$. The latter part of these increases is because the users at (or close to) leaves additionally have to receive about $2\log n$ nonces and to perform about $\log n$ signature verifications. Consequently, the asymptotic bounds for the complexities remain unchanged between SKE and SKE$^+$. This unchanged scalability well explains why the KY compiler could not be directly applicable to SKE: as soon as we invoke the KY compiler on an arbitrary GKE protocol, the message and bit complexities of the resulting protocol rise at least up to $O(n)$ because the compiler requires each user to receive nonces from all

other users.

The primary idea behind the KY compiler is to use the set of n nonces shared by users as a unique session identifier for all time points. Based on this idea, the KY compiler mandates the users to always include their nonce set in signing and verifying protocol messages. In this way, not only the freshness of any exchanged message is guaranteed but also no message can be relayed between user instances holding different sets of nonces. It is this observation that the KY compiler exploits to achieve provable security of the compiled protocol.

Our transformation, though similar in spirit as that by the KY compiler, reduces the number of nonces to be received per user to the order of log$n$ while achieving provable security of the protocol SKE$^+$. The two main observations underlying this result are that: (1) at a given point of time, each pre-defined session ID is unique for all concurrent runs of SKE$^+$ (see section 2 of [19] for the justification of pre-defined session IDs) and (2) even with at most $O(\log n)$ nonces per user, SKE$^+$ is able to guarantee the freshness of the messages exchanged among users. The first observation is clear from the description of SKE$^+$; no two active instances of a user possess a same session ID. The second observation becomes quite obvious once we notice that there is a chain of nonces in SKE$^+$: for all $2 \le i \le n$, two ordered sequences ncs_i and $\text{ncs}_{\varphi(i)}$ are linked by the common element ϕ_i. This chain of nonces enables each user U_i to verify the freshness of all messages from its ancestors, even when those messages are not signed with ϕ_i. In other words, the use of the nonce chain ensures that no singed message is replayed between two sessions even with a same session ID. We call this technique *nonce-chained authentication*. These two observations, taken together, suggest that a pre-defined session ID combined with the nonce-chained authentication technique serves as a unique session identifier for all time points and thereby obviates the need for each user to receive n nonces.

4.2 Security Result for SKE$^+$

Here we claim that the group key exchange protocol SKE$^+$ is secure against active adversaries under the security of protocol SKE against passive adversaries. The following theorem makes this claim precise.

Theorem 2 Let $Q = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$ and $Q' = (q_{\text{exec}} + q_{\text{send}}/2, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}} + q_{\text{send}}/2, q_{\text{test}})$. For any adversary with time complexity at most t and query complexity at most Q, its advantage in breaking the security of protocol SKE$^+$ is upper bounded by:

$$\text{Adv}_{\text{SKE}^+}(t, Q) \le \text{Adv}_{\text{SKE}}(t', Q') + |\mathcal{U}| \cdot \text{Succ}_{\Sigma}(t') + \frac{q_{\text{send}}^2 + q_{\text{exec}} q_{\text{send}}}{|\mathbb{G}|}$$

where $t' = t + O(|\mathcal{U}| q_{\text{exec}} t_{\text{SKE}} + q_{\text{send}} t_{\text{SKE}^+})$ and t_{SKE^+} is the time required for execution of SKE$^+$ by any party.

We prove Theorem 2 in section 5 by finding a reduction from the security of protocol SKE$^+$ to the security of protocol SKE. Assuming an active adversary $\mathcal{A}^+$ who attacks protocol SKE$^+$, we construct a passive adversary $\mathcal{A}^+$ that uses $\mathcal{A}^+$ in its attack on SKE. As in a typical reductionist approach, the adversary $\mathcal{A}$ simply runs $\mathcal{A}^+$ as a subroutine and answers the oracle queries of $\mathcal{A}^+$ on its own. The idea in constructing $\mathcal{A}$ is to use the fact

that in attacking SKE$^+$, the adversary $\mathcal{A}^+$ is able to relay messages only between user instances with the same session ID and the matching nonce sequences. Based on this idea, the adversary $\mathcal{A}$ obtains a transcript T of SKE for each unique combination of session ID and nonce sequences by calling its own Execute oracle, and generates a transcript T^+ of SKE$^+$ by patching T with appropriate signatures. $\mathcal{A}$ then use the messages of T^+ in answering $\mathcal{A}^+$'s Send queries directed to user instances which have the same session ID and nonce sequences as used in generating T^+. In this way, $\mathcal{A}^+$ is limited to sending messages already contained in T^+, unless signature forgery and nonce repetition occur. In essence, $\mathcal{A}$ is ensuring that $\mathcal{A}^+$'s capability of attacking protocol SKE$^+$ is demonstrated only on the session key associated with the patched transcript T^+ and thus is translated directly into the capability of attacking protocol SKE.

However, there exists a difficulty in constructing the passive adversary $\mathcal{A}$ from the active adversary $\mathcal{A}^+$. Since $\mathcal{A}^+$ 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 $\mathcal{A}$ 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 KY compiler. The proof for the KY compiler circumvents this simulation problem by letting $\mathcal{A}$ guess the session in which $\mathcal{A}^+$ will take advantage of its only chance to access the Test oracle. For the guessed session, $\mathcal{A}$ handles the queries of the active adversary by calling its own Execute oracle as described above, and for all other sessions, $\mathcal{A}$ honestly responds by directly executing protocol SKE$^+$ (*i.e.*, without accessing the Execute oracle). But while this approach works well against adversaries who are restricted to ask only a single Test query, it is not the case in our adversarial model where adversaries are allowed to query the Test oracle as many times as they want. Simply put, the probability of correctly guessing all the sessions to be tested is negligibly low.

One possible way out of this seemingly dead end situation is to use the result of Corollary 1 (given in section 3), which states that in attacking a key exchange protocol, the advantage of any adversary asking q_{test} Test queries is at most q_{test} times the maximum advantage obtainable by an adversary who asks only one Test query. Given Corollary 1, the security of protocol SKE$^+$ can be proved by taking the same betting approach as used in the proof for the KY compiler. However, this solution necessarily incurs a loss of non-constant factor (*i.e.*, $q_{\text{test}}(q_{\text{exec}} + q_{\text{send}})$) in the reduction. Fortunately, we have a better solution which uses Dump queries. Since the Dump oracle returns all short-term secrets used by an instance, the passive adversary $\mathcal{A}$ can perfectly simulate actions of the instance even when the active adversary $\mathcal{A}^+$ sends arbitrary messages that are not contained in the patched transcript T^+. $\mathcal{A}$ therefore now can use a patched transcript for all sessions without recourse to Corollary 1 and without the need for betting on one session. One immediate result of this is that there is no loss in our reduction from the security of protocol SKE$^+$ to the security of protocol SKE.

5. SECURITY PROOF FOR SKE$^+$

Let $\mathcal{A}^+$ be an active adversary attacking the authenticated protocol SKE$^+$, with time complexity t and query complexity $Q = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$. Then we con-

struct from $\mathcal{A}^+$ a passive adversary $\mathcal{A}$ attacking the unauthenticated protocol SKE, with time complexity t' and query complexity $Q' = (q_{\text{exec}} + q_{\text{send}}/2, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}} + q_{\text{send}}/2, q_{\text{test}})$. As indicated by the statement of the theorem, the advantage of $\mathcal{A}$ in breaking the security of SKE is equal to the advantage that $\mathcal{A}^+$ has in breaking the security of SKE$^+$, provided that none of the following two events occur during $\mathcal{A}^+$'s attack on protocol SKE$^+$.

- Repeat: The event that a same nonce is used by a user for two different instances, one activated by a Send query and the other activated by either an Execute or a Send query.
- Forge: The event that $\mathcal{A}^+$ outputs a valid forgery with respect to the public key PK_j of some user $U_j \in \mathcal{U}$ before making the query Corrupt(U_j).

We begin by bounding the probabilities of these events occurring. First, by a straightforward calculation, we immediately obtain the following inequality:

$$\Pr[\text{Repeat}] \leq \frac{q_{\text{send}}^2 + q_{\text{exec}} q_{\text{send}}}{|\mathbb{G}|}. \tag{5}$$

Next, the probability that the event Forge occurs is bounded by the following lemma.

Lemma 4 Pr[Forge] = $|\mathcal{U}| \cdot \text{Succ}_{\Sigma}(t')$, where t' is as in Theorem 2.

Proof: Assuming that the event Forge occurs, we construct from $\mathcal{A}^+$ an algorithm $\mathcal{F}$ who succeeds (with a non-negligible probability) in outputting an existential forgery against the signature scheme Σ. The algorithm $\mathcal{F}$ is given as input a public verification key PK and access to a signing oracle associated with PK. The goal of $\mathcal{F}$ is then to produce a message/signature pair (M, σ) such that $\text{Vrfy}_{PK}(M, \sigma) = 1$ and σ was not previously output by the signing oracle as a signature on message M.

$\mathcal{F}$ begins by choosing at random a user $U_f \in \mathcal{U}$ and setting PK_f to PK. For all other users in $\mathcal{U}$, $\mathcal{F}$ honestly generates a verification/signing key pair by running the key generation algorithm Kgen(1^κ). $\mathcal{F}$ then runs adversary $\mathcal{A}^+$ as a subroutine, simulating the oracles. $\mathcal{F}$ answers all the queries from $\mathcal{A}^+$ in the natural way by executing protocol SKE$^+$ on its own, except when $\mathcal{A}^+$ asks Send and Corrupt queries. In this latter case, $\mathcal{F}$ proceeds as follows:

- Send($\prod_i^\pi$, M): If $U_i = U_f$, $\mathcal{F}$ answers the query by accessing the signing oracle associated with PK. Otherwise, $\mathcal{F}$ answers exactly as specified in the protocol.
- Corrupt(U_i): If $U_i \neq U_f$, then $\mathcal{F}$ simply hands the long-term singing key of U_i which were generated by $\mathcal{F}$ itself. Otherwise, $\mathcal{F}$ halts and outputs "fail".

The simulation provided above is perfect unless the event that adversary $\mathcal{A}^+$ makes the query Corrupt(U_f) occurs. Notice that the event of $\mathcal{A}^+$ asking Corrupt(U_f) should not happen if Forge occurs against U_f.

Throughout the simulation, $\mathcal{F}$ monitors each Send query from $\mathcal{A}^+$, and checks if it includes a message/signature pair (M, σ) such that $\text{Vrfy}_{PK}(M, \sigma) = 1$ and σ was not previously output by any instance of U_f as a signature on M. If no such query is made until $\mathcal{A}^+$ stops, then $\mathcal{F}$ halts and outputs "fail". Otherwise, $\mathcal{F}$ outputs (M, σ) as a valid forgery with

respect to PK. Lemma 4 directly follows by noticing that this latter case occurs with probability Pr[Forge]/$|\mathcal{U}|$. ❑

Having bounded the probabilities that events Repeat and Forge occur, we now describe the construction of the passive adversary $\mathcal{A}$ attacking protocol SKE. After running Kgen(1^κ) to generate a verification/signing key pair (PK_i, SK_i) for each $U_i \in \mathcal{U}$, $\mathcal{A}$ invokes $\mathcal{A}^+$ as a subroutine and answers the oracle queries of $\mathcal{A}^+$ on its own. If Repeat or Forge ever occurs, $\mathcal{A}$ aborts and outputs a random bit. Otherwise, $\mathcal{A}$ outputs whatever bit $\mathcal{A}^+$ eventually outputs. The queries of $\mathcal{A}^+$ are answered as follows:

Execute queries Upon receiving the query Execute(sid, gid), $\mathcal{A}$ sends the same query to its own Execute oracle and receives in return a transcript T of an execution of protocol SKE. Given T, $\mathcal{A}$ generates a transcript T^+ of an execution of protocol SKE$^+$ by

(1) choosing a random $\phi_i \in \mathbb{G}$ for all $U_i \in$ gid,
(2) computing ncs_i for all $U_i \in$ gid,
(3) signing the messages in T as specified in SKE$^+$,
(4) and prepending $\{U_i \,|\, 0 \,|\, \phi_i\}_{U_i \in \text{gid}}$ to the signed transcript.

Then $\mathcal{A}$ returns the patched transcrip T^+ in response to the Execute query of $\mathcal{A}^+$ and adds the pair (sid, gid, ncs_1, T) into a list NTList which is maintained by $\mathcal{A}$ to link a simulated execution of SKE$^+$ to an execution of SKE.

Send queries If $\mathcal{A}^+$ makes a query of the form Send($\prod_i^\pi$, sid | gid), $\mathcal{A}$ sets sid_i^π = sid and gid_i^π = gid, chooses a random $\phi_i^\pi \in \mathbb{G}$, and returns $U_i \,|\, 0 \,|\, \phi_i^\pi$ to $\mathcal{A}^+$. If the query is of the form Send($\prod_i^\pi$, $U_i \,|\, 0 \,|\, \phi_j$) for some $U_j \in \text{gid}_i^\pi$, then there are the following two cases:

- If ϕ_j is not the last nonce that $\prod_i^\pi$ is expected to receive, $\mathcal{A}$ simply waits for the next nonce.
- Otherwise, $\mathcal{A}$ computes ncs_i, $\text{ncs}_{\varphi(i)}$, $\text{ncs}_{\varphi(\varphi(i))}$, …, ncs_1 as defined in the protocol and checks that the value of ncs_1 has ever been used as the value of ncs_1 for any other instance of user U_i. If this is true, then $\mathcal{A}$ simulates the actions of $\prod_i^\pi$ by executing protocol SKE$^+$ honestly. Notice in this case that $\prod_i^\pi$ will never accept because the replayed ncs_1 will be detected by the nonce-chained authentication technique of SKE$^+$. (Thus, in the rest of the proof simulation, we will exclude this case from consideration.) Otherwise, $\mathcal{A}$ replies to $\mathcal{A}^+$ by following the general instruction (given below) for answering Send queries of $\mathcal{A}^+$.

> The general instruction for answering query Send($\prod_i^\pi$, M):
> $\mathcal{A}$ checks the list NTList to see if there exists an entry of the form (sid_i^π, gid_i^π, $\text{ncs}_{1,i}^\pi$, T), where $\text{ncs}_{1,i}^\pi$ denotes ncs_1 held by $\prod_i^\pi$. If so, then $\mathcal{A}$ generates the message $\tilde{M}_i^d = U_i \,|\, d \,|\, * \,|\, \sigma_i^d$ from the appropriate message $\tilde{M}_i^d = U_i \,|\, d \,|\, *$ in T and returns it to adversary $\mathcal{A}^+$. Otherwise, $\mathcal{A}$ first obtains a transcript T of an execution of SKE by making the query Execute(sid_i^π, gid_i^π) to its own oracle, then proceeds as in the former case, and finally adds the tuple (sid_i^π, gid_i^π, $\text{ncs}_{1,i}^\pi$, T) to the list NTList for future use.

For all other Send queries (that is, the queries of the form Send($\prod_i^\pi$, $U_i|d|*|\sigma_j^d$) that requires some response message, $\mathcal{A}$ first verifies the correctness of the incoming message. If the verification fails, the instance is terminated immediately without accepting. Otherwise, $\mathcal{A}$ proceeds as follows:

- If no one in gid_i^π has been asked a Corrupt query before the Send query, $\mathcal{A}$ responds by following the general instruction above.
- On the contrary, suppose that some user in gid_i^π has been asked a Corrupt query before the Send query. In this case, $\mathcal{A}$ may have to simulate the action of $\prod_i^\pi$ without recourse to a fixed transcript, because the adversary $\mathcal{A}^+$ may have signed and sent an arbitrary message of its choice. Fortunately, $\mathcal{A}$ can do this by asking a query to its own Dump oracle. First, $\mathcal{A}$ finds an entry (sid_i^π, gid_i^π, $\text{ncs}_{1,i}^\pi$, T) in NTList and makes a Dump query to the U_i' s instance activated by the Execute query that resulted in the transcript T. Now, with the short-term secrets returned in response to the Dump query, $\mathcal{A}$ should be able to simulate the action of $\prod_i^\pi$ perfectly, following the protocol exactly.

Dump queries Upon receiving the query Dump($\prod_i^\pi$), $\mathcal{A}$ first finds an entry (sid_i^π, gid_i^π, $\text{ncs}_{1,i}^\pi$, T) in list NTList. $\mathcal{A}$ then makes a Dump query to the U_i's instance activated by the Execute query that resulted in the transcript T, and simply forwards the output of this Dump query to $\mathcal{A}$.

Corrupt queries These queries are answered in the obvious way. Namely, $\mathcal{A}$ responds to the query Corrupt(U_i) by returning the long-term signing key SK_i.

Reveal queries As can be seen from the way $\mathcal{A}$ handles Execute and Send queries of $\mathcal{A}^+$, no session keys are available to $\mathcal{A}$. However, the query Reveal($\prod_i^\pi$) can still be answered as follows:

- If either the query Dump($\prod_i^\pi$) was asked or some user in gid_i^π was corrupted before gid_i^π received its last incoming message, then $\mathcal{A}$ already has all the information needed to compute the appropriate session key and can therefore answer the query.
- Otherwise, $\mathcal{A}$ proceeds according to the general instruction (given below) for answering Reveal and Test queries of $\mathcal{A}$.

Test queries $\mathcal{A}$ answers all of $\mathcal{A}^+$'s Test queries by following the general instruction.

The general instruction for answering Reveal($\prod_i^\pi$) and Test($\prod_i^\pi$): $\mathcal{A}$ finds an entry (sid_i^π, gid_i^π, $\text{ncs}_{1,i}^\pi$, T) in NTList, asks the same query (either Reveal or Test query) to the U_i's instance activated by the Execute query that resulted in T, and then forwards the key K' received in return.

The simulation above is perfect for $\mathcal{A}^+$ as long as neither Repeat nor Forge occur. Note that even when a Send query is asked after some corruption, $\mathcal{A}$ perfectly simulates the actions of instances by using its own Dump queries.

To analyze the advantage of $\mathcal{A}$ in attacking protocol SKE, let $\text{Succ}_{\mathcal{A}}$ (resp. $\text{Succ}_{\mathcal{A}^+}$) be the event that $\mathcal{A}$ (resp. $\mathcal{A}^+$) correctly guesses the hidden bit used by its Test oracle. Then,

since $\mathcal{A}$ outputs whatever $\mathcal{A}^+$ does if neither Forge nor Repeat occur and outputs a random bit otherwise, it easily follows that:

$$\Pr[\text{Succ}_{\mathcal{A}}] = \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}} \wedge \overline{\text{Repeat}}] + \frac{1}{2}\Pr[\text{Forge} \vee \text{Repeat}]. \quad (6)$$

Since $\Pr[\text{Forge} \vee \text{Repeat}]$ is negligible, this equation implies that if $\Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}} \wedge \overline{\text{Repeat}}]$ is non-negligibly greater than 1/2, then so is $\text{Succ}_{\mathcal{A}}$.

To derive the statement of Theorem 2, we apply a series of simple modifications to the definitional equation $\text{Adv}_{\text{SKE}^+}(\mathcal{A}^+) = |2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+}] - 1|$ as follows:

$$\begin{aligned}
\text{Adv}_{\text{SKE}^+}(\mathcal{A}^+) &= |2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+}] - 1| \\
&= |2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \text{Forge}] + 2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}}] - 1| \\
&\le |2 \cdot \Pr[\text{Forge}] + 2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}}] - 1| \\
&= |2 \cdot \Pr[\text{Forge}] + 2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}} \wedge \text{Repeat}] + \\
&\quad 2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}} \wedge \overline{\text{Repeat}}] - 1|.
\end{aligned}$$

Applying Eq. (6) to the last equation above leads to:

$$\begin{aligned}
\text{Adv}_{\text{SKE}^+}(\mathcal{A}^+) \le |&2 \cdot \Pr[\text{Forge}] + 2 \cdot \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}} \wedge \text{Repeat}] - \\
&\Pr[\text{Forge} \vee \text{Repeat}] + 2 \cdot \Pr[\text{Succ}_{\mathcal{A}}] - 1|.
\end{aligned}$$

Since $\Pr[\text{Forge} \vee \text{Repeat}] \ge \Pr[\text{Forge}] + \Pr[\text{Succ}_{\mathcal{A}^+} \wedge \overline{\text{Forge}} \wedge \text{Repeat}]$, we get:

$$\text{Adv}_{\text{SKE}^+}(\mathcal{A}^+) \le |\Pr[\text{Forge} \vee \text{Repeat}]| + |2 \cdot \Pr[\text{Succ}_{\mathcal{A}}] - 1|.$$

From this, the following is immediate:

$$\text{Adv}_{\text{SKE}^+}(\mathcal{A}^+) \le |\Pr[\text{Forge} \vee \text{Repeat}] + 2 \cdot \Pr[\text{Succ}_{\mathcal{A}}] - 1|.$$

By the definition $\text{Adv}_{\text{SKE}}(\mathcal{A}) = |2 \cdot \Pr[\text{Succ}_{\mathcal{A}}] - 1|$, this inequality can be rewritten as:

$$\text{Adv}_{\text{SKE}^+}(\mathcal{A}^+) \le \text{Adv}_{\text{SKE}}(\mathcal{A}) + |\Pr[\text{Forge} \vee \text{Repeat}]|.$$

Now since $0 \le \Pr[\text{Forge} \vee \text{Repeat}] \le \Pr[\text{Forge}] + \Pr[\text{Repeat}]$ and since $\text{Adv}_{\text{SKE}}(\mathcal{A}) \le \text{Adv}_{\text{SKE}}(t', Q')$, we obtain:

$$\text{Adv}_{\text{SKE}^+}(\mathcal{A}^+) \le \text{Adv}_{\text{SKE}}(t', Q') + \Pr[\text{Forge}] + \Pr[\text{Repeat}].$$

This combined with Lemma 4 and Eq. (5) yields the statement of Theorem 2. ❑

6. CONCLUSION

We have proposed a new authenticated group key exchange (GKE) protocol provid-

ing forward secrecy. The protocol presented here is the first in the sense that up until now, there have been no forward-secure authenticated GKE protocols with constant round complexity and logarithmic computation complexity. Our protocol is also unique in that it is the only forward-secure authenticated GKE protocol that achieves logarithmic message and bit complexities in a broadcast network. Furthermore, our protocol attains provable security against active adversaries under the decisional Diffie-Hellman assumption. The formal model where the protocol has been proven secure captures the important security notions of implicit key authentication, forward secrecy, known key security, and strong corruption. In conclusion, we believe that our protocol represents the most scalable solution to the problem of securely establishing session keys in large groups.

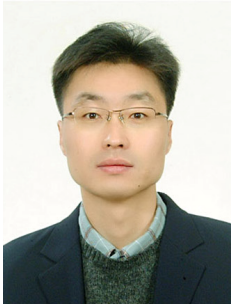
Although our definition of security is a de facto standard for analyzing group key exchange protocols, it does not guarantee any security against *insider attacks*. Indeed, almost all existing protocols including ours where the roles of participants are asymmetric are trivially vulnerable to, for example, *insider different key attacks* (or equivalently, in terms of Definition 1 given in [31], trivially fail to guarantee *agreement*). Even well-known symmetric protocols have been shown to be insecure in the face of malicious insiders [32]. Given the situation, the recent work [31] presented an interesting technique for transforming any protocol proven secure according to our security definition into one that is secure also against various insider attacks. Thus when insiders are suspected to be malicious, we may address the concern by applying this transformation technique. But then the resulting protocol would have a computation complexity which is at least linear in the number of users. We hence leave it as an open problem to find a protocol that has the same level of scalability as our protocol and also provides protection against insider attacks.

REFERENCES

1. E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater, "Provably authenticated group Diffie-Hellman key exchange," in *Proceedings of the 8th ACM Conference on Computer and Communications Security*, 2001, pp. 255-264.
2. C. Boyd and J. Nieto, "Round-optimal contributory conference key agreement," in *Proceedings of the 6th International Workshop on Practice and Theory in Public Key Cryptography*, LNCS 2567, 2003, pp. 161-174.
3. M. Abdalla, P. A. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Proceedings of the 8th International Workshop on Practice and Theory in Public Key Cryptography*, LNCS 3386, 2005, pp. 65-84.
4. K. K. Choo, C. Boyd, and Y. Hitchcock, "Errors in computational complexity proofs for protocols," in *Proceedings of Asiacrypt*, LNCS 3788, 2005, pp. 624-643.
5. J. Katz and M. Yung, "Scalable protocols for authenticated group key exchange," *Journal of Cryptology*, Vol. 20, 2007, pp. 265-294.
6. R. Dutta and R. Barua, "Provably secure costant round contributory group key agreement in dynamic setting," *IEEE Transactions on Information Theory*, Vol. 54, 2008, pp. 2007-2025.
7. S. Wang, Z. Cao, K. K. Choo, and L. Wang, "An improved identity-based key agreement protocol and its security proof," *Information Sciences*, Vol. 179, 2009, pp. 307-318.

8. T. Wu, Y. Tseng, and C. Yu, "A secure ID-based authenticated group key exchange protocol resistant to insider attacks," *Journal of Information Science and Engineering*, Vol. 27, 2011, pp. 915-932.
9. H. Kim, S. Lee, and D. Lee, "Constant-round authenticated group key exchange for dynamic groups," in *Proceedings of Asiacrypt*, LNCS 3329, 2004, pp. 245-259.
10. S. Hirose and S. Yoshida, "An authenticated Diffie-Hellman key agreement protocol secure against active attacks," in *Proceedings of the 1st International Workshop on Practice and Theory in Public Key Cryptography*, LNCS 1431, 1998, pp. 135-148.
11. M. Mayer and M. Yung, "Secure protocol transformation via "Expansion": from two-party to groups," in *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999, pp. 83-92.
12. B. Jung, S. Paeng, and D. Kim, "Attacks to Xu-Tilborg's conference key distribution scheme," *IEEE Communications Letters*, Vol. 8, 2004, pp. 446-448.
13. M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," in *Proceedings of Eurocrypt*, LNCS 950, 1995, pp. 275-286.
14. K. Becker and U. Wille, "Communication complexity of group key distribution," in *Proceedings of the 5th ACM Conference on Computer and Communications Security*, 1998, pp. 1-6.
15. M. Abdalla, P. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Proceedings of the 8th International Workshop on Practice and Theory in Public Key Cryptography*, LNCS 3386, 2005, pp. 65-84.
16. J. Nam, J. Lee, S. Kim, and D. Won, "DDH-based group key agreement in a mobile environment," *Journal of Systems and Software*, Vol. 78, 2005, pp. 73-83.
17. J. Nam, J. Paik, U. Kim, and D. Won, "Resource-aware protocols for authenticated group key exchange in integrated wired and wireless networks," *Information Sciences*, Vol. 177, 2007, pp. 5441-5467.
18. M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Proceedings of Eurocrypt*, LNCS 1807, 2000, pp. 139-155.
19. J. Nam, J. Paik, U. Kim, and D. Won, "Constant-round authenticated group key exchange with logarithmic computation complexity," in *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, LNCS 4521, 2007, pp. 158-176.
20. D. Wallner, E. Harder, and R. Agee, "Key management for multicast: issues and architectures," RFC 2627, 1999.
21. C. Wong, M. Gouda, and S. Lam, "Secure group communications using key graphs," *IEEE/ACM Transactions on Networking*, Vol. 8, 2000, pp. 16-30.
22. Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proceedings of the 7th ACM Conference on Computer and Communications Security*, 2000, pp. 235-244.
23. Y. Kim, A. Perrig, and G. Tsudik, "Communication-efficient group key agreement," in *Proceedings of the 16th International Conference on Information Security*, 2001, pp. 229-244.
24. S. Lee, Y. Kim, K. Kim, and D. Ryu, "An efficient tree-based group key agreement using bilinear map," in *Proceedings of the 1st International Conference on Applied Cryptography and Network Security*, LNCS 2846, 2003, pp. 357-371.

25. K. Ren, H. Lee, K. Kim, and T. Yoo, "Efficient authenticated key agreement protocol for dynamic groups," in *Proceedings of the 5th International Workshop on Information Security Applications*, LNCS 3325, 2004, pp. 144-159.
26. R. Barua, R. Dutta, and P. Sarkar, "Extending Joux's protocol to multi party key agreement," in *Proceedings of Indocrypt*, LNCS 2904, 2003, pp. 205-217.
27. R. Dutta, R. Barua, and P. Sarkar, "Provably secure authenticated tree based group key agreement," in *Proceedings of the 6th International Conference on Information and Communications Security*, LNCS 3269, 2004, pp. 92-104.
28. A. Joux, "A one round protocol for tripartite Diffie-Hellman," *Journal of Cryptology*, Vol. 17, 2003, 263-276.
29. M. Burmester and Y. Desmedt, "Efficient and secure conference-key distribution," in *Proceedings of International Workshop on Security Protocols*, LNCS 1189, 1997, pp. 119-129.
30. S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, Vol. 28, 1984, pp. 270-299.
31. J. Katz and J. Shin, "Modeling insider attacks on group key-exchange protocols," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, 2005, pp. 180-189.
32. Q. Tang and C. Mitchell, "Security properties of two authenticated conference key agreement protocols," in *Proceedings of the 7th International Conference on Information and Communications Security*, LNCS 3783, 2005, pp. 304-314.



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.

Minkyu Park received the B.E and M.E. degree in Computer Engineering from Seoul National University in 1991 and 1993, respectively. He received Ph.D. degree in Computer Engineering from Seoul National University in 2005. He is now an Associate Professor in Konkuk University, Korea. His research interest includes operating systems, real-time scheduling, embedded software, computer system security, and HCI.

Sangchul Han received his B.S. degree in Computer Science from Yonsei University in 1998 and his M.E. and Ph.D. degrees in Computer Engineering from Seoul National University in 2000 and in 2007, respectively. He is now an Assistant Professor of Department of Computer Engineering in Konkuk University. His research interests include computer security, real-time scheduling and embedded systems.

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 and 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 and 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.