Constant Round Group Key Exchange with Logarithmic Computational Complexity

Junghyun Nam, Youngsook Lee, and Dongho Won

School of Information and Communication Engineering, Sungkyunkwan University, Korea
{jhnam,yslee,dhwon}@security.re.kr

Abstract. Protocols for group key exchange (GKE) are cryptographic algorithms that describe how a group of parties communicating over a public network can come up with a common secret key. Due to their critical role in building secure multicast channels, a number of GKE protocols have been proposed over the years in a variety of settings. However despite many impressive achievements, it still remains a challenging problem to design a secure GKE protocol which scales very well for large groups. Our observation is that all provably-secure constant-round GKE protocols providing forward secrecy thus far are not fully scalable, but have a computational complexity that scales only linearly in group size. Motivated by this observation, we propose a new GKE protocol that not only offers full scalability in all directions but also attains provable security against active adversaries. Full scalability is achieved by using a complete binary tree structure where users are arranged on both internal and leaf nodes. Security is proved via reduction to the decisional Diffie-Hellman assumption in a well-defined formal model of communication and adversarial capabilities.

Keywords: Group key exchange, scalability, binary tree, provable security, DDH assumption.

1 Introduction

The primary goal of cryptography is to provide a means for communicating confidentially and with integrity over a public channel. Roughly speaking, confidentiality ensures that communications and messages are kept secret between authorized parties, and integrity guarantees that any unauthorized modifications to the transferred data will be detected. In practice, these two main security properties are best achieved with key exchange protocols which allow the parties communicating over an insecure network to establish a common secret key called a *session key*. That is, it is typical that the communicating parties, who want confidentiality and integrity, first generate a session key by running an appropriate key exchange protocol and then use this key together with standard cryptographic algorithms for message encryption and authentication. Thus, the problem of establishing confidential and integrity-preserving communication is commonly reduced to the problem of getting a right protocol for session key generation. Needless to say, a tremendous amount of research effort has been devoted to the design and analysis of key exchange protocols in a variety of different settings (e.g., [22, 29, 40, 9, 30] and their follow-ups).

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 (see [19] for a comprehensive list of examples). 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 [6, 8, 45, 5, 13, 18, 2, 32] 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 [31, 13, 33, 2, 37, 20].

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 group key exchange (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 sent per user in the protocol, and (3) the *bit complexity*, the maximum number of bits sent 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 should be able to run in a constant number of rounds. The computation cost of a protocol is directly related to the *computational complexity* which is defined as the maximum amount of computation done by any single user in the protocol. The computational complexity is mostly concerned with the number of public-key cryptographic operations such as modular exponentiations, signature generations, public-key decryptions, signature verifications, and public-key encryptions. (We note that the above definitions of various complexities are the same as those given in the full version of [33].[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 [13] with round complexity $O(n)$ in a group of size n. Although some protocols with provable security require only a constant number of rounds, they still suffer from the number of public-key operations that scales linearly in group size. Indeed, all the best-known constant-round protocols [11, 33, 34, 24] for authenticated key exchange have $O(n)$ computational complexity under the definition above. The protocol of [11] requires one distinct user to perform $O(n)$ public-key encryptions. The other protocols from [33, 34, 24] is all a novel extension of the protocol (i.e., protocol 3) by Burmester and Desmedt [14], 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 computational complexity scales logarithmically with group size. It is this observation that prompted the present work aimed at designing a provably-secure constant-round GKE protocol with logarithmic computational complexity.

Contribution. The result of this work is the first GKE protocol that is fully scalable in all key metrics and is provably authenticated in a well-defined formal model. To the best of our knowledge, all the provably-authenticated GKE protocols published up to now fail to achieve either constant round complexity or logarithmic computational complexity. We summarize in Table 1 the communication and computation requirements of our protocol and two other leading protocols [11, 33].[2] As the table 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. Furthermore, all the three measures for estimating the communication cost remain constant in our protocol, regardless of the number of users. The protocol of [11][3] features optimal round complexity, but lacks perfect forward secrecy [23].

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 [13, 11, 33, 34, 24]. From the standpoint of the adversary's capabilities, our security model is a unique combination of previous results from [13, 12, 2, 39],

[1] The full version of [33] is available at `http://eprint.iacr.org/2003/171`.

[2] Although the protocols from [34, 24] may perform better in practice than the protocol of [33], they fall into the same category from the computational complexity perspective.

[3] We refer to [20] for a security enhancement to this protocol.

Table 1. Complexity comparison

	Communication			Computation				
	Rounds	Messages	Bits	Exp	Sig/Dec	Ver/Enc	Div	Mul
Boyd-Nieto [11]	1	$O(1)$	$O(n)$		$O(1)$	$O(n)$		
Katz-Yung [33]	3	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$	$O(n \log n)$
Here	3	$O(1)$	$O(1)$	$O(\log n)$	$O(1)$	$O(\log n)$	$O(1)$	$O(\log n)$

Note. The communication cost is measured in a broadcast network model, but this is only for clarity of comparison.
Rounds: the number of communication rounds required to complete the protocol.
Messages: the maximum number of messages sent by any single user.
Bits: the maximum number of bits sent by any single user.
Exp: the maximum number of modular exponentiations performed per user.
Sig/Dec: the maximum number of signature generations or public-key decryptions performed per user.
Ver/Enc: the maximum number of signature verifications or 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.

which are in turn based on earlier work of Bellare, Pointcheval, and Rogaway [5]. 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. Secondly, we incorporate *strong corruption* [5] into the model by allowing the adversary to ask users to release any short-term and long-term secret information. A detailed discussion on this is deferred to Section 2. Our security proof of course captures important security notions of perfect forward secrecy and known key security [21]. 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.

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. [47] and Wong et al. [48] 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, e.g., [16, 41, 44]) 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. [35, 36] to the forward-secure case. Their protocols require no secure channels of any kind and offer distributed functionality. Later, Lee et al. [38] present a paring-based variant of the TGDH protocol of [35]. All these works [35, 36, 38], however, provide no explicit treatment of key exchange for initial group formation, focusing only on key updates upon group membership changes.

Ren et al. [43] 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. [4] and Dutta et al. [25] construct their protocols by combining a ternary tree structure with the one-round three-party protocol of Joux [30]. Back in 1994, Burmester and Desmedt [14] also proposed a tree-based GKE protocol. This protocol (i.e., protocol 2 of [14]) seems to be the first GKE protocol utilizing a binary tree structure, and distinguishes itself from all other protocols mentioned here in that there exists a bijective mapping between users and nodes of the tree used in the protocol. But, this protocol, in common with other protocols from [4, 43, 25], 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 [15] presented another tree-based protocol called CKDS. The CKDS protocol (more specifically, the multicast version of CKDS) improves on the protocol 2 of [14] and achieves the same level of complexities as our protocol. Interestingly, this protocol has not received

much attention in the literature despite its scalability. More interestingly, CKDS in its basic form is similar to our unauthenticated protocol (the SKE protocol in Section 3.1), although the two protocols are independently developed. However, CKDS considering passive adversaries only carries no formal analysis of security and instead justifies its security on purely heuristic grounds. In contrast, our protocol comes along with a rigorous proof of its semantic security against active adversaries in a concrete and well-established model. In this sense, another way to see our result may be as a provably-authenticated improvement on the CKDS protocol.

Organization. The remainder of this paper is organized as follows. Section 2 describes a formal model for security analysis of GKE protocols and reviews some cryptographic assumptions underlying the security of our construction. Section 3 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 [33]. Concluding this work, Section 5 poses a challenging open problem. Proofs of security for the unauthenticated and authenticated protocols are provided in Appendices A and B, respectively.

2 Formal Setting

Any form of security analysis of a cryptographic construction should be preceded by clear definitions of its security goals and tools. In this section we provide such a preliminary formalism for group key exchange, introducing our communication and adversarial model with an associated definition of security and describing some cryptographic building blocks used to construct our protocol.

2.1 Communication and Adversarial Model

Participants. Let $\mathcal{U}$ be a set of all users who are potentially interested in participating in a group key exchange protocol. The users in any subset of $\mathcal{U}$ may run the group key exchange protocol at any point in time to establish a session key. Each user may run the protocol multiple times either serially or concurrently, with possibly different groups of participants. Thus, at a given time, there could be many instances of a single user. We use Π_U^i to denote instance i of user U. Before the protocol is executed for the first time, each user U creates a long-term public/private key pair (PK_U, SK_U) by running a key generation algorithm $\mathcal{K}(1^\kappa)$. All instances of a user share the public/private keys of the user even if they participate in their respective sessions independently. Each private key is kept secret by its owner while the public keys of all users are publicized.

Partners. Intuitively, the *partners* of an instance is the set of all instances that should compute the same session key as the instance in an execution of the protocol. Like most of previous work, we use the notion of *session IDs* to define partnership between instances. Literally, a session ID (denoted as sid) is a unique identifier of a communication session. Following [5, 17, 18, 32], we assume that session IDs are assigned and provided by some higher-level protocol. While this assumption is unnecessary in some protocols [11, 33] which use only broadcast messages (in these protocols, a session ID can readily be defined as the concatenation of all message flows), it seems very useful in other protocols where some protocol messages are not broadcast and thus not all participants have the same view of a protocol run. We let $\mathcal{SID}$ be the algorithm used by the higher-level protocol to generate session IDs, and assume that $\mathcal{SID}$ is publicly available.

We also need the notion of *group IDs* to define partnership properly. A group ID (denoted as gid) is a set consisting of the identities of the users who intend to establish a session key

among themselves. This notion is clearly natural because it is impossible (not even defined) to ever execute a group key exchange protocol without participants. Indeed, a group ID is a both necessary and important input to any protocol execution.

In order for an instance to start to run the protocol, we require that both sid and gid should be given as input to the instance. We use sid_U^i and gid_U^i to denote respectively sid and gid provided to instance Π_U^i. Note that gid_U^i should always include U itself. Session IDs and group IDs are public and hence available to the adversary. Indeed, the adversary in our model generates these IDs on its own; it generates a session ID by running $\mathcal{SID}$ and a group ID by choosing a subset of $\mathcal{U}$.

An instance is said to *accept* when it successfully computes a session key in a protocol execution. Let acc_U^i be a boolean variable that evaluates to TRUE if Π_U^i has accepted, and FALSE otherwise. We say that any two instances Π_U^i and $\Pi_{U'}^j$, where $U \neq U'$, are *partners* of each other, or equivalently, *partnered* iff all the following three conditions are satisfied: (1) $\mathsf{sid}_U^i = \mathsf{sid}_{U'}^j$, (2) $\mathsf{gid}_U^i = \mathsf{gid}_{U'}^j$, and (3) $\mathsf{acc}_U^i = \mathsf{acc}_{U'}^j =$ TRUE. We also say that two instances Π_U^i and $\Pi_{U'}^j$ ($U \neq U'$) are *potential partners* of each other, or equivalently, *potentially partnered* iff the first two conditions above hold. We use pid_U^i and ppid_U^i to denote respectively the partners and the potential partners of the instance Π_U^i. Then it follows by the definitions that $\mathsf{pid}_U^i \subseteq \mathsf{ppid}_U^i$.

Adversary. The adversary in our model controls all message exchanges in the protocol and can ask participants to open up access to any secrets, either long-term or short-term. These capabilities of the adversary are modeled via various oracles to which the adversary is allowed to make queries. Unlike most previous models for group key exchange, we allow the adversary to query the Test oracle as many times as it wants[4]. This approach was recently suggested by Abdalla et al. [2] for password authenticated key exchange in the three-party setting and was also proved there to lead to a stronger model (for more details, see Lemmas 1 and 2 in Appendix B of [2]). What we found interesting is that allowing multiple Test queries is very useful in proving Theorem 1 which claims the security of our unauthenticated protocol against a passive adversary. We also strengthen the model by incorporating *strong corruption* [5] in which the adversary is allowed to ask user instances to release both short-term and long-term secrets. We treat strong corruption in a different manner than [5][5], and follow [39] in that we provide the adversary with an additional oracle called Dump which returns all short-term secrets used by an instance. Other oracles (Execute, Send, Reveal, and Corrupt) are as usual. In the following, we describe these relatively familiar oracles first and then Dump and Test oracles.

- Execute(sid, gid): This query prompts an honest execution of the protocol between a set of instances consisting of one instance for each user in gid, where the instances are all given the session ID sid and the group ID gid as their input. The transcript of the honest execution is returned to the adversary as the output of the query. This models passive attacks on the protocol.
- Send(Π_U^i, M): This query sends message M to instance Π_U^i. The instance Π_U^i proceeds as it would in the protocol upon receiving message M; the instance updates its state performing any required computation, and generates and sends out a response message as needed. The response message, if any, is the output of this query and is returned to the adversary. This models active attacks on the protocol, allowing the adversary to control at will all message flows between instances. A query of the form Send(Π_U^i, sid$\|$gid) prompts Π_U^i to initiate an execution of the protocol using session ID sid and group ID gid.

[4] The model in [1] appears to be the first one for group key exchange that does not restrict the adversary to ask only a single Test query.

[5] In the strong corruption model of [5], the Corrupt oracle returns both long-term and short-term secrets.

- $\mathsf{Reveal}(\Pi_U^i)$[6]: This query returns to the adversary the session key held by Π_U^i. This can be asked only if $\mathsf{acc}_U^i = \text{TRUE}$ and if the adversary has not queried $\mathsf{Test}(\Pi_{U'}^j)$ for some $\Pi_{U'}^j$ in pid_U^i (we will discuss this matter further in Section 2.2). Allowing this query enables us to see whether or not the protocol is secure against known key attacks.
- $\mathsf{Corrupt}(U)$: This query returns to the adversary all long-term secret information of U including the private key SK_U[7]. This models the adversary's capability of breaking into a user's machine and gaining access to the long-term data set stored there. The adversary can issue this query at any time regardless of whether U is currently executing the protocol or not. This query is considered to deal with forward secrecy of the protocol.
- $\mathsf{Dump}(\Pi_U^i)$: This query returns *all short-term secrets* used in the past or currently being used by instance Π_U^i[8]. But, neither the session key computed by Π_U^i nor any long-term secrets of U are not returned. This models the adversary's capability to embed a Trojan horse or other form of malicious code into a user's machine and then log all the session-specific information of the victim. The adversary is not allowed to ask this query if it has already queried $\mathsf{Test}(\Pi_{U'}^j)$ for some $\Pi_{U'}^j$ in ppid_U^i (see Section 2.2 for the reason for this).
- $\mathsf{Test}(\Pi_U^i)$: This query provides a means of defining security. The output of this query depends on the hidden bit b that the Test oracle chooses uniformly at random from $\{0, 1\}$ during its initialization phase. The Test oracle returns the real session key held by Π_U^i if $b = 1$, or returns a random session key drawn from the key space if $b = 0$. The adversary is allowed to query the Test oracle as many times as necessary. But, the query can be asked only when instance Π_U^i is *fresh* (see Definition 1 given below). All the queries to the oracle are answered using the same value of the hidden bit b that was chosen at the beginning. Namely, the keys returned by the Test oracle are either all real or all random.

Remark 1. The Dump oracle is essentially similar to the Session-state reveal oracle introduced in [17]. But as noted in [39], there is a technical difference between these two oracles. The Session-state reveal oracle can be queried only to obtain the internal state of an incomplete session, whereas the Dump oracle allows the adversary to obtain the recording of local history of an either incomplete or complete session.

Definition 1. *The instance Π_U^i is considered* unfresh *iff any of the following conditions hold:*

1. $\mathsf{acc}_U^i = \text{FALSE}$.
2. *The adversary queried* $\mathsf{Corrupt}(U')$ *for some U' in gid_U^i before some Π_V^j in ppid_U^i accepts.*
3. *The adversary queried* $\mathsf{Dump}(\Pi_{U'}^j)$ *for some $\Pi_{U'}^j$ in ppid_U^i.*
4. *The adversary queried* $\mathsf{Reveal}(\Pi_{U'}^j)$ *or* $\mathsf{Test}(\Pi_{U'}^j)$ *for some $\Pi_{U'}^j$ in pid_U^i.*

All other instances are considered fresh.

Remark 2. By "$\mathsf{Test}(\Pi_{U'}^j)$" in the fourth condition of Definition 1, we require that for each different set of partners, the adversary should access the Test oracle only once. One may think that this restriction weakens the ability of the adversary. However this is not the case because when all information on partnering is public, obtaining the same data multiple times (from a given set of instances partnered together) is no different than obtaining it once.

Remark 3. Regarding the restriction mentioned in Remark 2, we should point out a minor definitional error in the ROR model of [2]. The ROR model does not impose our restriction on

[6] While the Reveal oracle does not exist in the so-called ROR model of Abdalla et al. [2], it is available to the adversary in our model and is used to prove Lemma 1, enabling a modular approach in the security proof of our protocol. Anyway, allowing Reveal queries causes no harm, but rather provides more clarity.

[7] This definition of the Corrupt oracle corresponds to the so-called *weak corruption model* [5].

[8] This combined with the Corrupt oracle represents strong corruption.

accessing the Test oracle, but rather it allows the adversary to ask multiple Test queries regardless of whether the tested instances are partnered together or not. To make this approach workable, the ROR model mandates that in the case of $b = 0$, the Test oracle must return the same random key for all Test queries that are directed to the instances which are partnered together. At first glance, this way of accessing the Test oracle seems to work well enough, avoiding the need for our restriction. We note however that the ROR model mistakenly opts for this seemingly innocuous approach. The problem is that allowing the adversary to access the Test oracle more than once for each set of partners invalidates the proof of Lemma 2 in Appendix B of [2]. The reason for this will be explained at an appropriate point later in this paper (see Remark 4 after the proof of Lemma 1 in Section A). But, this is a minor discrepancy and can be easily resolved by imposing our restriction on accessing the Test oracle.

Definition 2. *An adversary is called* active *iff it is allowed to access all the oracles described above, and called* passive *iff it is allowed to access all but the* Send *oracle.*

We represent the amount of queries used by an adversary as an ordered sequence of six non-negative integers, $Q = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$, where the six elements refer to the numbers of queries that the adversary made respectively to its Execute, Send, Reveal, Corrupt, Dump, and Test oracles. We call this usage of queries by an adversary the *query complexity* of the adversary. Note that by Definition 2, the query complexity of a passive adversary is always represented as a sequence of the form $Q = (q_{\text{exec}}, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$.

2.2 Definitions of Security

Having introduced the communication and adversarial model, we now proceed to define what is meant for a group key exchange protocol to be secure. We then briefly review the cryptographic assumptions on which the security of our construction depends.

Secure Key Exchange. The security of a group key exchange protocol P against an adversary $\mathcal{A}$ is defined in terms of the probability that $\mathcal{A}$ succeeds in distinguishing random session keys from real session keys established by the protocol P. That is, the adversary $\mathcal{A}$ is considered successful in attacking P if it breaks the semantic security of session keys generated by P. This notion of security is defined in the context of the following two-stage game, where the goal of adversary $\mathcal{A}$ is to correctly guess the value of the hidden bit b chosen by the Test oracle.

- **Stage 1:** $\mathcal{A}$ makes queries to the Execute, Send (only if $\mathcal{A}$ is an active adversary), Reveal, Corrupt, Dump, and Test oracles as many times as it wants.
- **Stage 2:** Once $\mathcal{A}$ decides that Stage 1 is over, it outputs a bit b' as a guess for the value of the hidden bit b used by the Test oracle. $\mathcal{A}$ wins the game if $b = b'$.

In the game above, the adversary can keep querying the oracles even after it asked some Test queries. However, when there was the query $\mathsf{Test}(\Pi_U^i)$ asked, the adversary is prohibited from querying $\mathsf{Dump}(\Pi_{U'}^j)$ for some $\Pi_{U'}^j \in \mathsf{ppid}_U^i$ and from querying $\mathsf{Reveal}(\Pi_{U'}^j)$ for some $\Pi_{U'}^j \in \mathsf{pid}_U^i$. This restriction reflects the fact that the adversary can win the game unfairly by using the information obtained via the query $\mathsf{Dump}(\Pi_{U'}^j)$ or $\mathsf{Reveal}(\Pi_{U'}^j)$.

Given the game above, the advantage of $\mathcal{A}$ in attacking the protocol P is defined as $\mathsf{Adv}_P(\mathcal{A}) = |2 \cdot \Pr[b = b'] - 1|$. Note that this definition is equivalent to say that the advantage of $\mathcal{A}$ is the difference between the probabilities that $\mathcal{A}$ outputs 1 in the following two experiments constituting the game: the *real experiment* where all queries to the Test oracle are answered with the real session key, and the *random experiment* where all Test queries are answered with a random session key. Thus, if we denote the real and the random experiments respectively as $\mathsf{Exp}_P^{\mathsf{real}}(\mathcal{A})$

and $\mathsf{Exp}_P^{\mathsf{rand}}(\mathcal{A})$, the advantage of $\mathcal{A}$ can be equivalently defined as $\mathsf{Adv}_P(\mathcal{A}) = |\Pr[\mathsf{Exp}_P^{\mathsf{real}}(\mathcal{A}) = 1] - \Pr[\mathsf{Exp}_P^{\mathsf{rand}}(\mathcal{A}) = 1]|$, where the outcomes of the experiments is the bit output by $\mathcal{A}$.

We say that the group key exchange protocol P is *secure* if $\mathsf{Adv}_P(\mathcal{A})$ is negligible for all probabilistic polynomial time adversaries $\mathcal{A}$. To quantify the security of protocol P in terms of the amount of resources expended by adversaries, we let $\mathsf{Adv}_P(t, Q)$ denote the maximum value of $\mathsf{Adv}_P(\mathcal{A})$ over all $\mathcal{A}$ with time complexity at most t and query complexity at most Q.

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 $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{D}) = |\Pr[\mathcal{D}(\mathbb{G}, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{D}(\mathbb{G}, g, g^x, g^y, g^z) = 1]|$. We say that the DDH assumption holds in $\mathbb{G}$ (or equivalently, the DDH problem is hard in $\mathbb{G}$) if $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{D})$ is negligible for all probabilistic polynomial time algorithms $\mathcal{D}$. We denote by $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(t)$ the maximum value of $\mathsf{Adv}_{\mathbb{G}}^{\mathsf{ddh}}(\mathcal{D})$ over all $\mathcal{D}$ running in time at most t.

Signature Schemes. Let $\Sigma = (\mathsf{Kgen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme, where Kgen is the key generation algorithm, Sign is the signature generation algorithm, and Vrfy is the signature verification algorithm. Let $\mathsf{Succ}_{\Sigma}(\mathcal{A})$ denote the probability that $\mathcal{A}$ succeeds in generating an existential forgery under adaptive chosen message attack [28, 42]. We say that a signature scheme Σ is secure if $\mathsf{Succ}_{\Sigma}(\mathcal{A})$ is negligible for every probabilistic polynomial time adversary $\mathcal{A}$. We use $\mathsf{Succ}_{\Sigma}(t)$ to denote the maximum value of $\mathsf{Succ}_{\Sigma}(\mathcal{A})$ over all $\mathcal{A}$ running in time at most t.

3 A Scalable Protocol for Unauthenticated Group Key Exchange

This section presents a new group key exchange protocol called SKE (Scalable Key Exchange). Let $\mathcal{G} = \{U_1, U_2, \ldots, U_n\}$ 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 group key exchange protocol that has round complexity $O(1)$ and computational 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 $\mathcal{G}$.

3.1 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 $g \neq 1$ 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 $I : \mathbb{G} \rightarrow \mathbb{Z}_q$ be bijective and (for any element in $\mathbb{G}$) efficiently computable. Whether there are appropriate bijections from $\mathbb{G}$

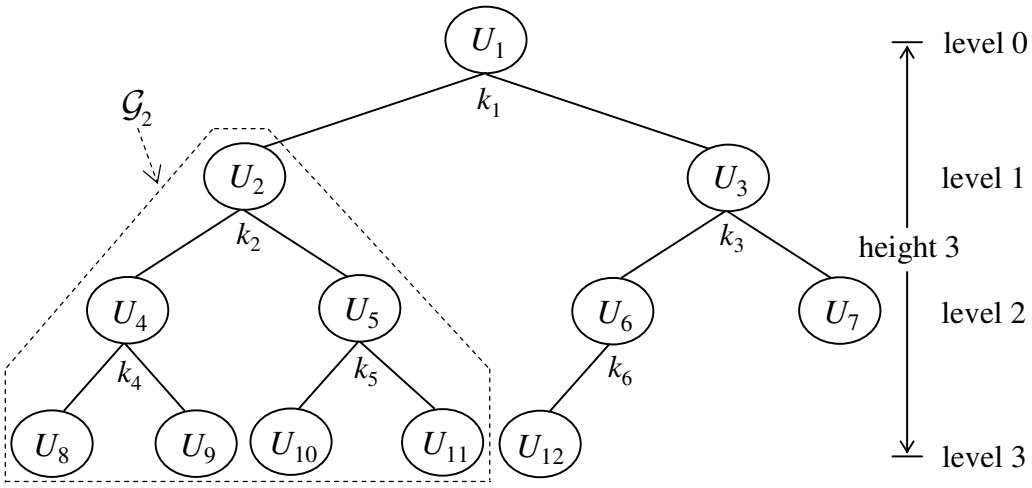

Fig. 1. A complete binary tree for $\mathcal{G} = \{U_1, \ldots, U_{12}\}$

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:

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

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:

1. 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 $y_{2i+1} = k_i x_{2i+1}^{-1}$.
2. 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:

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

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.

It is easy to see that this two-round protocol requires each user to send only a constant number of messages and bits without respect to the number of users in $\mathcal{G}$. Furthermore, the maximum amount of computation performed by a user in the protocol increases linearly with the *height* of the tree, i.e., logarithmically with the number of users.

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. However, as we will see in the next section, this unauthenticated key transport protocol can be converted into an authenticated key agreement protocol without compromising the protocol's full scalability and provable security (cf. Theorem 1).

3.2 Security Result for Protocol SKE

The following theorem presents our result on the security of protocol SKE. It says, roughly, that under the DDH assumption for $\mathbb{G}$, the group key exchange protocol SKE is secure against passive adversaries.

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:*

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

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.*

Outline of Proof. The full proof of Theorem 1 is given in Appendix A. At a high level, the proof proceeds by a mathematical induction on the height of the binary tree used in protocol SKE. Let SKE_h denote the protocol SKE but with the height of its input tree restricted to some fixed value $h > 0$. Namely, SKE_h is exactly the same as SKE, except that it can be run only for those groups such that $2^h \leq n < 2^{h+1}$. Then the basis step is to show that protocol SKE_1 is secure against passive adversaries. We take care of the basis step by Corollary 2. The induction step is to prove that for each $h \geq 1$, protocol SKE_{h+1} is secure against passive adversaries under the assumption of the security of protocol SKE_h against passive adversaries. The induction step is covered by Corollary 3. Consequently, Theorem 1 follows immediately from Corollary 2 and Corollary 3.

Towards the goal of proving the corollaries, we start with Lemma 1 which states that any key exchange protocol secure against passive adversaries making only a *single* Test query is also secure against passive adversaries who make *multiple* Test queries. Proving this lemma allows us to limit our security concern only to those cases where adversaries access the Test oracle only once. In Section A.1, we continue by proving Lemma 2 which plays a key role in deriving Corollary 2. Lemma 2 says that protocol SKE_1 is secure against passive adversaries asking only one query to the Test oracle, as long as the DDH assumption holds in $\mathbb{G}$. Combining Lemmas 1 and 2 completes the proof of Corollary 2. In Section A.2, we turn to Corollary 3 which is proved analogously to Corollary 2. Given Lemma 1, Corollary 3 directly follows from Lemma 3, the equivalent of Lemma 2 for the induction step. Thus, our final task is to prove Lemma 3, by

which we claim that for each $h \geq 1$, if the protocol SKE_h is secure against passive adversaries (asking multiple Test queries), then protocol SKE_{h+1} is secure against passive adversaries who make only one Test query. All the lemmas mentioned above are proved by a standard reduction argument.

4 A Scalable Protocol for Authenticated Group Key Exchange

Perhaps one of the most pleasing results of research on group key exchange is the one-round compiler presented by Katz and Yung [33] (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 let all the messages of the original protocol be signed and verified with the nonces. In this section, we convert the unauthenticated key transport protocol SKE into the authenticated key agreement 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_{U_i}, SK_{U_i}) by running $\mathsf{Kgen}(1^\kappa)$ and makes the verification key PK_{U_i} public. Recall that each user U_i receives as input a pair of session and group IDs $(\mathsf{sid}_{U_i}, \mathsf{gid}_{U_i})$ to participate in a protocol run. The protocol SKE^+ works in three rounds as follows:

Round 1: Each user $U_i \in \mathcal{G}$ chooses a random nonce $\tilde{k}_i \in \{g^0, g^1, \ldots, g^{q-1}\}$ and sends $\tilde{M}_i^0 = U_i \| 0 \| \tilde{k}_i$ to all other users in gid_{U_i}. After receiving all nonces from other users, user U_i sets $\mathsf{ncs}_{U_i} = \{(U_j, \tilde{k}_j) \mid U_j \in \mathsf{gid}_{U_i}\}$.

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 $\sigma_i^1 = \mathsf{Sign}_{SK_{U_i}}(U_i \| 1 \| z_i \| \mathsf{sid}_{U_i} \| \mathsf{ncs}_{U_i})$. 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 user 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 $\mathsf{Vrfy}_{PK_{U_j}}(U_j \| 1 \| z_j \| \mathsf{sid}_{U_i} \| \mathsf{ncs}_{U_i}, \sigma_j^1) = 1$. 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 $\sigma_i^2 = \mathsf{Sign}_{SK_{U_i}}(U_i \| 2 \| w_i \| y_{2i} \| y_{2i+1} \| \mathsf{sid}_{U_i} \| \mathsf{ncs}_{U_i})$, and sends $\tilde{M}_i^2 = U_i \| 2 \| w_i \| y_{2i} \| y_{2i+1} \| \sigma_i^2$ 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 $\mathsf{Vrfy}_{PK_{U_j}}(U_j \| 2 \| w_j \| y_{2j} \| y_{2j+1} \| \mathsf{sid}_{U_i} \| \mathsf{ncs}_{U_i}, \sigma_j^2) = 1$. If any of the verifications fail, U_i terminates without accepting. Otherwise, U_i computes its session key as follows. Assume first that $\mathsf{gid}_{U_i} = \{U_1, U_2, \ldots, U_n\}$, and let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a one-way hash function, where ℓ is the length of the session key to be distributed in SKE^+. Then, U_i derives the root node key k_1 as in SKE and calculates the session key K as

$$K = H(k_1 \| \tilde{k}_2 \| \tilde{k}_3 \| \cdots \| \tilde{k}_n).$$

The transformation above is essentially the same as the transformation by the KY compiler, except that here nonces are drawn from $\mathbb{G}$ and are used in session key calculation (another minor modification is the inclusion of sid in signing protocol messages). But the similarity does not mean that the proof of security for the KY compiler can be directly applicable to the SKE$^+$ protocol. Because the adversarial model where the KY compiler was proven secure is weaker than ours, the security of protocol SKE$^+$ should be reconsidered in light of our stronger model.

4.2 Security Result for Protocol SKE$^+$

As the following theorem states, the group key exchange protocol SKE$^+$ is secure against active adversaries in the random oracle model under the security of protocol SKE against passive adversaries.

Theorem 2. *Let* $Q = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$ *and* $Q' = (q_{\text{exec}} + q_{\text{send}}/2, 0, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}} + q_{\text{send}}/2, q_{\text{test}})$. *Assume that the hash function* H *used in protocol* SKE$^+$ *to derive the session key is a random oracle. 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:*

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

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.*

Tightness of Theorem 2. We prove the theorem in Appendix B 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 protocol 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 protocol SKE$^+$, the adversary $\mathcal{A}^+$ is unable to relay messages between user instances holding different sets of nonces. Based on this idea, the adversary $\mathcal{A}$ generates a transcript T^+ of protocol SKE$^+$, by querying its Execute oracle to obtain a transcript T of protocol SKE and patching the transcript T with appropriate signatures which are all generated from the same nonce set ncs. $\mathcal{A}$ then use this transcript T^+ in answering $\mathcal{A}^+$'s Send queries directed to the instances possessing ncs. 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 using its own Execute queries 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 Appendix A), which states that in attacking a key exchange protocol, the advantage of any adversary with query complexity $Q = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{dump}}, q_{\text{test}})$ is at most q_{test} times the maximum advantage obtainable by an adversary with query complexity $Q' = (q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$. 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 Conclusion and Open Problem

We have proposed a new group key exchange protocol. The protocol is the first in the sense that it not only offers full scalability in all key aspects of performance but also provides provable security against active adversaries. This means that our protocol provides a secure and practical way to generate session keys even for very 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 [32], trivially fail to guarantee *agreement*). Even well-known symmetric protocols have been shown to be insecure in the face of malicious insiders [10, 46]. Given the situation, the recent work [32] 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 computational 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. M. Abdalla, E. Bresson, O. Chevassut, and D. Pointcheval. Password-based group key exchange in a constant number of rounds. *9th International Workshop on Practice and Theory in Public Key Cryptography (PKC '06)*, LNCS vol. 3958, pp. 427–442, 2006.
2. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. *8th International Workshop on Practice and Theory in Public Key Cryptography (PKC '05)*, LNCS vol. 3386, pp. 65–84, 2005.
3. G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 4, pp. 628–639, 2000.

4. R. Barua, R. Dutta, and P. Sarkar. Extending Joux's protocol to multi party key agreement. *Progress in Cryptology – INDOCRYPT '03*, LNCS vol. 2904, pp. 205–217, 2003.
5. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. *Advances in Cryptology – EUROCRYPT '00*, LNCS vol. 1807, pp. 139–155, 2000.
6. M. Bellare and P. Rogaway. Entity authentication and key distribution. *Advances in Cryptology – CRYPTO '93*, LNCS vol. 773, pp. 232–249, 1993.
7. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *1st ACM Conference on Computer and Communications Security (CCS '93)*, pp. 62–73, 1993.
8. M. Bellare and P. Rogaway. Provably secure session key distribution — the three party case. *27th ACM Symposium on Theory of Computing (STOC '95)*, pp. 57–66, 1995.
9. S. Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. *1992 IEEE Symposium on Security and Privacy*, pp. 72–84, 1992.
10. J.-M. Bohli, M. Vasco, and R. Steinwandt. Secure group key establishment revisited. *Cryptology ePrint Archive*, Report 2005/395, 2005. Available at `http://eprint.iacr.org/`.
11. C. Boyd and J. Nieto. Round-optimal contributory conference key agreement. *6th International Workshop on Practice and Theory in Public Key Cryptography (PKC '03)*, LNCS vol. 2567, pp. 161–174, 2003.
12. E. Bresson, O. Chevassut, and D. Pointcheval. Group Diffie-Hellman key exchange secure against dictionary attacks. *Advances in Cryptology – ASIACRYPT '02*, LNCS vol. 2501, pp. 497–514, 2002.
13. E. Bresson, O. Chevassut, D. Pointcheval, and J.-J. Quisquater. Provably authenticated group Diffie-Hellman key exchange. *8th ACM Conference on Computer and Communications Security (CCS '01)*, pp. 255–264, 2001.
14. M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. *Advances in Cryptology – EUROCRYPT '94*, LNCS vol. 950, pp. 275–286, 1995.
15. M. Burmester and Y. Desmedt. Efficient and secure conference-key distribution. *1996 International Workshop on Security Protocols*, LNCS vol. 1189, pp. 119-129, 1997.
16. R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: a taxonomy and some efficient constructions. *IEEE INFOCOM '99*, vol. 2, pp. 708–716, 1999.
17. R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. *Advances in Cryptology – EUROCRYPT '01*, LNCS vol. 2045, pp. 453–474, 2001.
18. R. Canetti and H. Krawczyk. Universally composable notions of key exchange and secure channels. *Advances in Cryptology – EUROCRYPT '02*, LNCS vol. 2332, pp. 337–351, 2002.
19. K.-K. R. Choo. Provably-secure mutual authentication and key establishment protocols lounge. 2006. Available at `http://sky.fit.qut.edu.au/ choo/lounge.html`.
20. K.-K. Choo, C. Boyd, and Y. Hitchcock. Errors in computational complexity proofs for protocols. *Advances in Cryptology – ASIACRYPT '05*, LNCS vol. 3788, pp. 624–643, 2005.
21. D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, vol. 24, no. 8, pp. 533–536, 1981.
22. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
23. W. Diffie, P. Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.
24. R. Dutta and R. Barua. Constant round dynamic group key agreement. *8th International Conference on Information Security (ISC '05)*, LNCS vol. 3650, pp. 74–88, 2005.
25. R. Dutta, R. Barua, and P. Sarkar. Provably secure authenticated tree based group key agreement. *6th International Conference on Information and Communications Security (ICICS '04)*, LNCS vol. 3269, pp. 92–104, 2004.
26. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, vol. 33, no. 4, pp. 792–807, 1986.
27. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.
28. S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, vol. 17, no. 2, pp. 281–308, 1988.
29. I. Ingemarsson, D. Tang, and C. Wong. A conference key distribution system. *IEEE Transactions on Information Theory*, vol. 28, no. 5, pp. 714–720, 1982.
30. A. Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, vol. 17, no. 4, pp. 263–276, 2003. A preliminary version was presented at *ANTS IV*.
31. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. *Advances in Cryptology – EUROCRYPT '01*, LNCS vol. 2045, pp. 475–494, 2001.
32. J. Katz and J. Shin. Modeling insider attacks on group key-exchange protocols. *12th ACM Conference on Computer and Communications Security (CCS '05)*, pp. 180–189, 2005.
33. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. *Advances in Cryptology – CRYPTO '03*, LNCS vol. 2729, pp. 110–125, 2003.

34. H.-J. Kim, S.-M. Lee, and D. Lee. Constant-round authenticated group key exchange for dynamic groups. *Advances in Cryptology – ASIACRYPT '04*, LNCS vol. 3329, pp. 245–259, 2004.
35. Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. *7th ACM Conference on Computer and Communications Security (CCS '00)*, pp. 235–244, 2000.
36. Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. *IFIP SEC '01*, pp. 229–244, 2001.
37. H. Krawczyk. HMQV: a high-performance secure Diffie-Hellman protocol. *Advances in Cryptology – CRYPTO '05*, LNCS vol. 3621, pp. 546–566, 2005.
38. S. Lee, Y. Kim, K. Kim, and D.-H. Ryu. An efficient tree-based group key agreement using bilinear map. *1st International Conference on Applied Cryptography and Network Security (ACNS '03)*, LNCS vol. 2846, pp. 357–371, 2003.
39. 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, no. 1, pp. 73–83, 2005.
40. E. Okamoto and K. Tanaka. Key distribution system based on identification information. *IEEE Journal on Selected Areas in Communications*, vol. 7, no. 4, pp. 481–485, 1989.
41. A. Perrig, D. Song, and J. Tygar. ELK, a new protocol for efficient large-group key distribution. *2001 IEEE Symposium on Security and Privacy*, pp. 247–262, 2001.
42. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, vol. 13, no. 3, pp. 361–396, 2000.
43. K. Ren, H. Lee, K. Kim, and T. Yoo. Efficient authenticated key agreement protocol for dynamic groups. *5th International Workshop on Information Security Applications (WISA '04)*, LNCS vol. 3325, pp. 144–159, 2004.
44. A. Sherman and D. McGrew. Key establishment in large dynamic groups using one-way function trees. *IEEE Transactions on Software Engineering*, vol. 29, no. 5, pp. 444–458, 2003.
45. V. Shoup. On formal models for secure key exchange. *Cryptology ePrint Archive*, Report 1999/012, 1999. Available at `http://eprint.iacr.org/`.
46. Q. Tang and C. Mitchell. Security properties of two authenticated conference key agreement protocols. *7th International Conference on Information and Communications Security (ICICS '05)*, LNCS vol. 3783, pp. 304–314, 2005.
47. D. Wallner, E. Harder, and R. Agee. Key management for multicast: issues and architectures. RFC 2627, 1999.
48. C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *IEEE/ACM Transactions on Networking*, vol. 8, no. 1, pp. 16–30, 2000. A preliminary version was presented at *ACM SIGCOMM '98*.

A Proof of Theorem 1

We begin by proving the following lemma, which states 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,*

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

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. The idea of the proof is essentially the same as in the proof of Lemma 2 in Appendix B of [2], where the case of $Q = (q_{\text{exec}}, q_{\text{send}}, 0, 0, 0, q_{\text{test}})$ and $Q' = (q_{\text{exec}}, q_{\text{send}}, q_{\text{test}} - 1, 0, 0, 1)$ was considered.

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 $\mathsf{Exp}_P^{\mathsf{real}}(\mathcal{A})$ minus the probability that $\mathcal{A}$ outputs 1 in the random experiment $\mathsf{Exp}_P^{\mathsf{rand}}(\mathcal{A})$ (see the security definition in Section 2.2). Namely,

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

The proof proceeds by a standard hybrid argument [27]. Consider a sequence of $q_{\text{test}} + 1$ hybrid experiments $\mathsf{Exp}_P^i(\mathcal{A})$, $0 \leq i \leq q_{\text{test}}$, where each $\mathsf{Exp}_P^i(\mathcal{A})$ is defined as follows.

Experiment $\mathsf{Exp}_P^i(\mathcal{A})$:

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 i 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 $\mathsf{Exp}_P^0(\mathcal{A})$ and $\mathsf{Exp}_P^{q_{\text{test}}}(\mathcal{A})$ at the extremes of the sequence are identical to $\mathsf{Exp}_P^{\mathsf{real}}(\mathcal{A})$ and $\mathsf{Exp}_P^{\mathsf{rand}}(\mathcal{A})$, respectively. Notice that as we move from $\mathsf{Exp}_P^{i-1}(\mathcal{A})$ to $\mathsf{Exp}_P^i(\mathcal{A})$ in the sequence, we change the response of i-th Test query from the real session key to a random session key. Since there are q_{test} such moves from $\mathsf{Exp}_P^{\mathsf{real}}(\mathcal{A})$ to $\mathsf{Exp}_P^{\mathsf{rand}}(\mathcal{A})$, 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 $\mathsf{Exp}_P^{i-1}(\mathcal{A})$ and $\mathsf{Exp}_P^i(\mathcal{A})$ is at most $\mathsf{Adv}_P(t, Q')$, 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 remains to show that for every $1 \le i \le q_{\text{test}}$,

$$\left|\Pr[\mathsf{Exp}_P^{i-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Exp}_P^i(\mathcal{A}) = 1]\right| \le \mathsf{Adv}_P(t, Q'). \tag{1}$$

For this purpose, let $\varepsilon = |\Pr[\mathsf{Exp}_P^{i-1}(\mathcal{A}) = 1] - \Pr[\mathsf{Exp}_P^i(\mathcal{A}) = 1]|$. Then, using the adversary $\mathcal{A}$, we construct an adversary $\mathcal{A}_i$ attacking the protocol P, with advantage ε, time complexity t, and query complexity $Q' = (q_{\text{exec}}, 0, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$. $\mathcal{A}_i$ 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. $\mathcal{A}_i$ answers the oracle queries of $\mathcal{A}$ as follows:

- When $\mathcal{A}$ asks a query to the Execute, Reveal, Corrupt, or Dump oracle, $\mathcal{A}_i$ 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 $i - 1$ Test queries, $\mathcal{A}_i$ answers them with a random session key.
  - On the i-th Test query, $\mathcal{A}_i$ asks a query to its own Test oracle and returns the result it receives.
  - For all the remaining Test queries, $\mathcal{A}_i$ answers them with the real session key by accessing its own Reveal oracle.

It is easy to see that $\mathcal{A}_i$ has time complexity t and query complexity at most $Q' = (q_{\text{exec}}, 0, q_{\text{reve}} + q_{\text{test}} - 1, q_{\text{corr}}, q_{\text{dump}}, 1)$.

To quantify the advantage of $\mathcal{A}_i$, it now suffices to notice the following two facts:

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

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

Remark 4. The simulation in the proof of Lemma 1 fails if the adversary $\mathcal{A}$ is allowed to ask the query $\mathsf{Test}(\Pi_U^i)$ when some partner of Π_U^i has already been tested. It is quite easy to see why this is true. Let Π_U^i and $\Pi_{U'}^j$ be the two instances which are partnered together. Suppose that $\mathcal{A}$ asked the query $\mathsf{Test}(\Pi_U^i)$ before its i-th Test query, and asked $\mathsf{Test}(\Pi_{U'}^j)$ after the i-th Test query. $\mathcal{A}_i$ then would simulate the Test oracle by returning a random session key in response to $\mathsf{Test}(\Pi_U^i)$ and by returning the real session key in response to $\mathsf{Test}(\Pi_{U'}^j)$. The simulation is clearly not correct. This scenario is indeed feasible for the simulation given in the proof of Lemma 2 of [2]. Therefore, we suggest that the ROR model of [2] should be fixed to disallow the adversary to access the Test oracle more than once for each set of partners.

Corollary 1. *For any key exchange protocol P,*

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

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. □

A.1 The Basis Step

We prove here the induction basis for the proof of Theorem 1. Let SKE_i be the protocol as defined in Section 3.2. 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:*

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

where $t' = t + O(q_{\text{exec}}t_{\text{SKE}_1})$ 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*

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

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 $1/2 + \epsilon$. Then we construct from $\mathcal{A}_1$ a distinguisher $\mathcal{D}$ that solves the DDH problem for $\mathbb{G}$ with probability ϵ/q_{exec}.

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

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

and

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

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:*

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

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 following distribution Dist_1:

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

Then $\mathcal{D}''$ runs $\mathcal{D}'(\mathsf{T},\, K)$ 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 $(\mathsf{T},\, K)$ according to Dist_1. Note that if $n \in \{2, 3\}$, then generating $(\mathsf{T},\, K)$ according to Dist_1 requires $\mathcal{D}''$ to perform only a constant number of exponentiations in $\mathbb{G}$.

If $(g^{s_1}, g^{r_2}, g^{s_1' r_2})$ is a true Diffie-Hellman triple (i.e., $s_1 = s_1'$), then we have $\mathsf{Dist}_1 \equiv \mathsf{Real}_1$ 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 $\mathsf{Dist}_1 \equiv \mathsf{Fake}_1$. 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 $(\mathsf{T},\, K)$ 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 $(\mathsf{T},\, K)$ 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:*

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

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:

$$\begin{aligned} \log_g y_2 &= a_1 - a_2, \\ \log_g y_3 &= a_1 - a_3, \\ &\vdots \\ \log_g y_n &= a_1 - a_n. \end{aligned}$$

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 γ-th $\mathsf{Execute}$ 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 $\mathsf{Corrupt}$ 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 δ-th $\mathsf{Execute}$ query, $\mathcal{D}$ answers them in the natural way by executing protocol SKE_1 on its own. When $\mathcal{A}_1$ asks the δ-th $\mathsf{Execute}$ 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 $(g^{s_1}, g^{r_2}, g^{s_1' r_2}) \in \mathbb{G}^3$, $\mathcal{D}$ generates (T, K) according to the distribution Dist_1 and answers the δ-th $\mathsf{Execute}$ 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 $t' = t + O(q_{\mathrm{exec}} t_{\mathrm{SKE}_1})$.

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 δ-th $\mathsf{Execute}$ query; this happens with probability $1/q_{\mathrm{exec}}$. If $(g^{s_1}, g^{r_2}, g^{s_1' r_2})$ is a true Diffie-Hellman triple, then, by Claim 1, $\mathsf{Dist}_1 \equiv \mathsf{Real}_1$ and thus, by assumption, $\Pr[b = b'] = 1/2 + \epsilon$. So, the probability that $\mathcal{D}$ outputs 1 on a true Diffie-Hellman triple is also $1/2 + \epsilon$. If instead $(g^{s_1}, g^{r_2}, g^{s_1' r_2})$ is a random triple, then $\mathsf{Dist}_1 \equiv \mathsf{Fake}_1$ 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 $\Pr[\delta = \gamma] = 1/q_{\mathrm{exec}}$, we obtain $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{D}) = \epsilon/q_{\mathrm{exec}}$. Finally, since $\mathsf{Adv}_{\mathrm{SKE}_1}(\mathcal{A}_1) = 2\epsilon$ and $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{D}) \leq \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t')$ by definitions, it follows that $\mathsf{Adv}_{\mathrm{SKE}_1}(\mathcal{A}_1) \leq 2q_{\mathrm{exec}} \cdot \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t')$. This completes the proof of Lemma 2. ■

A.2 The Induction Step

We now claim that for each $h \geq 1$, 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.,* $2^{h+1} \leq n < 2^{h+2}$*). Then we have:*

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

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:*

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

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 3 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 $\mathsf{T}_{h,i}$ denote the transcript of protocol SKE_h executed by subgroup $\mathcal{G}_i$ to generate the subgroup key k_i. Then we write $(\mathsf{T}_{h,i}, k_i) \leftarrow \mathsf{Real}_h$ to denote the generation of a transcript/key pair $(\mathsf{T}_{h,i}, k_i)$ through a real execution of SKE_h. We also write $(\mathsf{T}_{h,i}, a_i) \leftarrow \mathsf{Rand}_h$ to denote the generation of $(\mathsf{T}_{h,i},\ a_i)$ where $\mathsf{T}_{h,i}$ is generated by a real execution of SKE_h and a_i is a random key chosen independently of $\mathsf{T}_{h,i}$ but chosen uniformly from $\mathbb{G}$.

With these notations, we now introduce the following two distributions:

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

and

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

The distribution Real_{h+1}, when either $l = m = 3$ (i.e., $n > 5$) or $l = 2$ and $m = 3$ (i.e., $n = \{4, 5\}$), 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 $m = \{3, 4, \ldots\}$, 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 now claim that distinguishing between two distributions Real_{h+1} and Rand_{h+1} is no easier than breaking the security of protocol SKE_h.

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:

$$\left|\Pr[\mathcal{D}(\mathsf{T}, K) = 1 \mid (\mathsf{T}, K) \leftarrow \mathsf{Real}_{h+1}] - \Pr[\mathcal{D}(\mathsf{T}, K) = 1 \mid (\mathsf{T}, K) \leftarrow \mathsf{Rand}_{h+1}]\right| \leq \mathsf{Adv}_{\mathrm{SKE}_h}(t', Q_h),$$

where $t' = t + O(t_{\exp})$ *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 $\mathsf{T}_{h,2}$, $\mathsf{T}_{h,3}$, …, $\mathsf{T}_{h,l}$ by making an $\mathsf{Execute}$ query for each of the subgroups $\mathcal{G}_2, \mathcal{G}_3, \ldots, \mathcal{G}_l$. Let $\Pi_{U \in \mathcal{G}_i}$ denote any instance activated by the $\mathsf{Execute}$ query directed to $\mathcal{G}_i$. Next, $\mathcal{A}'_h$ asks $l-1$ Test queries $\mathsf{Test}(\Pi_{U \in \mathcal{G}_2})$, $\mathsf{Test}(\Pi_{U \in \mathcal{G}_3})$, …, $\mathsf{Test}(\Pi_{U \in \mathcal{G}_l})$; recall that in our model, the adversary is allowed to ask multiple queries to its Test oracle as long as the tested instances are fresh and no two of them are partnered together. Let k'_i be either the real session key or a random session key returned in response to the query $\mathsf{Test}(\Pi_{U \in \mathcal{G}_i})$. We then write $(\mathsf{T}_{h,i}, k'_i) \leftarrow \mathsf{Test}_h$ to denote the above way of generating a transcript/key pair $(\mathsf{T}_{h,i}, k'_i)$.

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 $\mathcal{D}(\mathsf{T}, K)$, and outputs whatever bit $\mathcal{D}$ outputs. Distribution Dist_{h+1} is defined as follows:

$$\mathsf{Dist}_{h+1} \stackrel{\text{def}}{=} \left\{ (\mathsf{T}, K) \;\middle|\; \begin{array}{l} (\mathsf{T}_{h,2}, k'_2), \ldots, (\mathsf{T}_{h,l}, k'_l) \leftarrow \mathsf{Test}_h; \\ s_1, t_1 \in_R \mathbb{Z}_q; \\ r_2 = I(k'_2), \ldots, r_l = I(k'_l); \\ r_{l+1}, \ldots, r_m \in_R \mathbb{Z}_q; \\ w_1 = g^{s_1}; \\ z_2 = g^{r_2}, \ldots, z_m = g^{r_m}; \\ k_1 = g^{s_1 t_1}; \\ x_2 = g^{s_1 r_2}, \ldots, x_m = g^{s_1 r_m}; \\ y_2 = k_1 \cdot x_2^{-1}, \ldots, y_m = k_1 \cdot x_m^{-1}; \\ \mathsf{T} = (\mathsf{T}_{h,2}, \ldots, \mathsf{T}_{h,l}, w_1, z_2, \ldots, z_m, y_2, \ldots, y_m); \\ K = k_1 \end{array} \right\}$$

To generate (T, K) according to Dist_{h+1}, $\mathcal{A}'_h$ performs $O(m)$ exponentiations in $\mathbb{G}$ and makes $l-1$ $\mathsf{Execute}$ queries and $l-1$ Test queries. If we instantiate both l and m with 3, $\mathcal{A}'_h$ has time complexity $t' = t + O(t_{\exp})$ 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 $\mathsf{Dist}_{h+1} \equiv \mathsf{Real}_{h+1}$. On the other hand, if each k'_i is a random session key chosen independently of the transcript $\mathsf{T}_{h,i}$, then $\mathsf{Dist}_{h+1} \equiv \mathsf{Rand}_{h+1}$. 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 Rand_{h+1}.

Thus $\mathsf{Adv}_{\mathrm{SKE}_h}(\mathcal{A}'_h) = |\mu - \nu|$. Since $\mathsf{Adv}_{\mathrm{SKE}_h}(\mathcal{A}'_h) \leq \mathsf{Adv}_{\mathrm{SKE}_h}(t', Q_h)$, 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:*

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

Proof. In distribution Rand_{h+1}, the session key K is completely independent of the set of $l-1$ transcripts $\{\mathsf{T}_{h,i} \mid i \in [2,l]\}$ because each $a_i \in \mathbb{G}$ is chosen at random independently of $\mathsf{T}_{h,i}$. Therefore, if we define Rand'_{h+1} as the distribution derived from Rand_{h+1} by eliminating all the transcripts $\mathsf{T}_{h,2}, \mathsf{T}_{h,3}, \ldots, \mathsf{T}_{h,l}$, it is clear that:

$$\Pr[\mathcal{A}(\mathsf{T}, K_{(b)}) = b \mid (\mathsf{T}, K_{(1)}) \leftarrow \mathsf{Rand}'_{h+1}; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0,1\}] = \Pr[\mathcal{A}(\mathsf{T}, K_{(b)}) = b \mid (\mathsf{T}, K_{(1)}) \leftarrow \mathsf{Rand}_{h+1}; K_{(0)} \leftarrow \mathbb{G}; b \leftarrow \{0,1\}]. \quad (2)$$

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

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

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

Before continuing further, let us define

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

and

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

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 γ-th $\mathsf{Execute}$ 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 δ-th $\mathsf{Execute}$ 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 δ-th $\mathsf{Execute}$ query, $\mathcal{A}_h$ responds by calling its own $\mathsf{Execute}$ 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 $2^{h+1} \leq n < 2^{h+2}$.

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 δ-th $\mathsf{Execute}$ query. If k'_2, k'_3, …, k'_l in Dist_{h+1} are real session keys, then $\mathsf{Dist}_{h+1} \equiv \mathsf{Real}_{h+1}$ and thus, the probability that $\mathcal{A}_{h+1}$ correctly guesses the hidden bit b is $\mathsf{SuccPr}_{h+1}(\mathcal{A}_{h+1})$. Hence, the probability that $\mathcal{A}_h$ outputs 1 when its Test oracle returns actual session keys is also $\mathsf{SuccPr}_{h+1}(\mathcal{A}_{h+1})$. On the other hand, if k'_2, k'_3, …, k'_l in Dist_{h+1} are random session keys, then $\mathsf{Dist}_{h+1} \equiv \mathsf{Rand}_{h+1}$ and thus, by Claim 4, the probability that $\mathcal{A}_{h+1}$ correctly guesses the hidden bit b is $\mathsf{SuccPr}_1(\mathcal{A}_{h+1})$. So, the probability that $\mathcal{A}_h$ outputs 1 when its Test

oracle returns random session keys is also $\mathsf{SuccPr}_1(\mathcal{A}_{h+1})$. Therefore since $\Pr[\delta = \gamma] = 1/q_{\text{exec}}$, we obtain:

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

Note that this equation already implies that $|\mathsf{SuccPr}_{h+1}(\mathcal{A}_{h+1}) - \mathsf{SuccPr}_1(\mathcal{A}_{h+1})|$ is negligible and so $\mathsf{SuccPr}_{h+1}(\mathcal{A}_{h+1})$ 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 $\mathsf{Adv}_{\text{SKE}_{h+1}}(\mathcal{A}_{h+1}) = |2 \cdot \mathsf{SuccPr}_{h+1}(\mathcal{A}_{h+1}) - 1|$, we easily have:

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

From this, the following is immediate:

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

This completes the proof of Lemma 3. ■

B Proof of Theorem 2

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 construct 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 one same nonce is used by a user for two different instances, one activated by a Send query and the other activated by either an $\mathsf{Execute}$ or a Send query.
- Forge: The event that $\mathcal{A}^+$ outputs a valid forgery with respect to the public key PK_U of some user $U \in \mathcal{U}$ before making the query $\mathsf{Corrupt}(U)$. Let $d \in \{1, 2\}$ and let $*$ denote any message string. Then, more formally, Forge is the event that $\mathcal{A}^+$ makes a query of the form $\mathsf{Send}(\Pi_V^i, U\|d\| * \|\sigma_U^d)$ such that $\mathsf{Vrfy}_{PK_U}(U\|d\| * \|\mathsf{sid}_V^i\|\mathsf{ncs}_V^i, \sigma_U^d) = 1$ and σ_U^d was not previously output by any instance of user U as a signature on $U\|d\| * \|\mathsf{sid}_V^i\|\mathsf{ncs}_V^i$.

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

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

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

Lemma 4. $\Pr[\mathsf{Forge}] \le |\mathcal{U}| \cdot \mathsf{Succ}_{\Sigma}(t')$, *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 $\mathsf{Vrfy}_{PK}(M, \sigma) = 1$ 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 $U' \in \mathcal{U}$ and setting $PK_{U'}$ to PK. For all other users in $\mathcal{U}$, $\mathcal{F}$ honestly generates a verification/signing key pair by running the key generation algorithm $\mathsf{Kgen}(1^\kappa)$. $\mathcal{F}$ 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 $\mathsf{Corrupt}$ queries. In this latter case, $\mathcal{F}$ proceeds as follows:

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

The simulation provided above is perfect unless adversary $\mathcal{A}^+$ makes the query $\mathsf{Corrupt}(U')$.

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 $\mathsf{Vrfy}_{PK_{U'}}(M, \sigma) = 1$ and σ was not previously output by any instance of U' 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 $\Pr[\mathsf{Forge}]/|\mathcal{U}|$. □

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 $\mathsf{Kgen}(1^\kappa)$ to generate a verification/signing key pair (PK_U, SK_U) for each $U \in \mathcal{U}$, $\mathcal{A}$ 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 $\mathsf{Execute}(\mathsf{sid}, \mathsf{gid})$, $\mathcal{A}$ sends the same query to its own $\mathsf{Execute}$ 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 $\tilde{k}_U \in \mathbb{G}$ for all $U \in \mathsf{gid}$,
2. setting $\mathsf{ncs} = \{(U, \tilde{k}_U) \mid U \in \mathsf{gid}\}$,
3. signing the messages in T as specified in SKE$^+$ (more concretely, replacing each message $M_U^d = U\|d\|*$ in T with $\tilde{M}_U^d = U\|d\| * \|\sigma_U^d$, where $\sigma_U^d = \mathsf{Sign}_{SK_U}(U\|d\| * \|\mathsf{sid}\|\mathsf{ncs})$),
4. and prepending $\{U\|0\|\tilde{k}_U\}_{U \in \mathsf{gid}}$ to the signed transcript.

Then $\mathcal{A}$ returns the patched transcript T^+ in response to the $\mathsf{Execute}$ query of $\mathcal{A}^+$ and adds the pair $(\mathsf{ncs}, \mathsf{T})$ 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 $\mathsf{Send}(\Pi_U^i, \mathsf{sid}\|\mathsf{gid})$, $\mathcal{A}$ sets $\mathsf{sid}_U^i = \mathsf{sid}$ and $\mathsf{gid}_U^i = \mathsf{gid}$, chooses a random $\tilde{k}_U^i \in \mathbb{G}$, and returns $U\|0\|\tilde{k}_U^i$ to $\mathcal{A}^+$. If the query is of the form $\mathsf{Send}(\Pi_U^i, V\|0\|\tilde{k}_V)$ for some $V \in \mathsf{gid}_U^i$, then there are the following two cases.

- If $\tilde{k}_V$ is not the last nonce that Π_U^i is expected to receive, $\mathcal{A}$ simply waits for the next nonce from other users in gid_U^i.
- Otherwise, $\mathcal{A}$ defines ncs_U^i as specified in the protocol and replies to $\mathcal{A}^+$ with an appropriate message generated according to the general instruction (given below) for answering Send queries of $\mathcal{A}^+$.

The general instruction for answering query $\mathsf{Send}(\Pi_U^i, M)$:

$\mathcal{A}$ checks the list NTList to see if there exists an entry of the form $(\mathsf{ncs}_U^i, \mathsf{T})$. If so, then $\mathcal{A}$ generates the message $\tilde{M}_U^d = U\|d\| * \|\sigma_U^d$ from the appropriate message $M_U^d = U\|d\|*$ 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 $\mathsf{Execute}(\mathsf{sid}_U^i, \mathsf{gid}_U^i)$ to its own oracle, then proceeds as in the former case, and finally adds the pair $(\mathsf{ncs}_U^i, \mathsf{T})$ to the list NTList for future use.

For all other Send queries (that is, the queries of the form $\mathsf{Send}(\Pi_U^i, V\|d\| * \|\sigma_V)$) 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_U^i has been asked a $\mathsf{Corrupt}$ query before the Send query, $\mathcal{A}$ responds by following the general instruction above.
- On the contrary, suppose that some user in gid_U^i has been asked a $\mathsf{Corrupt}$ query before the Send query. In this case, $\mathcal{A}$ may has to simulate the action of Π_U^i without recourse to a fixed transcript obtained from its $\mathsf{Execute}$ oracle, 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 $(\mathsf{ncs}_U^i, \mathsf{T})$ in NTList and makes a Dump query to the U's instance activated by the $\mathsf{Execute}$ 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 Π_U^i perfectly, following the protocol exactly.

Dump Queries. Upon receiving the query $\mathsf{Dump}(\Pi_U^i)$, $\mathcal{A}$ first finds an entry $(\mathsf{ncs}_U^i, \mathsf{T})$ in list NTList. $\mathcal{A}$ then makes a Dump query to the U's instance activated by the $\mathsf{Execute}$ 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 $\mathsf{Corrupt}(U)$ by returning the long-term signing key SK_U.

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

- If either the query $\mathsf{Dump}(\Pi_U^i)$ was asked or some user in gid_U^i was corrupted before Π_U^i 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 following general instruction for answering Reveal and Test queries of $\mathcal{A}^+$.

The general instruction for answering $\mathsf{Reveal}(\Pi_U^i)$ and $\mathsf{Test}(\Pi_U^i)$:

$\mathcal{A}$ finds an entry $(\mathsf{ncs}_U^i, \mathsf{T}) \in \mathsf{NTList}$, asks the same query (either Reveal or Test query) to the U's instance activated by the $\mathsf{Execute}$ query that resulted in T, and receives in return a key K_{SKE} for protocol SKE. Let $\mathsf{gid}_U^i = \{U_1, U_2, \ldots, U_n\}$ with U_1 being the user at the root node. Then using K_{SKE}, $\mathcal{A}$ computes a key K as specified in protocol SKE^+, namely as $K = H(K_{\mathrm{SKE}}\|\tilde{k}_2\|\tilde{k}_3\|\cdots\|\tilde{k}_n)$. Finally, $\mathcal{A}$ returns the key K to $\mathcal{A}^+$.

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

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, consider the simulation described above for Test queries. If K_{SKE} is a real session key for protocol SKE, then K is also a real session key for protocol SKE$^+$. If instead K_{SKE} is random, then so is K since H is a random oracle. Let $\mathsf{Succ}_{\mathcal{A}}$ (resp. $\mathsf{Succ}_{\mathcal{A}^+}$) 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:

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

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

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

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

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

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

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

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

From this, the following is immediate:

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

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

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

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

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

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