12-2010

Dynamic group key exchange revisited

Guomin YANG
Singapore Management University, gmyang@smu.edu.sg

Chik How TAN
National University of Singapore

Citation

YANG, Guomin and TAN, Chik How. Dynamic group key exchange revisited. (2010). *Cryptology and Network Security: 9th International Conference, CANS 2010, Kuala Lumpur, Malaysia, December 12-14: Proceedings*. 6467, 261-277.

Available at: https://ink.library.smu.edu.sg/sis_research/7418

Dynamic Group Key Exchange Revisited

Guomin Yang and Chik How Tan

Temasek Laboratories
National University of Singapore
{tslyg,tsltch}@nus.edu.sg

Abstract. In a dynamic group key exchange protocol, besides the basic group setup protocol, there are also a join protocol and a leave protocol, which allow the membership of an existing group to be changed more efficiently than rerunning the group setup protocol. The join and leave protocols should ensure that the session key is updated upon every membership change so that the subsequent sessions are protected from leaving members (*backward security*) and the previous sessions are protected from joining members (*forward security*). In this paper, we present a new security model for dynamic group key exchange. Comparing to existing models, we do a special treatment to the state information that a user may use in a sequence of setup/join/leave sessions. Our treatment gives a clear and more concise definition of session freshness for group key exchange in the dynamic setting. We also construct a new dynamic group key exchange protocol that achieves strong security and high efficiency in the standard model.

Keywords: Group Key Exchange, Dynamic Group, Insider Security, Mutual Authentication, Strong Contributiveness.

1 Introduction

Group key exchange (GKE) protocols are mechanisms by which a group of $n > 2$ parties communicate over an insecure network can generate a common secret key (usually we call this common secret key a session key as a user may have multiple key exchange sessions). A static group key exchange (SGKE) protocol consists of a long-lived key generation algorithm, and a group setup protocol which is invoked whenever a group of parties want to establish a shared key. In contrast, a dynamic group key exchange (DGKE) protocol consists of a long-lived key generation algorithm and three sub-protocols: a setup protocol, a join protocol, and a leave protocol. The join and leave protocols allow the membership of an existing group to be changed more efficiently than rerunning the group setup protocol. In this paper, we focus on the dynamic setting.

Since the goal of a GKE protocol is to establish a shared key that is known only to the group members, the main security requirement is to make sure that no information, not even a single bit, of the agreed key is leaked to any passive or active adversaries outside the group. For the special case of two-party key exchange, the problem has been extensively studied (e.g., [3, 4, 2, 13, 25] and

S.-H. Heng, R.N. Wright, and B.-M. Goi (Eds.): CANS 2010, LNCS 6467, pp. 261–277, 2010.

many more). Though it might be natural to extend the existing results for two-party key exchange to the group setting, such an extension is less trivial for the dynamic group case. For dynamic group key exchange, special attentions should be paid to the join and leave protocols: these two protocols must make sure that upon every membership change, the session key is updated so that the subsequent sessions are protected from leaving members (namely, *backward security*) and the previous sessions are protected from joining members (namely, *forward security*[1]). In other words, for the leave/join protocol, we have to assume that the leaving/joining users are the adversary, who may misbehave (i.e. deviate from the protocol specification) in one session, and try to break the security of a subsequent/previous session among other parties.

To see more clearly the differences between a two-party (or static group) protocol and a dynamic group protocol: first, in the dynamic setting, the state information of the honest parties in two sessions (e.g., one involves a joining/leaving adversary and the other the adversary wants to attack) can be related, so the definition of a fresh (or clean) session is less straightforward when session state reveal queries are allowed; secondly, for several, say t, related sessions created in a sequence of setup/join/leave events, the adversary can choose to attack one session but at the same time actively participate in the rest $t-1$ sessions, in which the adversary may not honestly follow the protocol specification (e.g., the adversary may plant some trapdoor information in one session, and try to break the following sessions after he/she leaves the group); thirdly, when considering the notion of *forward secrecy*, a secure DGKE protocol should ensure a past session key K_i remains secure even when the adversary compromises both the long-lived keys and the state information of the subsequent join/leave sessions after session i.

For group key exchange protocols, it is also necessary to consider *insider* security. Two insider security issues need to be addressed: mutual authentication and key control resistance. Different from two-party protocols, mutual authentication for GKE protocol means when a party makes a decision "accept" in a session, he/she must be sure that all his/her partners have indeed participated in the session, in particular, a subset of protocol participants should not be able to impersonate another party. For key control resistance, we require a secure group key exchange protocol to be able to prevent a subset of users from controlling (any part) of the session key.

Related Work. The case of two-party key exchange protocols has been extensively studied in the past three decades (e.g., [16, 17, 3, 4, 2, 13, 25] and many more). A lot of efficient and provably secure two party key exchange protocols have been constructed, and many have been deployed in the real practice (e.g. SSL, SSH, IPSec).

[1] We remark that the notion of forward security with respect to the join protocol is different from the conventional notion of "forward secrecy". Informally, the latter means an established session key remains secure even if an adversary learns the long-lived keys of the protocol participants in the future.

Some early group key exchange protocols [12, 1, 29, 30, 21] extended two-party key exchange (e.g., the two-party Diffi-Hellman protocol) to the multi-party setting. In [9], based on the Bellare-Rogaway security model [3] for two-party key exchange, Bresson, Chevassut, Pointcheval, and Quisquater proposed the first computational security model (referred to as the BCPQ model) for (static) group key exchange protocols. The BCPQ model and its variants then became the *de facto* standard for analyzing group key exchange protocols.

In [23], Katz and Yung proposed a scalable compiler to transform any static GKE protocol secure against passive adversaries to a new protocol secure against active adversaries. They then applied their compiler to the Burmester-Desmedt protocol and obtained the first constant-round and fully-scalable static GKE protocol in the standard model.

In [8], a formal model for dynamic group key exchange was proposed, this model was later extended in [7] to consider concurrent protocol executions and strong corruption. Provably secure protocols which require linear round complexity are also proposed in [8, 7]. In [24], Lim et al. proposed another DGKE protocol provably secure in the model of Bresson et al., the protocol requires only two rounds but is proven secure in the random oracle model. In [18], Dutta et al. presented another security model for dynamic group key exchange, and a protocol with formal security proof, but neither their model nor the protocol was carefully designed, as a result the model didn't really capture forward/backward security [32] and the protocol is flawed [31]. In [26], Manulis presented a 3-round provably secure DGKE protocol (denoted by Dynamic TDH1) which uses a binary tree structure.

Insider Security. Katz and Shin [22] provided the first formal definition of insider security for static group key exchange protocols under the Universal Composability (UC) framework [14]. They also presented a generic compiler to build UC-secure Static GKE protocols. Recently, in [19], Furukawa et al. presented a more round-efficient UC-secure Static GKE protocol which requires only two rounds, but the protocol requires each protocol participants to perform $O(n)$ pairing operations for an n-party group.

The insider security by Katz and Shin [22] considers impersonation attacks by malicious protocol participants. Later, Bresson and Manulis [11] unified this notion with key confirmation, and unknown key share resistance into their definition of Mutual Authentication. Recently, Gorantla el al. [20] further unified the Mutual Authentication definition by Bresson and Manulis with the notion of Key Compromise Impersonation resistance.

The notion of key control was first introduced by Mitchell et al. [27], which refers to attacks by which part of the protocol participants aim to control the value of the session key. Resistance to key control attacks is formalized via the notion of "contributiveness" in [6, 11] for group key exchange protocols. In [10], Bresson and Manulis presented a compiler that can transform any GKE protocols to achieve contributiveness at the cost of 2 extra communication rounds.

Our Contributions. In this paper, we present a new security model for dynamic group key exchange protocols. To define forward and backward security, we provide a special treatment to the shared state information among a sequence of setup/join/leave sessions. Our treatment gives a clear and more concise definition of session freshness than existing models. We then construct an efficient DGKE protocol and prove its security in the standard model.

2 Security Definitions

We start by some existing definitions and notations [3, 5] for key exchange protocols.

Protocol Participants and Long-Lived Keys. Let $\mathcal{HU}$ denote a nonempty set of parties. Each party $U \in \mathcal{HU}$ is provided with a Long-Lived Key LLK_U that is generated by running a long-lived key generation algorithm KG. Later on, new users can still be created and added into the system by the adversary. Such an adversarial capability was first considered in [11]. Let $\mathcal{MU}$ denote the set of users added by the adversary. When being created, a long-lived key LLK_M for M is generated by the adversary with the restriction that LLK_M^{Pub}, which denotes the *public* part of LLK_M, has never been used by any other user in the system. However, we do *not* require LLK_M to be generated by honestly running the long-lived key generation algorithm KG, and the adversary may not know the secret part of LLK_M.

Instance Oracles. A party may run many instances concurrently except that an instance in a join or leave session cannot run concurrently with any of its ancestors (see footnote 2). We call each instance of a party an oracle, and we use Π_U^i ($i \in \mathbb{N}$) to denote the ith instance of party U. All the oracles within a party U share the same long-lived key LLK_U. An oracle is activated when it receives an incoming message from the network. Upon activation, the oracle performs operations by following the Setup, Join or Leave protocol. We assume that whenever a user wants to setup a new group or join an existing group, an *unused* oracle will be used.

Session and Partner IDs, State Information, and Session Keys. When an oracle Π_U^i is activated to start a protocol (Setup, Join or Leave), it learns its partner id pid_U^i (similar to [11], we let pid_U^i include the identity of U itself). At the time Π_U^i makes a decision Accept, it outputs (secret) session key k_U^i under a session id sid_U^i which is determined during the protocol execution. Since we are considering group key exchange in the dynamic setting, some state information of Π_U^i should also be saved. This information would be used if a Join or Leave event later. In this paper, we assume that, after an instance Π_U^i accepts, its state information (which includes two fields $\mathsf{pid}_{old} = \mathsf{pid}_U^i$ and $\mathsf{sid}_{old} = \mathsf{sid}_U^i$) will be passed to another unused instance Π_U^j (possibly picked by the adversary), and Π_U^j would replace Π_U^i to participate in the next Join or Leave session. Also, after Π_U^j receives the state information from Π_U^i, all the state information in Π_U^i is erased, and Π_U^j is labeled *used*.

Discussion. A straightforward way to deal with the join and leave events is to let Π_U^i keep the state information at the end of a session and become active again in the subsequent Join or Leave event. However, such an approach will introduce troubles when we later define the *freshness* of an instance, since one instance may participate in different sessions with different partners and generate different session IDs. On the other hand, as we will see later, our approach makes the security definition easy and clean. Besides, our approach is meaningful, it mimics the following space-friendly implementation in real practice: after a KE session is completed and the session key is returned to the upper layer application, the user saves the necessary state information at a safe place in the harddisk, and erases that copy of the program (which implements the DGKE protocol) and all its state information from the memory. Later, when a join or leave event occurs, the user starts a new copy of the protocol/program with the previously saved state information as the parameters to the program.

Definition 1. *A Dynamic Group Key Exchange Protocol $\mathcal{DGKE}$ consists of a Long-Lived Key generation algorithm KG, a group setup protocol Setup, a join protocol Join, and a leave protocol Leave.*

- $\mathsf{KG}(1^k)$*: On input a security parameter 1^k, the long-lived key generation algorithm provides each user with a long-lived key LLK_U.*
- $\mathsf{Setup}(\mathcal{I})$*: on input a set of user identities $\mathcal{I}$, the setup protocol creates a new multicast group $\mathcal{G} = \mathcal{I}$.*
- $\mathsf{Join}(\mathcal{G}', \mathcal{I})$*: on input an existing multicast group $\mathcal{G}'$ and a set of users $\mathcal{I}$ such that $\mathcal{I} \cap \mathcal{G}' = \emptyset$, the join protocol creates a new multicast group $\mathcal{G} = \mathcal{G}' \cup \mathcal{I}$.*
- $\mathsf{Leave}(\mathcal{G}', \mathcal{I})$*: on input an existing multicast group $\mathcal{G}'$ and a set of users $\mathcal{I} \subset \mathcal{G}'$, the leave protocol creates a new multicast group $\mathcal{G} = \mathcal{G}' \backslash \mathcal{I}$.*

An execution of $\mathcal{DGKE}$ consists of running the KG algorithm once, and many concurrent executions of the other three protocols. We say $\mathcal{DGKE}$ is correct if, when no adversary is present, all the parties in the group $\mathcal{G}$ compute the same session key at the end of the Setup, Join or Leave protocol.

Security Model. We consider the following game that involves all the users in the set $\mathcal{HU}$ and an adversary $\mathcal{A}$. All the users are connected via an unauthenticated network that is controlled by $\mathcal{A}$. The game is initiated by running the long-lived key generation algorithm KG to provide each user in $\mathcal{HU}$ with a long-lived key. The adversary $\mathcal{A}$ is then given $\{LLK_U^{Pub}\}_{U \in \mathcal{HU}}$ and interacts with the oracles via the queries described below.

- $\mathsf{Register}(M, LLK_M^{Pub})$: This query allows the adversary $\mathcal{A}$ to create and add a new user M with long-live (public) key LLK_M^{Pub} into the system. We require that neither the user identity M nor the long-lived public key LLK_M^{Pub} has been used by any other user in the system. However, we don't require LLK_M^{Pub} to be generated by running the KG algorithm. All the activities and operations of user M will be performed by the adversary $\mathcal{A}$.

- $\mathsf{Send}(\Pi_U^i, msg_{in})$: This query allows $\mathcal{A}$ to invoke instance Π_U^i with an incoming message msg_{in}. Upon receiving the message, Π_U^i performs operations according to the Setup, Join or Leave protocol, and generates the response. Should Π_U^i accept or reject will be made available to $\mathcal{A}$. When an oracle Π_U^i accepts, $\mathcal{A}$ chooses another unused instance Π_U^j. The state information of Π_U^j is then set to $\mathsf{St}_U^j = \mathsf{St}_U^i$ (this operation is assumed to be done within the user U. In particular, the adversary is unaware of the state information being passed, but the adversary may learn this information via a RevealState query (described below) to Π_U^j). Π_U^j is labeled *used* and will replace Π_U^i to participate in the subsequent Join or Leave protocol. The Setup, Join and Leave events are also activated by $\mathcal{A}$ through Send queries, as follows:
  1. When $\mathcal{A}$ wants to activate an *unused* instance Π_U^i to start the Setup protocol, it sets $msg_{in} = \mathsf{setup}\|\mathsf{pid}$ where pid is the partner id of the instance Π_U^i.
  2. When $\mathcal{A}$ wants activate an instance Π_U^i, which is either unused (i.e. U is going to join an existing group) or has been used (i.e. U is a member of the existing group), to start the Join protocol, it sets $msg_{in} = \mathsf{join}\|\mathsf{pid}_{old}\|\mathsf{sid}_{old}\|\mathsf{pid}_{new}$ where pid_{old} denotes the old group (with session id sid_{old}) on top of which a new group pid_{new} is to be built. It is worth noting that several groups with the same set of members may exist, so the session id is necessary to uniquely identify the (old) group.
  3. Similarly, when $\mathcal{A}$ wants activate an instance Π_U^i to start the Leave protocol, it sets $msg_{in} = \mathsf{leave}\|\mathsf{pid}_{new}$. Note that if Π_U^i participates in the leave protocol, it should already have the information of the existing group, so the fields pid_{old} and sid_{old} are not needed.
- $\mathsf{Corrupt}(U)$: This query allows the adversary to obtain the long-lived key LLK_U.
- $\mathsf{RevealKey}(\Pi_U^i)$: This query reveals the session key being held by the oracle Π_U^i.
- $\mathsf{RevealState}(\Pi_U^i)$: This query reveals all the state information, but not the long-lived key, currently being held by the oracle Π_U^i.
- $\mathsf{Test}(\Pi_U^i)$: This query is asked only once in the game, and is only available if oracle Π_U^i has accepted, and is *fresh* (see below). An unbiased coin b is tossed, if $b = 0$, a random value drawn from the session key space is returned; if $b = 1$, the real session key k_U^i is returned. After the Test query, the adversary can still perform those queries described above.

Oracle Freshness. An oracle Π_U^i is *fresh* if all of the following conditions hold:

1. $\mathsf{pid}_U^i \cap \mathcal{MU} = \emptyset$;
2. No user in pid_U^i is corrupted before the adversary makes a $\mathsf{Send}(\Pi_V^j, *)$ query with $(V \in \mathsf{pid}_U^i \wedge \mathsf{sid}_V^j = \mathsf{sid}_U^i)$;
3. No RevealState query is performed to an oracle Π_V^j with $(V \in \mathsf{pid}_U^i \wedge \mathsf{sid}_V^j = \mathsf{sid}_U^i)$ or any of its ancestors[2].

[2] We say Π_V^i is an ancestor of Π_V^j if there exists a path $(\Pi_V^i, ..., \Pi_V^t, ...\Pi_V^j)$ such that each instance in the path passes its state information to the next one.

4. No RevealKey query is performed to an oracle Π_V^j with $(V \in \mathsf{pid}_U^i \wedge \mathsf{sid}_V^j = \mathsf{sid}_U^i)$.

SK-Security. Before $\mathcal{A}$ terminates it outputs a bit b'. We say $\mathcal{A}$ wins the game if $b' = b$. We define the advantage of the adversary $\mathcal{A}$ attacking protocol $\mathcal{DGKE}$ to be

$$\mathsf{Adv}^{sk}_{\mathcal{A},\mathcal{DGKE}}(k) = |\Pr[b' = b] - \frac{1}{2}|$$

Definition 2 (SK-Security). *We say a dynamic group key exchange protocol $\mathcal{DGKE}$ is SK-secure if, for any polynomial time adversary $\mathcal{A}$, the advantage $\mathsf{Adv}^{sk}_{\mathcal{A},\mathcal{DGKE}}(k)$ is a negligible function of the security parameter k.*

Comparisons with Existing Models. In the early models by Bresson et al. [8, 7], a single instance will maintain the state information in a sequence of setup, join, and leave events. When defining the freshness, the adversary is not allowed to reveal the state information of the instance (or any of its partners) at any time. As a consequence, it does not capture the scenario that the adversary compromises the state information in the join/leave sessions which are subsequent sessions of the test session.

In the model by Dutta and Barua [18], the adversary can only passively received communication transcripts of the join and leave protocols, but in reality, the adversary is the joining/leaving user who actively participated in the join/leave session. In [32], it has been shown that the proven secure protocol in [18] doesn't provide backward security.

In the model by Manulis [26], similar to the approach by Bresson et al., a single instance will maintain the state information in a sequence of setup/join/leave events, however the adversary is allowed to reveal the state information of the instance output in the test query at some points. While in our model, due to our trick to the state information, each oracle will participate in at most one session, which makes the definition of session freshness more concise as we don't need to consider at which points the adversary is allowed to perform RevealState to an instance.

Another difference between Manulis's model and ours is that we have different meanings in backward security. In Manulis's model, backward security considers an adversary who compromises state information of the ancestors (see footnote 2) of the instances in the test session, while ours means the leaving users, who may plant some trapdoors in the previous sessions, cannot learn any information of the session key established among the remaining group of users, but our model requires none of the instances in the test session, or any of their ancestors, has been asked a RevealState query. The definition by Manulis is stronger than ours, however, such a definition may be too strong as no existing DGKE protocol (including the dynamic TDH1 by Manulis) can achieve such a security level. The reason is that in order to perform join/leave efficiently, some critical state information (such as the DH exponents) used by the join/leave protocol is related to state information of the previous session. In order to provide the backward security defined by Manulis, each instance needs to freshly generate all critical

state information in each session, in which case the join/leave protocol most likely gains no advantage than the setup protocol (i.e., the protocol essentially becomes a static one).

MA-Security. We say that an instance Π_U^i is *honest* if $U \in \mathcal{HU}$ and Π_U^i honestly performs its operations according to the protocol. Below we review the definition of MA-security in [20]. The definition is a modification of the MA-security in [11] by including the notion of Key Compromise Impersonation (KCI) resistance. Recall that in a KCI attack an adversary corrupts a user U and then impersonates other (uncorrupted) users to (honest instances of) U.

Definition 3 (MA-Security). *Let $\mathcal{DGKE}$ be a correct dynamic group key exchange protocol and $\mathcal{A}'$ an adversary who is allowed to perform* Register, Send, Corrupt, RevealKey *and* RevealState *queries. We say that $\mathcal{A}'$ breaks the mutual authentication of $\mathcal{DGKE}$ if at some point during the execution of $\mathcal{DGKE}$, there exists an honest instance Π_U^i who has accepted with k_U^i and another user $V \in \mathcal{HU} \cap \mathsf{pid}_U^i$ who is uncorrupted at the time Π_U^i accepts such that*

1. *There is no instance oracle Π_V^t with $(\mathsf{pid}_V^t, \mathsf{sid}_V^t) = (\mathsf{pid}_U^i, \mathsf{sid}_U^i)$, or*
2. *There is an instance oracle Π_V^t with $(\mathsf{pid}_V^t, \mathsf{sid}_V^t) = (\mathsf{pid}_U^i, \mathsf{sid}_U^i)$ that has accepted with $k_V^t \neq k_U^i$.*

Denote $\mathsf{Adv}^{ma}_{\mathcal{A}',\mathcal{DGKE}}(k)$ the probability that $\mathcal{A}'$ breaks the mutual authentication of $\mathcal{DGKE}$. We say a dynamic group key exchange protocol $\mathcal{DGKE}$ is MA-secure if for any polynomial time adversary $\mathcal{A}'$, $\mathsf{Adv}^{ma}_{\mathcal{A}',\mathcal{DGKE}}(k)$ is a negligible function of the security parameter k.

Contributiveness. We present below the notion of contributiveness for dynamic group key exchange protocols. A group key exchange protocol secure under this notion can resist key control attacks where a subset of insiders tries to control any part of the resulting session key.

Definition 4 (Co-Security). *Let $\mathcal{DGKE}$ be a correct dynamic group key exchange protocol and $\mathcal{A}'' = (\mathcal{A}_1'', \mathcal{A}_2'')$ an adversary who is allowed to perform* Register, Send, Corrupt, RevealKey *and* RevealState *queries. $\mathcal{A}''$ runs in two stages:*

- *(Prepare.) $\mathcal{A}_1''$ performs the oracle queries and outputs a bit b, an index j, along with some state information* St.
- *(Attack.) On input* St, *$\mathcal{A}_2''$ performs the oracle queries and finally outputs (U, i).*

We say that $\mathcal{A}''$ wins if

1. *Π_U^i has terminated accepting k_U^i such that the j-th bit of k_U^i is equal to b,*
2. *Π_U^i is honest and has not started its execution in the Prepare stage,*
3. *Π_U^i has never been asked a* RevealState *query in the Attack phase.*

Define

$$\mathsf{Adv}^{co}_{\mathcal{A}'',\mathcal{DGKE}}(k) = \Pr[\mathcal{A}'' \text{ wins}] - \frac{1}{2}.$$

A dynamic group key exchange protocol $\mathcal{DGKE}$ is said to provide contributiveness (or Co-security) if for any polynomial time adversary $\mathcal{A}''$, $\mathsf{Adv}^{co}_{\mathcal{A}'',\mathcal{DGKE}}(k)$ is a negligible function of the security parameter k.

Our definition of contributiveness is different from the existing definitions of *strong* contributiveness defined in [11,20]. The later requires that the adversary cannot control the whole key, and as a result it does not capture partial-key control attacks. Below we present a protocol that achieves strong contributiveness but fail to achieve our notion of contributiveness.

U_1	U_2	U_3	U_4	U_5
$k_i \xleftarrow{\$} \{0,1\}^\ell, x_i \xleftarrow{\$} \mathbb{Z}_q^*, \sigma_i^1 \leftarrow \mathcal{DS}.Sign(sk_i, M_i^1 \| \mathcal{I})$				
$M_1^1 = k_1 \| g^{x_1}, \sigma_1^1$	$M_2^1 = k_2 \| g^{x_2}, \sigma_2^1$	$M_3^1 = k_3 \| g^{x_3}, \sigma_3^1$	$M_4^1 = k_4 \| g^{x_4}, \sigma_4^1$	$M_5^1 = H(k_5) \| g^{x_5}, \sigma_5^1$
Broadcast Round 1				
$\mathsf{sid} \leftarrow H(\mathcal{I} \| k_1 \| k_2 \| k_3 \| k_4 \| H(k_5)), t_i^L \leftarrow H(g^{x_{L(i)} x_i}), t_i^R \leftarrow H(g^{x_{R(i)} x_i})$				
$T_i \leftarrow t_i^L \oplus t_i^R, \hat{T}_5 \leftarrow k_5 \oplus t_5^R, \sigma_i^2 \leftarrow \mathcal{DS}.Sign(sk_i, M_i^2)$				
$M_1^2 = T_1 \| \mathsf{sid}, \sigma_1^2$	$M_2^2 = T_2 \| \mathsf{sid}, \sigma_2^2$	$M_3^2 = T_3 \| \mathsf{sid}, \sigma_3^2$	$M_4^2 = T_4 \| \mathsf{sid}, \sigma_4^2$	$M_5^2 = \hat{T}_5 \| T_5 \| \mathsf{sid}, \sigma_5^2$
Broadcast Round 2				
Session Key $\mathsf{k} = H(\mathcal{I} \| k_1 \| k_2 \| k_3 \| k_4 \| k_5)$				

Fig. 1. A (Static) Group Key Exchange Protocol [20]. $\mathcal{I} = \{U_1, U_2, U_3, U_4, U_5\}$.

The protocol in Fig. 1 is proven secure under the strong contributiveness definition in [11,20]. However, the protocol does not provide partial-key control resistance. In the protocol, the keying materials k_i of U_i $(1 \leq i \leq 4)$ are sent in clear at the beginning of the protocol, so the last user (i.e. U_5 in our example) can (partially) control the session key as follows: after seeing the k_i's of all the other users, U_5 repeatedly tries different values for k_5 until the session key $\mathsf{k} = H(\mathcal{I} \| k_1 \| k_2 \| k_3 \| k_4 \| k_5)$ has a desired pattern. We can see that in order to control s bits of the final session key, the expected number of trials that U_5 needs to perform is 2^s.

A similar attack can be performed to other protocols (e.g., those in [8, 7, 24, 18, 26, 11]) where a user can repeatedly try different keying materials after seeing the keying material sent by other users.

However, in our definition of contributiveness, we don't allow the adversary to make the RevealState query. This restriction is necessary in our definition since otherwise the adversary can use the RevealState query to learn the keying material of the instance under attack, and then repeatedly choose the proper keying materials for other users. In contrast, such a restriction is not required in the strong contributiveness definition in [11,20]. So in general, these two notions are incomparable. However, we believe partial-key control resistance may be more important in some circumstances.

It is also worth noting that partial-key control attacks have been considered in some previous work, such as in the shielded-insider privacy security notion

by Desmedt et al. [15], and in the *weak* contributiveness notion by Bresson and Manulis [10]. We leave the relationship among these partial-key control resistance notions an open problem.

3 A New Dynamic Group Key Exchange Protocol

In this section, we present a new dynamic group key exchange protocol and show that it satisfies all the security definitions (i.e. SK-, MA-, and Co-Security) given in Sec. 2. Our protocol makes use of a commitment scheme $\mathcal{CMT} = (\mathsf{CMT}, \mathsf{CVF})$, a digital signature scheme $\mathcal{DS}$, and two pseudo-random function families $\hat{\mathbf{F}}$ and $\tilde{\mathbf{F}}$.

3.1 Primitives

Commitment Schemes. A commitment scheme $\mathcal{CMT}$ consists of two algorithms: a commitment algorithm CMT which takes a message M to be committed as input and returns a commitment C and an opening key δ, and a deterministic verification algorithm CVF which takes C, M, δ as input and returns either 0 or 1. We say $\mathcal{CMT} = (\mathsf{CMT}, \mathsf{CVF})$ is a perfectly hiding and computationally binding commitment scheme with binding error ϵ if $\mathcal{CMT}$ achieves all the following properties.

- Consistency: for any message M

$$\Pr[(C, \delta) \xleftarrow{\$} \mathsf{CMT}(M) : \mathsf{CVF}(C, M, \delta) = 1] = 1.$$

- Perfectly Hiding: for any messages M_0 and M_1 such that $|M_0| = |M_1|$, the commitments C_0 and C_1 are identically distributed where $(C_0, \delta_0) \xleftarrow{\$} \mathsf{CMT}(M_0)$, and $(C_1, \delta_1) \xleftarrow{\$} \mathsf{CMT}(M_1)$.
- Computationally Binding: For every polynomial time algorithm $\mathcal{M}$

$$\mathrm{Adv}^{\mathrm{bind}}_{\mathcal{CMT},\mathcal{M}} \stackrel{\mathrm{def}}{=} \Pr\left[\begin{array}{l}(C, (M_0, \delta_0), (M_1, \delta_1)) \leftarrow \mathcal{M}(1^k) : \\ M_0 \neq M_1 \wedge \mathsf{CVF}(C, M_0, \delta_0) = 1 \\ \wedge \mathsf{CVF}(C, M_1, \delta_1) = 1\end{array}\right] \leq \epsilon(k).$$

In our protocol we will make use of a commitment scheme with the following additional property: for any message M, an honest execution of $\mathsf{CMT}(M)$ generates a commitment C that is uniformly distributed in the range of $\mathsf{CMT}(\cdot)$. We call such kind of commitment schemes *Uniformly Distributed* commitment schemes. A typical example of this type of commitment schemes is the Pedersen commitment scheme [28] where the computationally binding property holds under the Discrete Log assumption.

Pseudo-random Function Family. A family of efficiently computable functions $\mathbf{F} = \{\mathsf{F}_K : \mathsf{D} \rightarrow \mathsf{R} | K \in \mathsf{K}\}$ is called a pseudo-random function family, if for any polynomial time algorithm $\mathcal{A}$,

$$\mathsf{Adv}^{prf}_{\mathbf{F},\mathcal{A}}(k) \stackrel{\mathrm{def}}{=} \Pr[\mathcal{A}^{\mathsf{F}_\kappa(\cdot)}(1^k) = 1] - \Pr[\mathcal{A}^{\mathsf{RF}(\cdot)}(1^k) = 1]$$

is negligible where $\kappa \xleftarrow{\$} \mathsf{K}$ and $\mathsf{RF} : \mathsf{D} \rightarrow \mathsf{R}$ is a truly random function.

Digital Signature Scheme. A digital signature scheme $\mathcal{DS}$ consists of three algorithms: a key generation algorithm $\mathcal{DS}.Kg$ that takes a security parameter 1^k as input and returns a long-lived key pair (pk, sk) where pk is public and sk is private; a signing algorithm $\mathcal{DS}.Sign$ that takes a private key sk and a message $m \in \{0,1\}^*$ as input, and returns a signature σ; and a verification algorithm $\mathcal{DS}.Ver$ that takes a public key pk, a message m and a signature σ as input, and returns a bit $b \in \{0,1\}$ indicating the validity of the signature. The consistency requirement is that for any security parameter k and any message $m \in \{0,1\}^*$,

$$\Pr[(pk, sk) \leftarrow \mathcal{DS}.Kg(1^k) : \mathcal{DS}.Ver(pk, m, \mathcal{DS}.Sign(sk, m)) = 1] = 1.$$

We say $\mathcal{DS}$ is existentially unforgeable under chosen message attacks (uf-cma), if for any polynomial time algorithm $\mathcal{F}$,

$$\mathsf{Adv}^{uf-cma}_{\mathcal{DS},\mathcal{F}}(k) \stackrel{\text{def}}{=} \Pr\left[\begin{array}{l}(pk, sk) \leftarrow \mathcal{DS}.Kg(1^k),\\ (m^*, \sigma^*) \leftarrow \mathcal{F}^{\mathcal{DS}.Sign(sk,\cdot)}(pk) :\\ \mathcal{DS}.Ver(pk, m^*, \sigma^*) = 1\\ \wedge\ \mathcal{F} \text{ has never queried } \mathcal{DS}.Sign(sk, m^*)\end{array}\right]$$

is negligible.

3.2 The Protocol

Our protocol is an improved version of the protocol by Kim et al. [24] with the following differences: (1) the protocol in [24] is in the random oracle model while ours is in the standard model; (2) the protocol in [24] cannot achieve MA- and Co-security.

Protocol Design. The setup protocol makes use of two-party Diffie-Hellman key exchange to form a ring structure: each party generate an ephemeral public/private key pair (x_i, g^{x_i}), and generates a left key K_i^L (based on $g^{x_{i-1}x_i}$) with his left neighbor and a right key K_i^R (based on $g^{x_{i+1}x_i}$) with his right neighbor, and broadcasts $K_i^L \oplus K_i^R$. Then each group member can recover the left/right keys of all other group members due to the ring structure. One of the members also conceals its keying material using his right key, and others just send the keying materials in clear. Now only the legitimate group members can recover the concealed keying material and compute the final session key. To ensure contributiveness, we require each party to commit their keying material before receiving others' keying materials.

For the join/leave event, we let part of the existing group members to perform the same procedures as in the setup protocol, but the rest of the users do not need to run the full setup protocol, so computational and communication cost can be saved. And to ensure forward/backward security, the state information of the each user is updated using two pseudo-random functions. Below are the details of the protocol.

Let $\mathbb{G}$ denote a cyclic group of prime order q, and g is a generator of $\mathbb{G}$. Our dynamic group key exchange protocol works as follows:

- KG: For each user U_i inside the system, a long lived key pair $(pk_i, sk_i) \leftarrow \mathcal{DS}.Kg(1^k)$ is generated.
- Setup (Fig. 2): The following protocol is performed among a set $\mathcal{I} = \{U_1, U_2, ..., U_n\}$ of users.
  1. (Round 1) Each user U_i chooses $k_i \xleftarrow{\$} \{0,1\}^\ell, x_i \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $(k_i', \delta_i) \leftarrow \mathsf{CMT}(k_i)$. U_i then broadcasts $M_i^1 = k_i' \| g^{x_i}$.
  2. (Round 2) Upon receiving all the messages M_j^1 $(j \neq i)$, each U_i computes $\mathsf{sid}_i \leftarrow k_1' \| k_2' \| ... \| k_n'$, $t_i^L \leftarrow \hat{\mathsf{F}}_{g^{x_{L(i)} x_i}}(1)$[3], $t_i^R \leftarrow \hat{\mathsf{F}}_{g^{x_{R(i)} x_i}}(1)$, $\omega_i \leftarrow t_i^L \oplus t_i^R$. U_i $(1 \leq i < n)$ sets $M_i^2 = \omega_i \| k_i \| \delta_i$ and U_n computes $T_n \leftarrow t_n^R \oplus (k_n \| \delta_n)$ and sets $M_n^2 = \omega_n \| T_n$. Each U_i then generates a signature σ_i^2 on the message $M_i^1 \| M_i^2 \| \mathcal{I} \| \mathsf{sid}_i$ and broadcast $M_i^2 \| \sigma_i^2$.
  3. (Key Computation) Upon receiving $M_j^2 \| \sigma_j^2$ $(j \neq i)$, each U_i verifies all the signatures. If the signatures are valid, U_i derives $t_{i-1}^L \leftarrow t_i^L \oplus \omega_{i-1}$, $t_{i-2}^L \leftarrow t_{i-1}^L \oplus \omega_{i-2}$, ... until $t_1^L = t_n^R$ is derived. U_i then derives $k_n \| \delta_n \leftarrow t_n^R \oplus T_n$. U_i then verifies if $\mathsf{CVF}(k_j', k_j, \delta_j) = 1$ for all $j \neq i$. If all the verifications are successful, U_i computes the session key as $\mathsf{k}_i \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(1)$ where $\hat{k} \leftarrow \bigoplus_{U_j \in \mathcal{I}} k_j$.
  4. (Post Computation) Each U_i computes $h_i^L \leftarrow \hat{\mathsf{F}}_{g^{x_{L(i)} x_i}}(0), h_i^R \leftarrow \hat{\mathsf{F}}_{g^{x_{R(i)} x_i}}(0)$ and $X \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(0)$, saves (h_i^L, h_i^R, X) with $\mathsf{pid}_i = \mathcal{I}$ and sid_i in the memory, and erases all other state information.

U_1	U_2	U_3	U_4
	$k_i \xleftarrow{\$} \{0,1\}^\ell, (k_i', \delta_i) \leftarrow \mathsf{CMT}(k_i), x_i \xleftarrow{\$} \mathbb{Z}_q^*$		
$M_1^1 = k_1' \| g^{x_1}$	$M_2^1 = k_2' \| g^{x_2}$	$M_3^1 = k_3' \| g^{x_3}$	$M_4^1 = k_4' \| g^{x_4}$
	Broadcast Round 1		
	$\mathsf{sid}_i \leftarrow k_1' \| k_2' \| k_3' \| k_4', t_i^L \leftarrow \hat{\mathsf{F}}_{g^{x_{L(i)} x_i}}(1), t_i^R \leftarrow \hat{\mathsf{F}}_{g^{x_{R(i)} x_i}}(1), \omega_i \leftarrow t_i^L \oplus t_i^R$ $T_4 \leftarrow t_4^R \oplus (k_4 \| \delta_4), \sigma_i^2 \leftarrow \mathcal{DS}.Sign(sk_i, M_i^1 \| M_i^2 \| \mathcal{I} \| \mathsf{sid}_i)$		
$M_1^2 = \omega_1 \| k_1 \| \delta_1, \sigma_1^2$	$M_2^2 = \omega_2 \| k_2 \| \delta_2, \sigma_2^2$	$M_3^2 = \omega_3 \| k_3 \| \delta_3, \sigma_3^2$	$M_4^2 = \omega_4 \| T_4, \sigma_4^2$
	Broadcast Round 2		
	Session Key $\mathsf{k} = \tilde{\mathsf{F}}_{\hat{k}}(1)$ where $\hat{k} = k_1 \oplus k_2 \oplus k_3 \oplus k_4$		
	Post Computation		
	$h_i^L \leftarrow \hat{\mathsf{F}}_{g^{x_{L(i)} x_i}}(0), h_i^R \leftarrow \hat{\mathsf{F}}_{g^{x_{R(i)} x_i}}(0), X \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(0)$		
h_1^L, h_1^R, X	h_2^L, h_2^R, X	h_3^L, h_3^R, X	h_4^L, h_4^R, X

Fig. 2. The Setup Protocol. $\mathcal{I} = \{U_1, U_2, U_3, U_4\}$.

[3] Here for simplicity we directly use the Diffie-Hellman key as the pseudo-random function key, in practice, one may need to first apply a Key Drivation Function (KDF) to the Diffie-Hellman key, and then use the output of the KDF as the pseudo-random function key.

- Join (Fig. 3): Given an old group $\mathcal{I} = \{U_1, U_2, ..., U_n\}$ where each member has state information (h_i^L, h_i^R, X) and a set of new users $\mathcal{J} = \{U_{n+1}, U_{n+2}, ..., U_{n+k}\}$, the Join protocol works as follows:
  1. (Round 1): Each U_i $(1 \leq i \leq n+k)$ chooses $\hat{k}_i \xleftarrow{\$} \{0,1\}^\ell$ and computes $(\hat{k}_i', \hat{\delta}_i) \leftarrow \mathsf{CMT}(\hat{k}_i)$. Each U_j $(j \in \{1, n, n+1, ..., n+k\})$ also chooses $\hat{x}_j \xleftarrow{\$} \mathbb{Z}_q^*$, and U_2 sets $\hat{x}_2 = X$. Then each U_i $(1 \leq i \leq n+k)$ broadcasts M_i^1 where $M_j^1 = \hat{k}_j' \| g^{\hat{x}_j}$ for $j \in \{1, 2, n, n+1, ..., n+k\}$ and $M_\ell^1 = \hat{k}_\ell'$ for $\ell \in \{3, ..., n-1\}$.
  2. (Round 2): Upon receiving all the messages, each $U_i (1 \leq i \leq n+k)$ computes $\mathsf{sid}_i \leftarrow \hat{k}_1' \| \hat{k}_2' \| ... \| \hat{k}_{n+k}'$. Each U_j $(j \in \{1, 2, n, n+1, ..., n+k\})$ then computes $t_j^L \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{L(j)}\hat{x}_j}}(1)$, $t_j^R \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{R(j)}\hat{x}_j}}(1)$ where $\hat{x}_{L(n)} = \hat{x}_2, \hat{x}_{R(2)} = \hat{x}_n$, and $\omega_i \leftarrow t_i^L \oplus t_i^R$. Each U_ℓ $(\ell \in \{3, ..., n-1\})$ also computes $t_\ell^R = t_2^R, t_\ell^L = t_2^L$ (as U_ℓ also has X), $\omega_\ell \leftarrow t_\ell^L \oplus t_\ell^R$. U_i $(i \in \{1, 2, n+1, ..., n+k\})$ sets $M_i^2 = \omega_i \| \hat{k}_i \| \hat{\delta}_i$, U_ℓ $(3 \leq \ell \leq n-1)$ sets $M_\ell^2 = \hat{k}_\ell \| \hat{\delta}_\ell$, and U_n computes $T_n \leftarrow t_n^R \oplus (\hat{k}_n \| \hat{\delta}_n)$ and sets $M_n^2 = \omega_n \| T_n$. Each U_i $(1 \leq i \leq n+k)$ generates a signature σ_i^2 on the message $M_i^1 \| M_i^2 \| \mathcal{I}' \| \mathsf{sid}_i$ and broadcasts $M_i^2 \| \sigma_i^2$.
  3. (Key Computation): Each $U_i \in \mathcal{I}'$ performs the same procedures as he/she does in the Key Computation phase of the Setup protocol, except that for each U_ℓ $(3 \leq \ell \leq n-1)$, $t_\ell^L = t_2^L$ and $t_\ell^R = t_2^R$ are used. The final session key of each U_i is computed as $\mathsf{k}_i \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(1)$ where $\hat{k} \leftarrow \bigoplus_{U_j \in \mathcal{I}'} \hat{k}_j$.
  4. (Post Computation) Each U_v $(v \in \{1, n+1, ..., n+k\})$ computes $h_v^L \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{L(v)}\hat{x}_v}}(0)$, each U_j $(j \in \{n, n+1, ..., n+k\})$ computes $h_j^R \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{R(j)}\hat{x}_j}}(0)$, and $h_1^R, h_n^L, h_\ell^L, h_\ell^R$ $(2 \leq \ell \leq n-1)$ remain unchanged. Each U_i $(i \in \{1, 2, ..., n+k\})$ computes $X' \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(0)$, saves (h_i^L, h_i^R, X') with $\mathsf{pid}_i = \mathcal{I}'$ and sid_i in the memory, and erases all other state information.
- Leave (Fig. 4): Let $\mathcal{I} = \{U_1, U_2, ..., U_n\}$ be an existing group where each member U_i has state information (h_i^L, h_i^R, X). Let $\mathcal{I}' = \mathcal{I} \backslash \mathcal{J}$ where $\mathcal{J} = \{U_{l_1}, U_{l_2}, ..., U_{l_{n'}}\}$ denotes the set of leaving users, and $N(\mathcal{J})$ be the set of neighbors of those leaving users, i.e. $N(\mathcal{J}) = \{U_{l_1-1}, U_{l_1+1}, ..., U_{l_{n'}-1}, U_{l_{n'}+1}\}$. The Leave protocol works as follows:
  1. (Round 1): Each U_i in $\mathcal{I}'$ randomly chooses $\tilde{k}_i \in \{0,1\}^\ell$ and computes $(\tilde{k}_i', \tilde{\delta}_i) \leftarrow \mathsf{CMT}(\tilde{k}_i)$. Each U_ℓ $(\ell \in \mathcal{I}' \backslash N(\mathcal{J}))$ sets $M_\ell^1 = \tilde{k}_\ell'$. Each U_j in $N(\mathcal{J})$ additionally chooses $\tilde{x}_j \xleftarrow{\$} \mathbb{Z}_q^*$, and sets $M_j^1 = \tilde{k}_i' \| g^{\tilde{x}_j}$. Each U_i in $\mathcal{I}'$ broadcasts M_i^1.
  2. (Round 2): Upon receiving all the messages, each U_i in $\mathcal{I}'$ computes sid_i as the concatenation of all the $\tilde{k}_j'$ sent by $U_j \in \mathcal{I}'$. Each pair of neighbors U_{l_j-1} and U_{l_j+1} in $N(\mathcal{J})$ generate $h_{l_j-1}^R = \hat{\mathsf{F}}_{g^{\tilde{x}_{l_j+1}\tilde{x}_{l_j-1}}}(0)$ and $h_{l_j+1}^L = \hat{\mathsf{F}}_{g^{\tilde{x}_{l_j-1}\tilde{x}_{l_j+1}}}(0)$, respectively. Each U_i in $\mathcal{I}'$ generates $t_i^L = \hat{\mathsf{F}}_{h_i^L}(1)$, $t_i^R = \hat{\mathsf{F}}_{h_i^R}(1)$, and $\omega_i \leftarrow t_i^L \oplus t_i^R$. The user $U_{l_{n'}+1}$ additionally computes $T_{l_{n'}+1} \leftarrow t_{l_{n'}+1}^R \oplus (\tilde{k}_{l_{n'}+1} \| \tilde{\delta}_{l_{n'}+1})$. Each U_i in $\mathcal{I}' \backslash \{U_{l_{n'}+1}\}$ then sets $M_i^2 = \omega_i \| \tilde{k}_i \| \tilde{\delta}_i$, $U_{l_{n'}+1}$ sets $M_{l_{n'}+1}^2 = \omega_{l_{n'}+1} \| T_{l_{n'}+1}$. Finally, each

U_1	U_2	U_3	U_4	U_5
h_1^L, h_1^R, X	h_2^L, h_2^R, X	h_3^L, h_3^R, X	h_4^L, h_4^R, X	
		$\hat{k}_i \xleftarrow{\$} \{0,1\}^\ell, (\hat{k}'_i, \hat{\delta}_i) \leftarrow \mathsf{CMT}(\hat{k}_i), \hat{x}_i \xleftarrow{\$} \mathbb{Z}_q^*, \hat{x}_2 \leftarrow X$		
$M_1^1 = \hat{k}'_1 \| g^{\hat{x}_1}$	$M_2^1 = \hat{k}'_2 \| g^X$	$M_3^1 = \hat{k}'_3$	$M_4^1 = \hat{k}'_4 \| g^{\hat{x}_4}$	$M_5^1 = \hat{k}'_5 \| g^{\hat{x}_5}$
		Broadcast Round 1		
		$t_i^L \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{L(i)}\hat{x}_i}}(1), t_i^R \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{R(i)}\hat{x}_i}}(1)$ for $\{U_1, U_2, U_4, U_5\}$, $t_3^L = t_2^L$ & $t_3^R = t_2^R$		
		$\omega_i \leftarrow t_i^L \oplus t_i^R, T_4 \leftarrow t_4^R \oplus (\hat{k}_4 \| \hat{\delta}_4)$, $\mathsf{sid}_i \leftarrow \hat{k}'_1 \| \hat{k}'_2 \| \hat{k}'_3 \| \hat{k}'_4 \| \hat{k}'_5, \sigma_i^2 \leftarrow \mathcal{DS}.Sign(sk_i, M_i^1 \| M_i^2 \| \mathcal{I}' \| \mathsf{sid}_i)$		
$M_1^2 = \omega_1 \| \hat{k}_1 \| \hat{\delta}_1, \sigma_1^2$	$M_2^2 = \omega_2 \| \hat{k}_2 \| \hat{\delta}_2, \sigma_2^2$	$M_3^2 = \hat{k}_3 \| \hat{\delta}_3, \sigma_3^2$	$M_4^2 = \omega_4 \| T_4, \sigma_4^2$	$M_5^2 = \omega_5 \| \hat{k}_5 \| \hat{\delta}_5, \sigma_5^2$
		Broadcast Round 2		
		Session Key $\mathsf{k} = \tilde{\mathsf{F}}_{\hat{k}}(1)$ where $\hat{k} \leftarrow \hat{k}_1 \oplus \hat{k}_2 \oplus ... \oplus \hat{k}_5$		
		Post Computation		
		$h'^L_i \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{L(i)}\hat{x}_i}}(0), h'^R_i \leftarrow \hat{\mathsf{F}}_{g^{\hat{x}_{R(i)}\hat{x}_i}}(0), X' \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(0)$		
h'^L_1, h_1^R, X'	h_2^L, h_2^R, X'	h_3^L, h_3^R, X'	h_4^L, h'^R_4, X'	h'^L_5, h'^R_5, X'

Fig. 3. The Join Protocol. $\mathcal{I}' = \{U_1, U_2, U_3, U_4, U_5\}, \mathcal{J} = \{U_5\}$.

U_i in $\mathcal{I}'$ generates a signature σ_i^2 on the message $M_i^1 \| M_i^2 \| \mathcal{I}' \| \mathsf{sid}_i$ and broadcasts $M_i^2 \| \sigma_i^2$.

3. (Key Computation): Each $U_i \in \mathcal{I}'$ performs the same procedures as he/she does in the Key Computation phase of the Setup protocol. The final session key of each U_i is computed as $\mathsf{k}_i \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(1)$ where $\hat{k} \leftarrow \bigoplus_{U_j \in \mathcal{I}'} \tilde{k}_j$.
4. (Post Computation): Each $U_i \in \mathcal{I}'$ computes $h'^L_i \leftarrow \hat{\mathsf{F}}_{h_i^L}(0), h'^R_i \leftarrow \hat{\mathsf{F}}_{h_i^R}(0), X' \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(0)$, saves (h'^L_i, h'^R_i, X') with $\mathsf{pid}_i = \mathcal{I}'$ and sid_i in the memory, and erases all other state information.

3.3 Security Analysis

We prove that our protocol is secure with respect to the security definitions (i.e. SK-, MA, and Co-security) given in Sec. 2.

Decisional Diffie-Hellman (DDH) Problem: Fix a generator g of $\mathbb{G}$. The DDH assumption claims that $\{g, g^a, g^b, Z\}$ and $\{g, g^a, g^b, g^{ab}\}$ are computationally indistinguishable where a, b are randomly selected from $\mathbb{Z}_q$ and Z is a random element of $\mathbb{G}$.

Theorem 1. *The proposed dynamic group key exchange protocol is SK-secure if the DDH assumption holds in the underlying group $\mathbb{G}$, $\mathcal{DS}$ is a uf-cma secure digital signature scheme, $\mathcal{CMT}$ is a uniformly distributed perfectly hiding commitment scheme, $\hat{\boldsymbol{F}}$ and $\tilde{\boldsymbol{F}}$ are two independent pseudo-random function families.*

We prove the Theorem in three cases: (1) the Test query is made to a setup session, (2) the Test query is made to a join session, and (3) the Test query is

U_1	U_2	U_3	U_4	U_5
h_1^L, h_1^R, X	h_2^L, h_2^R, X	h_3^L, h_3^R, X	h_4^L, h_4^R, X	h_5^L, h_5^R, X
$\tilde{k}_i \xleftarrow{\$} \{0,1\}^\ell, (\tilde{k}'_i, \tilde{\delta}_i) \leftarrow \mathsf{CMT}(\tilde{k}_i), \tilde{x}_i \xleftarrow{\$} \mathbb{Z}_q^*$				
$M_1^1 = \tilde{k}'_1$	$M_2^1 = \tilde{k}'_2$	$M_3^1 = \tilde{k}'_3 \| g^{\tilde{x}_3}$		$M_5^1 = \tilde{k}'_5 \| g^{\tilde{x}_5}$
Broadcast Round 1				
		$h_3^R \leftarrow \hat{\mathsf{F}}_{(g^{\tilde{x}_5})^{\tilde{x}_3}}(0)$		$h_5^L \leftarrow \hat{\mathsf{F}}_{(g^{\tilde{x}_3})^{\tilde{x}_5}}(0)$
		$\mathsf{sid}_i \leftarrow \tilde{k}'_1 \| \tilde{k}'_2 \| \tilde{k}'_3 \| \tilde{k}'_5, t_i^L \leftarrow \hat{\mathsf{F}}_{h_i^L}(1), t_i^R \leftarrow \hat{\mathsf{F}}_{h_i^R}(1), \omega_i \leftarrow t_i^L \oplus t_i^R$		
		$T_5 \leftarrow t_5^R \oplus (\tilde{k}_5 \| \tilde{\delta}_5), \sigma_i^2 \leftarrow \mathcal{DS}.Sign(sk_i, M_i^1 \| M_i^2 \| \mathcal{I}' \| \mathsf{sid}_i)$		
$M_1^2 = \omega_1 \| \tilde{k}_1 \| \tilde{\delta}_1, \sigma_1^2$	$M_2^2 = \omega_2 \| \tilde{k}_2 \| \tilde{\delta}_2, \sigma_2^2$	$M_3^2 = \omega_3 \| \tilde{k}_3 \| \tilde{\delta}_3, \sigma_3^2$		$M_5^2 = \omega_5 \| T_5, \sigma_5^2$
Broadcast Round 2				
		Session Key $\mathsf{k} = \tilde{\mathsf{F}}_{\hat{k}}(1)$ where $\hat{k} = \tilde{k}_1 \oplus \tilde{k}_2 \oplus \tilde{k}_3 \oplus \tilde{k}_5$		
Post Computation				
		$h'^L_i \leftarrow \hat{\mathsf{F}}_{h_i^L}(0), h'^R_i \leftarrow \hat{\mathsf{F}}_{h_i^R}(0), X' \leftarrow \tilde{\mathsf{F}}_{\hat{k}}(0)$		
h'^L_1, h'^R_1, X'	h'^L_2, h'^R_2, X'	h'^L_3, h'^R_3, X'		h'^L_5, h'^R_5, X'

Fig. 4. The Leave Protocol. $\mathcal{I}' = \{U_1, U_2, U_3, U_5\}, \mathcal{J} = \{U_4\}$.

made to a leave session. To prove (1), we just follow the same approach as other existing work does, we define a sequence of games, starting from the original SK-security game, ending with a game in which the adversary has no advantage, and show that the difference between each two consecutive games is negligible. For case (2) and (3), we use the idea that $f(0)$ and $f(1)$ "looks" random and independent to any polynomial time adversary if $f(\cdot)$ is a secure pseudo-random function, so even if the adversary see one of them, we can still replace the other with a random element.

Theorem 2. *The proposed dynamic group key exchange protocol is MA-secure if $\mathcal{DS}$ is a uf-cma secure digital signature scheme, and $\mathcal{CMT}$ is a computationally binding commitment scheme.*

Theorem 3. *The proposed dynamic group key exchange protocol is Co-secure if $\mathcal{CMT}$ is a perfectly hiding and computationally binding commitment scheme, and $\tilde{\boldsymbol{F}}$ is a pseudo-random function family.*

The detailed proofs are deferred to the full paper.

4 Conclusions and Future Work

In this paper, we presented a new security model for dynamic group key exchange (DGKE) protocols and a new definition of contributiveness which captures partial-key control attacks. Comparing to existing security models, our new model is more concise and easy to use. We also presented a new DGKE protocol that provides strong security as well as high efficiency. Some possible future work includes 1) study the relationship among the existing partial-key control resistance notions; and 2) construct a robust protocol that can handle the situation of user crash during protocol execution.

References

1. Ateniese, G., Steiner, M., Tsudik, G.: Authenticated group key agreement and friends. In: ACM Conference on Computer and Communications Security, pp. 17–26 (1998)
2. Bellare, M., Canetti, R., Krawczyk, H.: A modular approach to the design and analysis of authentication and key exchange protocols. In: Proc. 30th ACM Symp. on Theory of Computing, pp. 419–428. ACM, New York (May 1998)
3. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
4. Bellare, M., Rogaway, P.: Provably secure session key distribution – the three party case. In: Proc. 27th ACM Symp. on Theory of Computing, Las Vegas, pp. 57–66. ACM, New York (1995)
5. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
6. Bohli, J.-M., Gonzalez Vasco, M.I., Steinwandt, R.: Secure group key establishment revisited. Int. J. Inf. Sec. 6(4), 243–254 (2007)
7. Bresson, E., Chevassut, O., Pointcheval, D.: Dynamic group Diffie-Hellman key exchange under standard assumptions. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 321–336. Springer, Heidelberg (2002)
8. Bresson, E., Chevassut, O., Pointcheval, D.: Provably authenticated group Diffie-Hellman key exchange - the dynamic case. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 290–309. Springer, Heidelberg (2001)
9. Bresson, E., Chevassut, O., Pointcheval, D., Quisquater, J.-J.: Provably authenticated group Diffie-Hellman key exchange. In: ACM Conference on Computer and Communications Security, pp. 255–264 (2001)
10. Bresson, E., Manulis, M.: Contributory group key exchange in the presence of malicious participants. IET Information Security 2(3), 85–93 (2008)
11. Bresson, E., Manulis, M.: Securing group key exchange against strong corruptions and key registration attacks. International Journal of Applied Cryptography 1(2), 91–107 (2008)
12. Burmester, M., Desmedt, Y.: A secure and efficient conference key distribution system (extended abstract). In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 275–286. Springer, Heidelberg (1995)
13. Canetti, R., Krawczyk, H.: Analysis of key-exchange protocols and their use for building secure channels. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 453–474. Springer, Heidelberg (2001)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), `http://eprint.iacr.org/`
15. Desmedt, Y., Pieprzyk, J., Steinfeld, R., Wang, H.: A non-malleable group key exchange protocol robust against active insiders. In: Katsikas, S.K., López, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. LNCS, vol. 4176, pp. 459–475. Springer, Heidelberg (2006)
16. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory 22, 644–654 (1976)
17. Diffie, W., Van Oorschot, P.C., Wiener, M.J.: Authentication and authenticated key exchanges. Designs, Codes, and Cryptography 2(2), 107–125 (1992)

18. Dutta, R., Barua, R.: Constant round dynamic group key agreement. In: Zhou, J., López, J., Deng, R.H., Bao, F. (eds.) ISC 2005. LNCS, vol. 3650, pp. 74–88. Springer, Heidelberg (2005)
19. Furukawa, J., Armknecht, F., Kurosawa, K.: A universally composable group key exchange protocol with minimum communication effort. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 392–408. Springer, Heidelberg (2008)
20. Choudary Gorantla, M., Boyd, C., González Nieto, J.M.: Modeling key compromise impersonation attacks on group key exchange protocols. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 105–123. Springer, Heidelberg (2009)
21. Just, M., Vaudenay, S.: Authenticated multi-party key agreement. In: Kim, K.-c., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 36–49. Springer, Heidelberg (1996)
22. Katz, J., Shin, J.S.: Modeling insider attacks on group key-exchange protocols. In: ACM Conference on Computer and Communications Security, pp. 180–189 (2005)
23. Katz, J., Yung, M.: Scalable protocols for authenticated group key exchange. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 110–125. Springer, Heidelberg (2003)
24. Kim, H.-J., Lee, S.-M., Lee, D.H.: Constant-round authenticated group key exchange for dynamic groups. In: Lee, P.J. (ed.) ASIACRYPT 2004. LNCS, vol. 3329, pp. 245–259. Springer, Heidelberg (2004)
25. Krawczyk, H.: HMQV: A High-Performance Secure Diffie-Hellman Protocol. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 546–566. Springer, Heidelberg (2005)
26. Manulis, M.: Provably secure group key exchange. PhD Thesis, Ruhr University Bochum (2007), `http://www.manulis.eu/phd.html`
27. Mitchell, C., Ward, M., Wilson, P.: On key control in key agreement protocols. Electronics Letters 34, 980–981 (1998)
28. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1991)
29. Steer, D.G., Strawczynski, L., Diffie, W., Wiener, M.J.: A secure audio teleconference system. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 520–528. Springer, Heidelberg (1988)
30. Steiner, M., Tsudik, G., Waidner, M.: Diffie-Hellman key distribution extended to group communication. In: ACM Conference on Computer and Communications Security, pp. 31–37 (1996)
31. Tan, C.-H., Yang, G.: Comment on provably secure constant round contributory group key agreement in dynamic setting. IEEE Transactions on Information Theory (to appear)
32. Teo, J.C.M., Tan, C.H., Ng, J.M.: Security analysis of provably secure constant round dynamic group key agreement. IEICE Transactions 89-A(11), 3348–3350 (2006)