Constant-Round Authenticated Group Key Exchange with Logarithmic Computation Complexity*

Junghyun Nam[1], Juryon Paik[2], Ung Mo Kim[2], and Dongho Won[2,**]

[1] Department of Computer Science, Konkuk University, Korea
jhnam@kku.ac.kr
[2] Department of Computer Engineering, Sungkyunkwan University, Korea
juryon.paik@gmail.com, umkim@ece.skku.ac.kr, 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 constant-round authenticated GKE protocols providing forward secrecy thus far are not fully scalable, but have a computation complexity that scales only linearly in group size. Motivated by this observation, we propose a new and the first forward-secure authenticated GKE protocol that achieves both constant round complexity and logarithmic computation complexity. In particular, our GKE protocol is fully scalable in all key metrics when considered in the context of a broadcast network. The scalability of the protocol is achieved by using a complete binary tree structure combined with a so-called "nonce-chained authentication technique". Besides its scalability, our protocol features provable security against active adversaries under the decisional Diffie-Hellman assumption. We provide a rigorous proof of security for the protocol in a well-defined formal model of communication and adversary capabilities. The result of the current work means that forward-secure generation of session keys even for very large groups can be now done both securely and efficiently.

Keywords: Cryptography, group key exchange, scalability, binary tree, nonce-chained authentication, provable security.

1 Introduction

The primary goal of cryptography is to provide a means for communicating confidentially and with integrity over a public channel. Roughly speaking,

* This work was supported by the Korean Ministry of Information and Communication under the Information Technology Research Center (ITRC) support program supervised by the Institute of Information Technology Assessment (IITA).

** Corresponding author.

J. Katz and M. Yung (Eds.): ACNS 2007, LNCS 4521, pp. 158–176, 2007.
© Springer-Verlag Berlin Heidelberg 2007

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*. Typically, 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., [19,24,37,8,25] 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 [16] 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 [6,7,40,5,11,15,2,28] 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 [27,11,29,2,33,17].

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 both sent and received per user in the protocol, and (3) the *bit complexity*, the maximum number of bits (i.e., the maximum combined length of messages) both sent and received per user in the protocol. In order for a GKE protocol to be scalable, it is desirable in many real-life applications that the protocol be able to complete in a constant number of rounds. The computation cost of a protocol is directly related to the *computation complexity* which we define as the maximum amount of computation done by a single user in the protocol.

By computation, we do not mean simple traverses of the identities of the protocol participants, but mean *any kinds of cryptographic operations* such as public-key and symmetric-key operations, modular arithmetic operations, hash function evaluations, etc. Although the above definitions of various complexities are largely based on those given in the full version of [29][1], there is a noteworthy difference in defining message and bit complexities. Our definitions for these complexities counts both the sent and received traffics whereas those in the full version of [29] considers only the sent one. We believe this modification provides a more accurate way to measure the communication efficiency of any distributed protocols.

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 [11] even with round complexity $O(n)$ in a group of size n. Still up to now, there are only a very limited number of constant-round protocols [9,29,30,21] carrying a claimed proof of security against active adversaries in a formal model. However, all these constant-round protocols suffer from the number of public-key operations that scales linearly in group size, and thus exhibit $O(n)$ computation complexity under the definition above. These best-known protocols are categorized as key agreement protocols, but the situation is not much different for authenticated key transport protocols [23,35,26]. Indeed, we are unaware of any, provably secure or not, authenticated GKE protocols achieving both constant round complexity and logarithmic computation complexity. The protocols of [23,35,9,26] requires one distinct user to perform $O(n)$ modular exponentiations or public-key encryptions. The other protocols from [29,30,21] is all a novel extension of the protocol (i.e., protocol 3) by Burmester and Desmedt [12], but commonly require each user to perform $O(n)$ signature verifications. For moderate size groups, these previous solutions are clearly appealing. But for large groups, many applications will likely demand a protocol whose computation complexity scales logarithmically with group size. It is this observation that prompted the present work aimed at designing an authenticated GKE protocol which scales very well for large groups.

Contribution. The result of this work is the first forward-secure authenticated GKE protocol that achieves $O(1)$ round complexity and $O(\log n)$ computation complexity. In Tables 1 and 2, we summarize the computation and communication requirements of our protocol and other authenticated GKE protocols [23,35,9,29].[2] (By the tables, we are not arguing that one is overall superior to another, but meant to provide an asymptotic analysis for comparing scalability of different protocols.) Like the protocols of [23,35], our GKE protocol is categorized as a key transport protocol. The protocol of [9][3] features optimal

[1] The full version of [29] is available at `http://www.cs.umd.edu/~jkatz`

[2] Although the protocols from [30,21] may perform better in practice than the protocol of [29], they fall into the same category from the computation complexity perspective.

[3] We refer to [17] for a security enhancement to this protocol.

Table 1. Computation requirements of authenticated GKE protocols

	Exp	Sig/Dec	Ver/Enc	Div	Mul
Boyd-Nieto [9]		$O(1)/$	$/O(n)$		
Katz-Yung [29]	$O(1)$	$O(1)/$	$O(n)/$	$O(1)$	$O(n \log n)$
Hirose-Yoshida [23]	$O(n)$	$O(n)/$	$O(n)/$		$O(n)$
Mayer-Yung [35]		$O(1)/$	$O(n)/O(n)$		
Here	$O(\log n)$	$O(1)/$	$O(\log n)/$	$O(1)$	$O(\log n)$

Note. "Mayer-Yung [35]" refers to *a-MKT with Consistency 1* of [35].
Exp: the maximum number of modular exponentiations performed per user.
Sig/Dec: the maximum numbers of signature generations and public-key decryptions performed per user.
Ver/Enc: the maximum numbers of signature verifications and public-key encryptions performed per user.
Div: the maximum number of modular divisions performed per user.
Mul: the maximum number of modular multiplications performed per user.

Table 2. Communication requirements of authenticated GKE protocols

	Rounds	Messages		Bits	
		PtP	Broadcast	PtP	Broadcast
Boyd-Nieto [9]	1	$O(n)$	$O(n)$	$O(n^2)$	$O(n)$
Katz-Yung [29]	3	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Hirose-Yoshida [23]	3	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Mayer-Yung [35]	4	$O(n)$	$O(n)$	$O(n^2)$	$O(n)$
Here	3	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$

Note. "Mayer-Yung [35]" refers to *a-MKT with Consistency 1* of [35].
Rounds: the number of communication rounds required to complete the protocol.
Messages: the maximum number of messages both sent and received per user.
Bits: the maximum number of bits both sent and received per user.
PtP: the point-to-point network model.
Broadcast: the broadcast network model.

round complexity [4], but lacks perfect forward secrecy [20]. As Table 1 shows, the maximum computation rate per user is bounded by $O(\log n)$ in our protocol, whereas this rate per user rises up to $O(n)$ in the other protocols. Thus from a theoretical point of view, our main contribution is to show the possibility of achieving logarithmic computation complexity in constructing forward-secure constant-round protocols for authenticated group key exchange. However, it is also important from a practical viewpoint to notice that for reasonable values of n, the actual computation in our protocol can be heavier than that in the other protocols.

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

bit complexities.) Thus if we assume a broadcast network, our protocol can be regarded as the first forward-secure authenticated GKE protocol that not only achieves $O(1)$ round complexity but also bounds all other complexities (i.e., bit, message, and computation complexities) by $O(\log n)$.

Furthermore, our protocol is provably secure against a powerful active adversary under the decisional Diffie-Hellman assumption. We provide a rigorous proof of security for the protocol in a refinement of the standard security model [11,9,29,30,21]. From the standpoint of the adversary's capabilities, our security model is a unique combination of previous results from [11,10,2,36], 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 [2]. 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 [18]. 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. [41] and Wong et al. [42] 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., [39]) 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. [31,32] to the forward-secure case. Their protocols require no secure channels of any kind and offer distributed functionality. Later, Lee et al. [34] present a pairing-based variant of the TGDH protocol of [31]. All these works [31,32,34], however, provide no explicit treatment of key exchange for initial group formation, focusing only on key updates upon group membership changes.

Ren et al. [38] 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. [3] and Dutta et al. [22] construct their protocols by combining a ternary tree structure with the one-round tripartite protocol of Joux [25]. Back in 1994, Burmester and Desmedt [12] also proposed a tree-based GKE protocol. This protocol (i.e., protocol 2 of [12]) seems to be the first GKE protocol utilizing a binary tree structure, and differs from all other protocols mentioned above in that there exists a bijective mapping between protocol participants and tree nodes. But, this protocol, in common with other protocols from [3,38,22], 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 [13] presented a graph-based protocol called CKDS. The CKDS protocol (more precisely, the multicast version of CKDS) has a potential to achieve the same level of complexities as our protocol, in the sense that the minimum spanning tree of the graph it used could have a height of $O(\log n)$. But unlike our provably-authenticated protocol, this protocol assumes a passive adversary and justifies its security on purely heuristic grounds without providing no formal analysis of security.

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.

2.1 Communication and Adversary 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_i^π to denote the π-th instance of user U_i. Before the protocol is executed for the first time, each user $U_i \in \mathcal{U}$ creates a long-term public/private key pair (PK_i, SK_i) 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 works, 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 [14,15,28], we assume that session IDs are assigned and provided by some higher-level protocol. While this assumption is unnecessary in some protocols [9,29] 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. Supporting this assumption, Katz and Shin [28] have recently made an interesting observation: *since a user may be running many instances of a key exchange protocol concurrently, users in practice need a means to identify the sessions to which incoming messages belong. Therefore, in some sense, predefined session IDs are implicit even in the models [11,9,29] that use a customized*

definition of session IDs. 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_i^π and gid_i^π to denote respectively sid and gid provided to instance Π_i^π. Note that gid_i^π 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 $\mathcal{U}$. However, there is an important point regarding the generation of session IDs. Our model does not require the adversary to be honest in generating session IDs. This means that the adversary may try to replay a session ID as many times as necessary for its attack, but only at its own risk. In other words, the uniqueness of a session ID is not guaranteed by the model but should be checked by users themselves.

An instance is said to *accept* when it successfully computes a session key in a protocol execution. Let acc_i^π be a boolean variable that evaluates to TRUE if Π_i^π has accepted, and FALSE otherwise. We say that any two instances Π_i^π and Π_j^ω are *partners* of each other, or equivalently, *partnered* iff all the following three conditions are satisfied: (1) $\mathsf{sid}_i^\pi = \mathsf{sid}_j^\omega$, (2) $\mathsf{gid}_i^\pi = \mathsf{gid}_j^\omega$, and (3) $\mathsf{acc}_i^\pi = \mathsf{acc}_j^\omega =$ TRUE. We also say that two instances Π_i^π and Π_j^ω are *potential partners* of each other, or equivalently, *potentially partnered* iff the first two conditions above hold. We use pid_i^π and ppid_i^π to denote respectively the partners and the potential partners of the instance Π_i^π. Then it follows by the definitions that $\mathsf{pid}_i^\pi \subseteq \mathsf{ppid}_i^\pi$.

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

[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.

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 [36] 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_i^π, M): This query sends message M to instance Π_i^π. The instance Π_i^π 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_i^π, sid$\|$gid) prompts Π_i^π to initiate an execution of the protocol using session ID sid and group ID gid.
- Reveal(Π_i^π)[6]: This query returns to the adversary the session key held by Π_i^π. This oracle call captures the idea that exposure of some session keys should not affect the security of other session keys [18]. The adversary is not allowed to ask this query if it has already queried Test(Π_j^ω) for some Π_j^ω in pid_i^π (see below for the description of the Test oracle).
- Corrupt(U_i): This query returns to the adversary all long-term secret information of U_i including the private key SK_i[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_i is currently executing the protocol or not. This oracle call captures the idea that damage due to loss of U_i's long-term secrets should be restricted to those sessions where U_i will participate in the future.
- Dump(Π_i^π): This query returns *all short-term secrets* used in the past or currently being used by instance Π_i^π[8]. But, neither the session key computed by Π_i^π nor any long-term secrets of U_i are not returned. This models the adversary's capability to embed a Trojan horse or other form of malicious

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

[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 enable 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.

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 $\mathsf{Test}(\Pi_j^\omega)$ for some Π_j^ω in ppid_i^π.

– $\mathsf{Test}(\Pi_i^\pi)$: 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_i^π 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_i^π is *fresh* (see Section 2.2 for the definition of freshness). 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 the model of Canetti and Krawczyk [14]. But as noted in [36], 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. *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 1, 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 Security Definition and Assumptions

Freshness. The notion of *freshness* is used in the definition of security to prohibit the adversary from asking the Test query against an instance whose session key (or some information about the key) can be exposed by trivial means.

Definition 2. *The instance Π_i^π is considered unfresh iff any of the following conditions hold:*

1. $\mathsf{acc}_i^\pi = \text{TRUE}$.
2. *The adversary queried* $\mathsf{Corrupt}(U_j)$ *for some* U_j *in* gid_i^π *before some instance in* ppid_i^π *accepts.*
3. *The adversary queried* $\mathsf{Dump}(\Pi_j^\omega)$ *for some* Π_j^ω *in* ppid_i^π.
4. *The adversary queried* $\mathsf{Reveal}(\Pi_j^\omega)$ *or* $\mathsf{Test}(\Pi_j^\omega)$ *for some* Π_j^ω *in* pid_i^π.

All other instances are considered fresh.

Remark 2. By "$\mathsf{Test}(\Pi_j^\omega)$" in the fourth condition of Definition 2, 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 the instances partnered together) is no different than obtaining it once.

Security. 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 any allowed oracle queries at will as many times as it wishes.
- **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 $\mathsf{Test}(\Pi_i^\pi)$ asked, the adversary is prohibited from querying $\mathsf{Dump}(\Pi_j^\omega)$ for some $\Pi_j^\omega \in \mathsf{ppid}_i^\pi$ and from querying $\mathsf{Reveal}(\Pi_j^\omega)$ for some $\Pi_j^\omega \in \mathsf{pid}_i^\pi$. This restriction reflects the fact that the adversary can win the game unfairly by using the information obtained via the query $\mathsf{Dump}(\Pi_j^\omega)$ or $\mathsf{Reveal}(\Pi_j^\omega)$.

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 $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{D}) = |\Pr[\mathcal{D}(\mathbb{G}, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{D}(\mathbb{G}, g, g^x, g^y, g^z) = 1]|$. We say that the DDH assumption holds in $\mathbb{G}$ (or equivalently, the DDH problem is hard in $\mathbb{G}$) if $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{D})$ is negligible for all probabilistic polynomial time algorithms $\mathcal{D}$. We denote by $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(t)$ the maximum value of $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G}}(\mathcal{D})$ 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. 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 GKE protocol with round complexity $O(1)$ and computation complexity $O(\log n)$. Towards the goal, we arrange the users in a complete binary tree where all the levels, except perhaps the last, are full; while on the last level, any missing nodes are to the right of all the nodes that are present. Fig. 1 shows an example of a complete binary tree of height 3 with 6 leaves and 6 internal nodes. Users in $\mathcal{G}$ are placed at nodes in a straightforward way that U_i has U_{2i} as its left child and U_{2i+1} as its right child. Let N_i denote the node at which U_i is positioned and let $\mathcal{G}_i$ denote the subgroup consisting of all users located in the subtree rooted at node N_i. Each internal node N_i is associated with a node key k_i. In the protocol, the node key k_i is first generated by U_i and then shared as the subgroup key among the users in $\mathcal{G}_i$. Accordingly, k_1 serves as the group key (i.e., session key) shared by all users in $\mathcal{G}$.

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 of $\mathbb{G}$, and (3) a function I mapping elements of $\mathbb{G}$ to elements of $\mathbb{Z}_q$. A standard

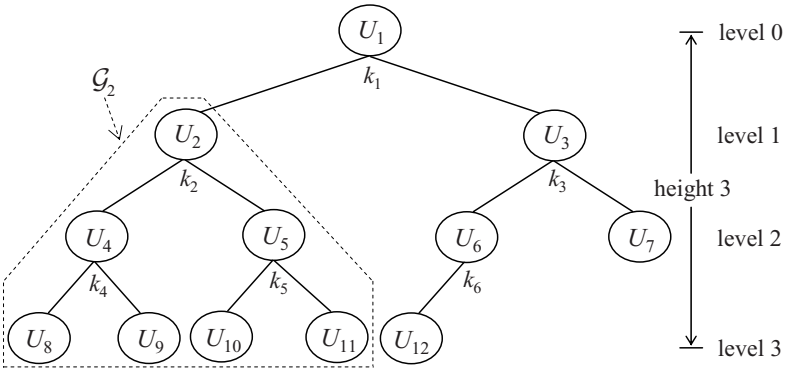

Fig. 1. A complete binary tree for $\mathcal{G} = \{U_1, \ldots, U_{12}\}$

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

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

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

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

3.2 Security Result for Protocol SKE

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

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.*

At a high level, the proof of Theorem 1 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. 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 actual proof of the theorem is omitted here due to lack of space, and will be given in the full version of this paper.

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

4.1 Description of SKE^+

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

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

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

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

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

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

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

- When 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 $\mathsf{Vrfy}_{PK_j}(U_j \| 1 \| z_j \| \mathsf{sid}_i \| \mathsf{ncs}_i \| \mathsf{ncs}_{\varphi(i)} \| \cdots \| \mathsf{ncs}_1, \sigma_j^1) = 1$. If any of the verifications fail, U_i aborts the protocol without accepting (i.e., without computing a session key). Otherwise, U_i computes x_{2i}, y_{2i}, x_{2i+1}, y_{2i+1} and w_i as in protocol SKE, generates a signature $\sigma_i^2 = \mathsf{Sign}_{SK_i}(U_i \| 2 \| w_i \| y_{2i} \| y_{2i+1} \| \ \mathsf{sid}_i \| \mathsf{ncs}_i \| \mathsf{ncs}_{\varphi(i)} \| \cdots \| \mathsf{ncs}_1)$, 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 $\mathsf{Vrfy}_{PK_j}(U_j \| 2 \| w_j \| y_{2j} \| y_{2j+1} \| \mathsf{sid}_i \| \mathsf{ncs}_j \| \mathsf{ncs}_{\varphi(j)} \| \cdots \| \mathsf{ncs}_1, \sigma_j^2) = 1$. If any of the verifications fail, U_i terminates without accepting. Otherwise, U_i derives the root node key k_1 as in SKE and sets the session key K equal to k_1.

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

protocol, the message and bit complexities of the resulting protocol rise at least up to $O(n)$ because the compiler requires each user to receive nonces from all other users.

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

Our transformation, though similar in spirit as that by the KY compiler, reduces the number of nonces to be received per user to the order of $\log n$ while achieving provable security of the protocol SKE$^+$. The two main observations underlying this result are that: (1) at a given point of time, each pre-defined session ID is unique for all concurrent runs of SKE$^+$ (see Section 2.1 for the justification of pre-defined session IDs) and (2) even with at most $O(\log n)$ nonces per user, SKE$^+$ is able to guarantee the freshness of the messages exchanged among users. The first observation is clear from the description of SKE$^+$; no two active instances of a user possess a same session ID. The second observation becomes quite obvious once we notice that there is a chain of nonces in SKE$^+$: for all $2 \leq i \leq n$, two ordered sequences ncs_i and $\mathsf{ncs}_{\varphi(i)}$ are linked by the common element ϕ_i. This chain of nonces enables each user U_i to verify the freshness of all messages from its ancestors, even when those messages are not signed with ϕ_i. In other words, the use of the nonce chain ensures that no singed message is replayed between two sessions even with a same session ID. We call this technique *nonce-chained authentication*. These two observations, taken together, suggest that a pre-defined session ID combined with the nonce-chained authentication technique serves as a unique session identifier for all time points and thereby obviates the need for each user to receive n nonces.

4.2 Security Result for Protocol SKE$^+$

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

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

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

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.*

Proof Idea. We can prove the theorem by finding a reduction from the security of protocol SKE^+ to the security of protocol SKE. Assuming an active adversary $\mathcal{A}^+$ who attacks protocol SKE^+, we construct a passive adversary $\mathcal{A}$ that uses $\mathcal{A}^+$ in its attack on SKE. As in a typical reductionist approach, the adversary $\mathcal{A}$ simply runs $\mathcal{A}^+$ as a subroutine and answers the oracle queries of $\mathcal{A}^+$ on its own. The idea in constructing $\mathcal{A}$ is to use the fact that in attacking SKE^+, the adversary $\mathcal{A}^+$ is able to relay messages only between user instances with the same session ID and the matching nonce sequences. Based on this idea, the adversary $\mathcal{A}$ obtains a transcript T of SKE for each unique combination of session ID and nonce sequences by calling its own $\mathsf{Execute}$ oracle, and generates a transcript T^+ of SKE^+ by patching T with appropriate signatures. $\mathcal{A}$ then use the messages of T^+ in answering $\mathcal{A}^+$'s Send queries directed to user instances which have the same session ID and nonce sequences as used in generating T^+. In this way, $\mathcal{A}^+$ is limited to sending messages already contained in T^+, unless signature forgery and nonce repetition occur. In essence, $\mathcal{A}$ is ensuring that $\mathcal{A}^+$'s capability of attacking protocol SKE^+ is demonstrated only on the session key associated with the patched transcript T^+ and thus is translated directly into the capability of attacking protocol SKE. Due to space limitations here, we will provide the proof of the theorem in the full version of this paper.

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. 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.
4. K. Becker and U. Wille. Communication complexity of group key distribution. *5th ACM Conference on Computer and Communications Security (CCS '98)*, pp. 1–6, 1998.
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. Provably secure session key distribution — the three party case. *27th ACM Symposium on Theory of Computing (STOC '95)*, pp. 57–66, 1995.
8. 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.

9. 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.
10. 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.
11. 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.
12. 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.
13. M. Burmester and Y. Desmedt. Efficient and secure conference-key distribution. *1996 International Workshop on Security Protocols*, LNCS vol. 1189, pp. 119–129, 1997.
14. 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.
15. 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.
16. 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`.
17. 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.
18. D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, vol. 24, no. 8, pp. 533–536, 1981.
19. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
20. W. Diffie, P. Oorschot, and M. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992.
21. 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.
22. 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.
23. S. Hirose and S. Yoshida. An authenticated Diffie-Hellman key agreement protocol secure against active attacks. *1st International Workshop on Practice and Theory in Public Key Cryptography (PKC '98)*, LNCS vol. 1431, pp. 135–148, 1998.
24. 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.
25. 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*.
26. B. Jung, S. Paeng, and D. Kim. Attacks to Xu-Tilborg's conference key distribution scheme. *IEEE Communications Letters*, vol. 8, no. 7, pp. 446–448, 2004.

27. 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.
28. 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.
29. J. Katz and M. Yung. Scalable protocols for authenticated group key exchange. *Advances in Cryptology – CRYPTO '03*, LNCS vol. 2729, pp. 110–125, 2003.
30. 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.
31. 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.
32. Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. *IFIP SEC '01*, pp. 229–244, 2001.
33. H. Krawczyk. HMQV: a high-performance secure Diffie-Hellman protocol. *Advances in Cryptology – CRYPTO '05*, LNCS vol. 3621, pp. 546–566, 2005.
34. 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.
35. M. Mayer and M. Yung. Secure protocol transformation via "Expansion": From two-party to groups. *6th ACM Conference on Computer and Communications Security (CCS '99)*, pp. 83–92, 1999.
36. 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.
37. 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.
38. 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.
39. 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.
40. V. Shoup. On formal models for secure key exchange. *Cryptology ePrint Archive*, Report 1999/012, 1999. Available at `http://eprint.iacr.org/`.
41. D. Wallner, E. Harder, and R. Agee. Key management for multicast: issues and architectures. RFC 2627, 1999.
42. 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*.