Constant Round Dynamic Group Key Agreement

Ratna Dutta and Rana Barua
Cryptology Research Group
Stat-Math Unit
203, B.T. Road, Kolkata
India 700108
e-mail:{ratna_r,rana}@isical.ac.in

Abstract

We present a fully symmetric constant round authenticated group key agreement protocol in dynamic scenario. Our proposed scheme achieves forward secrecy and is provably secure under DDH assumption in the security model of Bresson *et al.* providing, we feel, better security guarantee than previously published results. The protocol is efficient in terms of both communication and computation power.
Keywords: group key agreement, DDH problem, provable security [1]

1 Introduction

A group key agreement protocol allows a group of users communicating over an untrusted, open network to come up with a common secret value called a session key. This session key can be used to facilitate desirable security services, such as confidentiality and data integrity.

Authenticated group key agreement allows two or more parties to agree upon a common secret key even in the presence of active adversaries. These protocols are designed to deal with the problem to ensure users in the group that no other principals aside from members of the group can learn any information about the session key. The design of secure and efficient authenticated group key agreement protocols gets much attention in current research with increasing applicability in numerous group-oriented and collaborative applications [13, 17, 5, 18, 11, 19, 25, 31].

Constructing forward secure authenticated key agreement scheme in a formal security model has recently received much importance. Efficiency is another critical concern in designing such protocols for practical applications. In particular, number of rounds may be crucial in an environment where quite a large number of users are involved and the group-membership is dynamic. In a dynamic group key agreement, the users can join or leave the group at any time. Such schemes must ensure that the session key is updated upon every membership change, so that the subsequent sessions are protected from leaving members and the previous sessions are protected from joining members. The cost of updates associated with group membership changes should be minimum. There are quite a number of dynamic group key agreement protocols [14, 15, 16, 28, 26, 27, 31]. In this paper, we study the problem of dynamic authenticated group key agreement. We design our algorithm for join and leave to ensure minimum modification to the computation already precomputed when a pool of users join or leave the group and the session key is updated.

Our Contribution : The main contribution of this paper is to obtain a provably secure constant round

[1] This is a preliminary version of the paper that will be presented in ISC 2005.

authenticated group key agreement protocol in dynamic scenario where a user can join or leave the group at his desire with updated key. We propose in Section 3 a scheme that is proven to be secure against passive adversary assuming the intractability of decision Diffie-Hellman (DDH) problem. Then we authenticate this unauthenticated protocol by incorporating digital signature and provide a concrete security analysis against active adversaries in the model as formalized by Bresson *et al.* [15]. We appropriately modify the Katz-Yung [25] technique to achieve authentication in our protocol. Finally, we extend this static authenticated protocol to dynamic authenticated protocol by introducing algorithms for join and leave. We prove (Section 4) that the security of both the static and dynamic authenticated protocols rely on that of the unauthenticated protocol. The security model of Bresson *et al.* [15] is adopted for the security analysis of the dynamic case. Our protocol achieves forward secrecy, is fully symmetric and being of constant round, is more efficient as compared to the protocol of Bresson *et al.* [15] (whose round complexity is linear in the number of group members). Our security result holds in the standard model and thus provides better security guarantees than previously published results in the random oracle model.

More recently, Kim *et al.* [28] proposed a very efficient constant round dynamic authenticated group key agreement protocol and provide a security analysis of their static authenticated protocol which is shown to be secure under computation Diffie-Hellman (CDH) assumption using random hash oracle. They did not consider the security analysis of their dynamic authenticated protocol. Unlike [28], we have achieved the security of our dynamic scheme in the standard model under standard DDH assumption without using any random oracle. We separately analyze the security of our static unauthenticated protocol, static authenticated protocol and dynamic authenticated protocol and reduce the security of the static authenticated protocol and dynamic authenticated protocol to that of the unauthenticated protocol.

Our proposed scheme considers the users $U_1, U_2, \ldots, U_n$ participating in the protocol on a ring where U_{i-1}, U_{i+1} are respectively the left and right neighbors of U_i for $1 \leq i \leq n$ with $U_0 = U_n, U_{n+1} = U_1$. Only 2 rounds are required in our protocol which makes our protocol efficient from communication point of view. User $U_i, 1 \leq i \leq n$, sends a message in first round only to its neighbors U_{i-1}, U_{i+1} and a message in second round to the rest of the $n-1$ users. Each user sends one message in each round with bit length at most $2|q| + 2|s|$ where $|q|$ is the length of q, the order of the underlying group on which DDH problem is assumed to be hard and $|s|$ is the length of signature. Each group member computes at most 3 modular exponentiations (1 in round 1 and 2 in round 2), $2n-2$ modular multiplications ($n-1$ multiplications for recovery of all right keys and $n-1$ multiplications for session key computation), 1 division, 2 signature generation and $n+1$ signature verification.

Our protocol is more efficient as compared to the protocol of Burmester and Desmedt [18] (BD) in terms of both communication and computation power. Moreover, we emphasize that our protocol is dynamic. The authentication in BD protocol was introduced by Katz and Yung [25] (KY) that requires 3 rounds. Table 1 analyzes the efficiency of our static authenticated protocol and authenticated protocol KY [25] where both the schemes are forward secure, achieve provable security under DDH assumption in standard model. We use the following notations:

n	total number of users in a group
R	total number of rounds
PTP	maximum number of point-to-point communication per user
Exp	maximum number of modular exponentiations computed per user
Mul	maximum number of modular multiplications computed per user
Div	maximum number of divisions computed per user
Sig	maximum number of signatures generated per user
Ver	maximum number of signature verification per user

Protocol	Communication		Computation					Hardness Assumption	Remarks
	R	PTP	Exp	Mul	Div	Sig	Ver		
KY [25]	3	$3(n-1)$	3	$\frac{n^2}{2}+\frac{3n}{2}-3$	1	2	$2(n-1)$	DDH	static
Our protocol	2	$n+1$	3	$2n-2$	1	2	$n+1$	DDH	dynamic

Table 1: Protocol comparison

In each round of authenticated BD protocol, a user sends message to the rest of the users (although the communication in the second round can be reduced). In contrast, each user in our protocol sends a message only to its two neighbors in the first round and a message to the rest of the users in the second round. Our protocol differs from the BD protocol in the way the session key is computed after the rounds are over. Each user computes $\frac{n^2}{2}+\frac{3n}{2}-3$ modular multiplications in BD protocol. On a more positive note, each user in our protocol requires to compute at most $2n$ modular multiplications. This makes our protocol much more efficient as compared to BD protocol. Besides, our protocol has the ability to detect the presence of a corrupted group member, although we cannot detect who among the group members are behaving improperly. If an invalid message is sent by a corrupted member, then this can be detected by all legitimate members of the group and the protocol execution may be stopped instantly. This feature makes our protocol interesting when the adversarial model no longer assumes that the group members are honest.

2 Preliminaries

In this section, we define the Decision Diffie-Hellman (DDH) problem and describe the security model in which we prove the security of our group key agreement protocol. We use the notation $a \longleftarrow S$ to denote that a is generated randomly from S.

2.1 Decision Diffie-Hellman (DDH) problem

Let $G = \langle g \rangle$ be a multiplicative group of some large prime order q. Then Decision Diffie-Hellman (DDH) problem on G is defined as follows:

Instance : (g^a, g^b, g^c) for some $a, b, c \in Z_q^*$.

Output : yes if $c = ab \bmod q$ and output no otherwise.

We consider two distributions as:

$$\Delta_{\mathsf{Real}} = \{a, b \longleftarrow Z_q^*, A = g^a, B = g^b, C = g^{ab} : (A, B, C)\}$$
$$\Delta_{\mathsf{Rand}} = \{a, b, c \longleftarrow Z_q^*, A = g^a, B = g^b, C = g^c : (A, B, C)\}.$$

The advantage of any probabilistic, polynomial-time, 0/1-valued distinguisher $\mathcal{D}$ in solving DDH problem on G is defined to be : $\mathsf{Adv}_{\mathcal{D},G}^{\mathsf{DDH}} = |\mathsf{Prob}[(A,B,C) \longleftarrow \Delta_{\mathsf{Real}} : \mathcal{D}(A,B,C) = 1] - \mathsf{Prob}[(A,B,C) \longleftarrow \Delta_{\mathsf{Rand}} : \mathcal{D}(A,B,C) = 1]|$. The probability is taken over the choice of $\log_g A, \log_g B, \log_g C$ and $\mathcal{D}$'s coin tosses. $\mathcal{D}$ is said to be a (t,ϵ)-DDH distinguisher for G if $\mathcal{D}$ runs in time at most t such that $\mathsf{Adv}_{\mathcal{D},G}^{\mathsf{DDH}}(t) \geq \epsilon$.

DDH assumption : There exists no (t,ϵ)-DDH distinguisher for G. In other words, for every probabilistic, polynomial-time, 0/1-valued distinguisher $\mathcal{D}$, $\mathsf{Adv}_{\mathcal{D},G}^{\mathsf{DDH}} \leq \epsilon$ for sufficiently small $\epsilon > 0$.

2.2 Security Model

We describe below the adversarial model following Bresson *et al.*'s [15] formal security model that we adopt for the security analysis of our protocols. This model is more general in the sense that it covers authenticated key agreement in group setting and suited for dynamic groups.

Let $\mathcal{P} = \{U_1, \ldots, U_n\}$ be a set of n (fixed) users or participants. At any point of time, any subset of $\mathcal{P}$ may decide to establish a session key. Thus a user can execute the protocol for group key agreement several times withdifferent partners, can join or leave the group at his desire by executing the protocols for Join or Leave. We identify the execution of protocols for key agreement, member(s) join and member(s) leave as different sessions. The adversarial model consists of allowing each user an unlimited number of instances with which it executes the protocol for key agreement or inclusion or exclusion of a user or a set of users. We assume adversary never participates as a user in the protocol. This adversarial model allows concurrent execution of the protocol. The interaction between the adversary $\mathcal{A}$ and the protocol participants occur only via oracle queries, which model the adversary's capabilities in a real attack. Let S, S_1, S_2 be three sets defined as:

$$S = \{(V_1, i_1), \ldots, (V_l, i_l)\}, S_1 = \{(V_{l+1}, i_{l+1}), \ldots, (V_{l+k}, i_{l+k})\}, S_2 = \{(V_{j_1}, i_{j_1}), \ldots, (V_{j_k}, i_{j_k})\}$$

where $\{V_1, \ldots, V_l\}$ is any non-empty subset of $\mathcal{P}$. We will require the following notations.

Π^i_U	:	i-th instance of user U.
sk^i_U	:	session key after execution of the protocol by Π^i_U.
sid^i_U	:	session identity for instance Π^i_U. We set $\mathsf{sid}^i_U = S = \{(U_1, i_1), \ldots, (U_k, i_k)\}$ such that $(U, i) \in S$ and $\Pi^{i_1}_{U_1}, \ldots, \Pi^{i_k}_{U_k}$ wish to agree upon a common key.
pid^i_U	:	partner identity for instance Π^i_U, defined by $\mathsf{pid}^i_U = \{U_1, \ldots, U_k\}$, such that $(U_j, i_j) \in \mathsf{sid}^i_U$ for all $1 \leq j \leq k$.
acc^i_U	:	0/1-valued variable which is set to be 1 by Π^i_U upon normal termination of the session and 0 otherwise.

We will make the assumption that in each session at most one instance of each user participates. Further, an instance of a particular user participates in exactly one session. This is not a very restrictive assumption, since a user can spawn an instance for each session it participates in. On the other hand, there is an important consequence of this assumption. Suppose there are several sessions which are being concurrently executed. Let the session ID's be $\mathsf{sid}_1, \ldots, \mathsf{sid}_k$. Then for any instance Π^i_U, there is exactly one j such that $(U, i) \in \mathsf{sid}_j$ and for any $j_1 \neq j_2$, we have $\mathsf{sid}_{j_1} \cap \mathsf{sid}_{j_2} = \emptyset$. Thus at any particular point of time, if we consider the collection of all instances of all users, then the relation of being in the same session is an equivalence relation whose equivalence classes are the session IDs.

We assume that the adversary has complete control over all communications in the network. All information that the adversary gets to see is written in a transcript. So a transcript consists of all the public information flowing across the network. The following oracles model an adversary's interaction with the users in the network:

- $\mathsf{Send}(U, i, m)$: This query models an active attack, in which the adversary may intercept a message and then either modify it, create a new one or simply forward it to the intended participant. The output of the query is the reply (if any) generated by the instance Π^i_U upon receipt of message m. The adversary is allowed to prompt the unused instance Π^i_U to initiate the protocol with partners $U_2, \ldots, U_l, l \leq n$, by invoking $\mathsf{Send}(U, i, \langle U_2, \ldots, U_l \rangle)$.

– $\mathsf{Execute}(S)$: This query models passive attacks in which the attacker eavesdrops on honest execution of group key agreement protocol among unused instances $\Pi_{V_1}^{i_1}, \ldots, \Pi_{V_l}^{i_l}$ and outputs the transcript of the execution. A transcript consists of the messages that were exchanged during the honest execution of the protocol.

– $\mathsf{Join}(S, S_1)$: This query models the insertion of user instances $\Pi_{V_{l+1}}^{i_{l+1}}, \ldots, \Pi_{V_{l+k}}^{i_{l+k}}$ in the group $\{V_1, \ldots, V_l\} \subset \mathcal{P}$ for which Execute have already been queried. The output of this query is the transcript generated by the invocation of algorithm Join. If $\mathsf{Execute}(S)$ has not taken place, then the adversary is given no output.

– $\mathsf{Leave}(S, S_2)$: This query models the removal of user instances $\Pi_{V_{j_1}}^{i_{j_1}}, \ldots, \Pi_{V_{j_k}}^{i_{j_k}}$ from the group $\{V_1, \ldots V_l\} \subset \mathcal{P}$. If $\mathsf{Execute}(S)$ has not taken place, then the adversary is given no output. Otherwise, algorithm Leave is invoked. The adversary is given the transcript generated by the honest execution of procedure Leave.

– $\mathsf{Reveal}(U, i)$: This outputs session key sk_U^i. This query models the misuse of the session keys, *i.e* known session key attack.

– $\mathsf{Corrupt}(U)$: This outputs the long-term secret key (if any) of player U. The adversarial model that we adopt is a weak-corruption model in the sense that only the long-term secret keys are compromised, but the ephemeral keys or the internal data of the protocol participants are not corrupted. This query models (perfect) forward secrecy.

– $\mathsf{Test}(U, i)$: This query is allowed only once, at any time during the adversary's execution. A bit $b \in \{0, 1\}$ is chosen uniformly at random. The adversary is given sk_U^i if $b = 1$, and a random session key if $b = 0$. This oracle computes the adversary's ability to distinguish a real session key from a random one.

An adversary which has access to the Execute, Join, Leave, Reveal, Corrupt and Test oracles, is considered to be passive while an active adversary is given access to the Send oracle in addition. (For static case, there is no Join or Leave queries as a group of fixed size is considered.)

The adversary can ask Send, Execute, Join, Leave, Reveal and Corrupt queries several times, but Test query is asked only once and on a fresh instance. We say that an instance Π_U^i is *fresh* unless either the adversary, at some point, queried $\mathsf{Reveal}(U, i)$ or $\mathsf{Reveal}(U', j)$ with $U' \in \mathsf{pid}_U^i$ or the adversary queried $\mathsf{Corrupt}(V)$ (with $V \in \mathsf{pid}_U^i$) before a query of the form $\mathsf{Send}(U, i, *)$ or $\mathsf{Send}(U', j, *)$ where $U' \in \mathsf{pid}_U^i$.

Finally adversary outputs a guess bit b'. Such an adversary is said to win the game if $b = b'$ where b is the hidden bit used by the Test oracle.

Let Succ denote the event that the adversary $\mathcal{A}$ wins the game for a protocol XP. We define

$$\mathsf{Adv}_{\mathcal{A},\mathsf{XP}} := |2 \ \mathsf{Prob}[\mathsf{Succ}] - 1|$$

to be the advantage of the adversary $\mathcal{A}$ in attacking the protocol XP.

The protocol XP is said to be a *secure unauthenticated group key agreement* (KA) protocol if there is no polynomial time *passive* adversary with non-negligible advantage. In other words, for every probabilistic, polynomial-time, 0/1 valued algorithm $\mathcal{A}$, $\mathsf{Adv}_{\mathcal{A},\mathsf{XP}} < \frac{1}{M^L}$ for every fixed $L > 0$ and sufficiently large integer M. We say that protocol XP is a *secure authenticated group key agreement* (AKA) protocol if there is no polynomial time *active* adversary with non-negligible advantage. Next we define

$\mathsf{Adv}^{\mathsf{KA}}_{\mathsf{XP}}(t, q_E)$:= the maximum advantage of any passive adversary attacking protocol XP, running in time t and making q_E calls to the Execute oracle.

$\mathsf{Adv}^{\mathsf{AKA}}_{\mathsf{XP}}(t, q_E, q_S)$:= the maximum advantage of any active adversary attacking protocol XP, running in time t and making q_E calls to the Execute oracle and q_S calls to the Send oracle.

$\mathsf{Adv}^{\mathsf{AKA}}_{\mathsf{XP}}(t, q_E, q_J, q_L, q_S)$:= the maximum advantage of any active adversary attacking protocol XP, running in time t and making q_E calls to the Execute oracle, q_J calls to Join oracle, q_L calls to the Leave oracle and q_S calls to the Send oracle.

3 Protocol

Suppose a set of n users $\mathcal{P} = \{U_1, \ldots, U_n\}$ wish to establish a common session key among themselves. Quite often, we identify a user U_i with it's instance $\Pi^{d_i}_{U_i}$ (for some integer d_i that is session specific) during a protocol execution. We consider the users $U_1, \ldots, U_n$ participating in the protocol are on a ring and U_{i-1}, U_{i+1} are respectively the left and right neighbors of U_i for $1 \leq i \leq n$, $U_0 = U_n, U_{n+1} = U_1$ and U_{n+i} is taken to be U_i. As mentioned earlier, we consider a multiplicative group G of some large prime order q with g as a generator. We also consider a hash function $\mathcal{H} : \{0,1\}^* \to Z_q^*$.

3.1 Unauthenticated Key Agreement Protocol

$$\begin{array}{cccccl} U_1 & U_2 & U_3 & U_4 & U_5 & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \\ x_1 & x_2 & x_3 & x_4 & x_5 & \\ g^{x_1} & g^{x_2} & g^{x_3} & g^{x_4} & g^{x_5} & : \text{Round-1} \end{array}$$

$$\text{Communications} \longrightarrow U_i \text{ sends } g^{x_i} \text{ to } U_{i-1}, U_{i+1},\ 1 \leq i \leq 5,\ U_0 = U_5, U_6 = U_1.$$

$$U_i \text{ computes } K_i^L = g^{x_{i-1}x_i},\ K_i^R = g^{x_i x_{i+1}},\ 1 \leq i \leq 5,\ x_0 = x_5, x_6 = x_1$$

$$\begin{array}{cccccl} \frac{K_1^R}{K_1^L} & \frac{K_2^R}{K_2^L} & \frac{K_3^R}{K_3^L} & \frac{K_4^R}{K_4^L} & \frac{K_5^R}{K_5^L} & : \text{Round-2} \end{array}$$

$$\text{Communications} \longrightarrow U_i,\ 1 \leq i \leq 5 \text{ sends } \frac{K_i^R}{K_i^L} \text{ to } U_j,\ 1 \leq j \leq 5, j \neq i$$

$$U_i,\ 1 \leq i \leq 5 \text{ recovers } K_j^R,\ 1 \leq j \leq 5, j \neq i$$
$$\text{The session key } \mathsf{sk} = K_1^R K_2^R K_3^R K_4^R K_5^R = g^{x_1x_2+x_2x_3+x_3x_4+x_4x_5+x_5x_1}$$

Figure 1: The unauthenticated group key agreement among $n = 5$ users.

First we informally describe our unauthenticated protocol KeyAgree that involves two rounds and a key computation phase. At the start of the session, each user $U_i = \Pi_{U_i}^{d_i}$ chooses randomly a private key $x_i \in Z_q^*$. In the first round, U_i computes $X_i = g^{x_i}$ and sends X_i to its neighbors U_{i-1}, U_{i+1}. After this communication is over, U_i receives X_{i-1} from U_{i-1} and X_{i+1} from U_{i+1}. U_i then computes it's left key $K_i^L = X_{i-1}^{x_i}$, right key $K_i^R = X_{i+1}^{x_i}$, $Y_i = K_i^R / K_i^L$ and sends Y_i to the rest of the users in the second round. Finally in the key computation phase, U_i computes $\overline{K}_{i+1}^R, \overline{K}_{i+2}^R, \ldots, \overline{K}_{i+(n-1)}^R$ as follows making use of his own right key K_i^R: $\overline{K}_{i+1}^R = Y_{i+1} K_i^R, \overline{K}_{i+2}^R = Y_{i+2}\overline{K}_{i+1}^R, \ldots, \overline{K}_{i+(n-1)}^R = Y_{i+(n-1)}\overline{K}_{i+(n-2)}^R$. Then U_i verifies if $\overline{K}_{i+(n-1)}^R$ is same as that of his left key $K_i^L (= K_{i+(n-1)}^R)$. If verification fails, then U_i aborts. Otherwise, U_i has the correct right keys of all the users. U_i computes the session key $\mathsf{sk}_{U_i}^{d_i} = \overline{K}_1^R \overline{K}_2^R \ldots \overline{K}_n^R$ which is equal to $g^{x_1x_2 + x_2x_3 + \cdots + x_nx_1}$. U_i also computes and stores $x = \mathcal{H}(\mathsf{sk}_{U_i}^{d_i})$ for a join operation and stores his left key and right key K_i^L, K_i^R respectively for a leave operation as we will see in the subsequent subsections. We refer x as the seed which is common to all users involved in the session. Figure 1 illustrates the protocol with $n = 5$ users.

Observe that each user computes 3 exponentiations (1 in round 1 and 2 in round 2) and at most $2n-2$ multiplications ($n-1$ multiplications for recovery of all right keys and $n-1$ multiplications for session key computation). The formal description of the protocol is given below.

procedure KeyAgree($U[1, \ldots, n], x[1, \ldots, n]$)
    (Round 1):
for $i = 1$ to n **do in parallel**
    $U_i (= \Pi_{U_i}^{d_i})$ computes $X_i = g^{x_i}$ and sends X_i to U_{i-1} and U_{i+1} ;
end for
Note that $X_0 = X_n$ and $X_{n+1} = X_1$.
    (Round 2):
for $i = 1$ to n **do in parallel**
    U_i computes the left key $K_i^L = X_{i-1}^{x_i}$, the right key $K_i^R = X_{i+1}^{x_i}$ and $Y_i = K_i^R / K_i^L$;
    U_i sends Y_i to the rest of the users;
end for
Note that $K_i^R = K_{i+1}^L$ for $1 \le i \le n-1$, $K_n^R = K_1^L$ and $K_{i+(n-1)}^R = K_i^L$.
    (Key Computation):
for $i = 1$ to n **do in parallel**
U_i computes $\overline{K}_{i+1}^R = Y_{i+1} K_i^R$;
    for $j = 2$ to $n-1$ **do**
        U_i computes $\overline{K}_{i+j}^R = Y_{i+j}\overline{K}_{i+(j-1)}^R$;
    end for
    U_i verifies if $K_{i+(n-1)}^R = \overline{K}_{i+(n-1)}^R$ (*i.e.* if $K_i^L = \overline{K}_{i+(n-1)}^R$);
    if verification fails, **then** U_i sets $\mathsf{acc}_{U_i}^{d_i} = 0$, $\mathsf{sk}_{U_i}^{d_i} = \mathsf{NULL}$ and aborts the protocol;
    else U_i computes the session key $\mathsf{sk}_{U_i}^{d_i} = \overline{K}_1^R \overline{K}_2^R \ldots \overline{K}_n^R$, the seed $x = \mathcal{H}(\mathsf{sk}_{U_i}^{d_i})$ and stores K_i^L, K_i^R;
    end if
end for
end KeyAgree

3.2 Authenticated Key Agreement Protocol

We authenticate the unauthenticated protocol of Section 3.1 by incorporating a standard digital signature scheme $\mathsf{DSig} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ where $\mathcal{K}$ is the key generation algorithm, $\mathcal{S}$ is the signature generation algorithm and $\mathcal{V}$ is the signature verification algorithm. As part of this signature scheme, $\mathcal{K}$ generates a signing and a verification key sk_i (or sk_{U_i}) and pk_i (or pk_{U_i}) respectively for each user U_i. Session identity is an important issue of our authentication mechanism which uniquely identifies the session and is same for all instances participating in the session.

Suppose instances $\Pi^{d_1}_{U_{i_1}}, \ldots, \Pi^{d_k}_{U_{i_k}}$ wish to agree upon a common key in a session. Then according to our definition, $\mathsf{sid}^{d_j}_{U_{i_j}} = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$. Note that the instance numbers can be easily generated using counter. We make the assumption that in each session at most one instance of each user participates and an instance of a particular user participates in exactly one session. As mentioned earlier, this is a reasonable assumption to avoid collisions in the session identities.

At the start of the session, $\Pi^{d_j}_{U_{i_j}}$ need not to know the entire set $\mathsf{sid}^{d_j}_{U_{i_j}}$. This set is built up as the protocol proceeds. We use a variable partial session-identity psid^{d}_{U} for instance Π^{d}_{U} involved in a session to keep the partial information about it's session identity. Initially, $\mathsf{psid}^{d_j}_{U_{i_j}}$ is set to be $\{(U_{i_j}, d_j)\}$ by $\Pi^{d_j}_{U_{i_j}}$ and finally after completion of the session, $\mathsf{psid}^{d_j}_{U_{i_j}}$ grow into full session identity $\mathsf{sid}^{d_j}_{U_{i_j}}$. We assume that any instance $\Pi^{d_j}_{U_{i_j}}$ knows it's partner identity $\mathsf{pid}^{d_j}_{U_{i_j}}$ *i.e.* the set of users with which it is partnered in the particular session. We describe below the algorithm $\mathsf{AuthKeyAgree}$ that is obtained by modifying algorithm $\mathsf{KeyAgree}$ by introducing signatures in the communication.

procedure $\mathsf{AuthKeyAgree}(U[1, \ldots, n], x[1, \ldots, n])$
  (Round 1):
for $i = 1$ to n **do in parallel**
  $U_i (= \Pi^{d_i}_{U_i})$ sets its partial session-identity $\mathsf{psid}^{d_i}_{U_i} = \{(U_i, d_i)\}$;
  U_i chooses randomly $x_i \in Z^*_q$ and computes $X_i = g^{x_i}$ and $\sigma_i = \mathcal{S}(sk_{U_i}, M_i)$ where $M_i = U_i|1|X_i$;
  U_i sends $M_i|\sigma_i$ to U_{i-1} and U_{i+1};
end for
Note that $M_0|\sigma_0 = M_n|\sigma_n$ and $M_{n+1}|\sigma_{n+1} = M_1|\sigma_1$.
  (Round 2):
for $i = 1$ to n **do in parallel**
  U_i, on receiving $M_{i-1}|\sigma_{i-1}$ from U_{i-1} and $M_{i+1}|\sigma_{i+1}$ from U_{i+1}, verifies σ_{i-1} on M_{i-1} and σ_{i+1} on M_{i+1} using the verification algorithm $\mathcal{V}$ and the respective verification keys $pk_{U_{i-1}}$, $pk_{U_{i+1}}$;
  if verification fails, **then** U_i sets $\mathsf{acc}^{d_i}_{U_i} = 0$, $\mathsf{sk}^{d_i}_{U_i} = \mathsf{NULL}$ and aborts;
  else U_i computes the left key $K^L_i = X^{x_i}_{i-1}$, the right key $K^R_i = X^{x_i}_{i+1}$, $Y_i = K^R_i / K^L_i$ and signature $\overline{\sigma}_i = \mathcal{S}(sk_{U_i}, \overline{M}_i)$ where $\overline{M}_i = U_i|2|Y_i|d_i$;
  U_i sends $\overline{M}_i|\overline{\sigma}_i$ to the rest of the users;
  end if
end for
Note that $K^R_i = K^L_{i+1}$ for $1 \leq i \leq n-1$, $K^R_n = K^L_1$ and $K^R_{i+(n-1)} = K^L_i$.
  (Key Computation):
for $i = 1$ to n **do in parallel**
  for $j = 1$ to n, $j \neq i$ **do**
    U_i, on receiving $\overline{M}_j|\overline{\sigma}_j$ from U_j verifies $\overline{\sigma}_j$ on $\overline{M}_j$ using

the verification algorithm $\mathcal{V}$ and the verification key pk_{U_j};
if verification fails, **then** U_i sets $\mathsf{acc}_{U_i}^{d_i} = 0$, $\mathsf{sk}_{U_i}^{d_i} = \mathsf{NULL}$ and aborts;
else U_i extracts d_j from $\overline{M}_j$ and sets $\mathsf{psid}_{U_i}^{d_i} = \mathsf{psid}_{U_i}^{d_i} \cup \{(U_j, d_j)\}$;
end for
U_i computes $\overline{K}_{i+1}^R = Y_{i+1} K_i^R$;
$j = 2$ to $n-1$ **do**
U_i computes $\overline{K}_{i+j}^R = Y_{i+j} \overline{K}_{i+(j-1)}^R$;
end for
U_i verifies if $K_{i+(n-1)}^R = \overline{K}_{i+(n-1)}^R$ (*i.e.* if $K_i^L = \overline{K}_{i+(n-1)}^R$);
if verification fails, **then** U_i sets $\mathsf{acc}_{U_i}^{d_i} = 0$, $\mathsf{sk}_{U_i}^{d_i} = \mathsf{NULL}$ and aborts;
else U_i computes the session key $\mathsf{sk}_{U_i}^{d_i} = \overline{K}_1^R \overline{K}_2^R \ldots \overline{K}_n^R$, the seed $x = \mathcal{H}(\mathsf{sk}_{U_i}^{d_i})$ and stores K_i^L, K_i^R;
end if
end if
end for
end AuthKeyAgree

3.3 Dynamic Key Agreement Protocol

3.3.1 Join

Suppose $U[1,\ldots,n]$ be a set of users with respective secret keys $x[1,\ldots,n]$ and an execution of AuthKeyAgree among the instances $\Pi_{U_1}^{t_1},\ldots,\Pi_{U_n}^{t_n}$ has already been done. So all these instances $\Pi_{U_i}^{t_i}, 1 \le i \le n$, have a common session key and also a common seed $x \in Z_q^*$ resulting from this execution of AuthKeyAgree. Let the set of users $U[n+1,\ldots,n+m]$ with secret keys $x[n+1,\ldots,n+m]$ want to join the group $U[1,\ldots,n]$. The new instances involved in the procedure Join are $\Pi_{U_1}^{d_1},\ldots,\Pi_{U_{n+m}}^{d_{n+m}}$.

We consider a ring of $l = m+3$ users $V_1 = U_1$, $V_2 = U_2$, $V_3 = U_n$, $V_i = U_{n+i-3}$ for $4 \le i \le l$ with V_2 now using the seed x as it's private key. We set $y_1 = x_1, y_2 = x, y_3 = x_n$, $y_i = x_{n+i-3}$ and $\hat{d}_1 = d_1, \hat{d}_2 = d_2, \hat{d}_3 = d_n$, $\hat{d}_i = d_{n+i-3}$. The left and right neighbors of V_i are respectively V_{i-1} and V_{i+1} for $1 \le i \le l$ with $V_0 = V_l$ and $V_{l+1} = V_1$. We take V_{l+i} to be V_i and V_2 is the representative of the set of users $U[2,\ldots,n-1]$. We invoke KeyAgree (for unauthenticated version of join algorithm) or AuthKeyAgree (for authenticated version of join algorithm) for l users $V[1,\ldots,l]$ with respective keys $y[1,\ldots,l]$. For simplicity, we describe the unauthenticated version of the precedure Join and mention the additional modifications required for it's authenticated version.

Let for $1 \le i \le l$, $\hat{X}_i = g^{y_i}$; $\hat{X}_0 = \hat{X}_l, \hat{X}_{l+1} = \hat{X}_1$; $\hat{K}_i^L = \hat{X}_{i-1}^{y_i}; \hat{K}_i^R = \hat{X}_{i+1}^{y_i}; \hat{Y}_i = \hat{K}_i^R / \hat{K}_i^L$. In round 1, V_i sends $\hat{X}_i$ to both V_{i-1} and V_{i+1}. Additionally, V_1 sends $\hat{X}_1$ and V_3 sends $\hat{X}_3$ to all users $U[3,\ldots,n-1]$ in this round. In the second round, V_i computes it's left key $\hat{K}_i^L$, right key $\hat{K}_i^R$ and sends $\hat{Y}_i$ to the rest of the users in $V[1,\ldots,l]$. Additionally, V_i sends $\hat{Y}_i$ to all users in $U[3,\ldots,n-1]$. If the protocol does not abort, V_i computes the session key $\mathsf{sk}_{V_i}^{\hat{d}_i}$ in the key computation phase which is the product of l right keys corresponding to l users $V[1,\ldots,l]$. V_i also computes the seed $\mathcal{H}(\mathsf{sk}_{V_i}^{\hat{d}_i})$ and stores $\hat{K}_i^L, \hat{K}_i^R$ that can be used for subsequent dynamic operations. Although active participations of the users $U[3,\ldots,n-1]$ are not required during the protocol execution, these users should be able to compute the common session key, the seed, the left key and the right key. Fortunately, these users have x, $\hat{X}_1 = g^{y_1}$ and $\hat{X}_3 = g^{y_3}$. So each can compute and store U_2's left key $\hat{K}_2^L = g^{y_1 x}$, right key $\hat{K}_2^R = g^{y_3 x}$ and proceeding in the same way as V_2 does, recover right keys of l users $V[1,\ldots,l]$, computes the session key and the common seed. The joining algorithm Join is fomally described below.

procedure $\mathsf{Join}(U[1,\ldots,n+m],x[1,\ldots,n+m])$
1. Set $l=m+3$; $V_1=U_1,V_2=U_2,V_3=U_n;\hat{d}_1=d_1,\hat{d}_2=d_2,\hat{d}_3=d_n;y_1=x_1,y_2=x,y_3=x_n$;
   and for $4\le i\le l$, $V_i=U_{n+i-3};\hat{d}_i=d_{n+i-3};y_i=x_{n+i-3}$;
2. We consider a ring of l users $V[1,\ldots,l]$ with respective instance numbers $\hat{d}[1,\ldots,l]$
   and secret keys $y[1,\ldots,l]$;
3. **call** $\mathsf{KeyAgree}(V[1,\ldots,l],y[1,\ldots,l])$;
4. Let for $1\le i\le l$, $\hat{X}_i=g^{y_i}$; $\hat{X}_0=\hat{X}_l,\hat{X}_{l+1}=\hat{X}_1$; $\hat{K}_i^L=\hat{X}_{i-1}^{y_i};\hat{K}_i^R=\hat{X}_{i+1}^{y_i};\hat{Y}_i=\hat{K}_i^R/\hat{K}_i^L$;
5. V_1 and V_3, in round 1, additionally send $\hat{X}_1$ and $\hat{X}_3$ respectively to all users in $U[3,\ldots,n-1]$;
6. V_i, in round 2, additionally sends $\hat{Y}_i$ to all users in $U[3,\ldots,n-1]$;
7. **for** $i=3$ to $n-1$ **do**
8. U_i computes $\hat{K}_3^R=\hat{Y}_3K_2^R$;
9. $j=2$ to $l-1$ **do**
10. U_i computes $\hat{K}_{2+j}^R=\hat{Y}_{2+j}\hat{K}_{2+(j-1)}^R$;
11.   **end do**
12. U_i computes $\mathsf{sk}_{U_i}^{d_i}=\hat{K}_1^R\hat{K}_2^R\ldots\hat{K}_l^R$;
13. **end for**

end Join

If we invoke procedure AuthKeyAgree instead of KeyAgree in line 3 of the above algorithm, then messages transmitted during the protocol execution are properly structured with signatures appended to them generated and verified according to the algorithm AuthKeyAgree. At the end of the session, if the protocol terminates normally without abort, then each user V_i, $1\le i\le l$ additionally has a common session identity $\mathsf{sid}_{V_i}^{\hat{d}_i}=\{(V_1,\hat{d}_1),\ldots,(V_l,\hat{d}_l)\}$ apart from the common session key, the seed, the left and the right keys. Users $U[3,\ldots,n-1]$ are also able to compute this session identity from the messages received by them during the protocol execution.

3.3.2 Leave

Suppose $U[1,\ldots,n]$ is a set of users with respective secret keys $x[1,\ldots,n]$ and an execution of AuthKeyAgree among the instances $\Pi_{U_1}^{t_1},\ldots,\Pi_{U_n}^{t_n}$ has already been done. Let K_i^L,K_i^R, $1\le i\le n$ are the left and right keys respectively of U_i computed and stored in this session. Let the set of users $\{U_{l_1},\ldots,U_{l_m}\}$ wants to leave the group $U[1,\ldots,n]$. Then the new user set is $U[1,\ldots,l_1-L]\cup U[l_1+R,\ldots,l_2-L]\cup\ldots\cup U[l_m+R,\ldots,n]$ where U_{l_i-L} and U_{l_i+R} are respectively the left and right neighbours of the leaving user U_{l_i}, $1\le i\le m$. Then for any leaving user U_l, $l-L=l-i$ if the consecutive users $U_l,U_{l-1},\ldots,U_{l-(i-1)}$ are all leaving and U_{l-i} is not leaving the group. Similarly, $l+R=l+i$ if consecutive users $U_l,U_{l+1},\ldots,U_{l+(i-1)}$ are all leaving and U_{l+i} is not leaving the group. We reindex these $n-m$ remaining users and denote the new user set by $V[1,\ldots,n-m]$. We also reindex the left and right keys and denote by two arrays $\hat{K}^L[1,\ldots,n-m]$ and $\hat{K}^R[1,\ldots,n-m]$ respectively the left and right keys of users $V[1,\ldots,n-m]$. The new instances involved in the procedure Leave are $\Pi_{V_1}^{d_1},\ldots,\Pi_{V_{n-m}}^{d_{n-m}}$.

We consider a ring of $n-m$ users $V[1,\ldots,n-m]$. For a leaving user U_{l_i}, it's left neighbor U_{l_i-L} and right neighbor U_{l_i+R} respectively choose new secret keys $x_{j_1},x_{j_2}\in Z_q^*$ where $j_1=l_i-L$ and $j_2=l_i+R$, computes $X_{j_1}=g^{x_{j_1}},X_{j_2}=g^{x_{j_2}}$. Note that in the ring, the left and right neighbors of U_{j_1} are respectively U_{j_1-1} and U_{j_2} and that of U_{j_2} are respectively U_{j_1} and U_{j_2+1}. U_{j_1} sends X_{j_1} (properly structured with corresponding signature as in AuthKeyAgree) to it's neighbors U_{j_1-1}, U_{j_2} and U_{j_2} sends X_{j_2} (properly structured) to it's neighbors U_{j_1}, U_{j_2+1}. This is the first round. In the second round, each user V_i, after proper verification of the received messages, computes $Y_i=\hat{K}_i^R/\hat{K}_i^L$ and sends Y_i (properly structured associating signature) to the rest of the users in $V[1,\ldots,n-m]$. The key computation phase is exactly the same as in the procedure AuthKeyAgree among $n-m$ users $V_1,\ldots.V_{n-m}$. The algorithm Leave is formally described below.

procedure $\mathsf{Leave}(U[1,\ldots,n], x[1,\ldots,n], \{U_{l_1},\ldots,U_{l_m}\})$
(Round 1):
Let K_i^L, K_i^R be respectively the left and right keys of user U_i, $1 \le i \le n$, computed and stored in a previous session among instances $\Pi_{U_1}^{t_1},\ldots,\Pi_{U_n}^{t_n}$.
for $i=1$ to m **do in parallel**
  Let $j_1 = l_i - L; j_2 = l_i + R$;
  U_{j_1}, U_{j_2} respectively choose randomly new secret keys $x_{j_1}, x_{j_2} \in Z_q^*$ and computes $X_{j_1} = g^{x_{j_1}}$, $X_{j_2} = g^{x_{j_2}}$ and $\sigma_{j_1} = \mathcal{S}(sk_{U_{j_1}}, M_{j_1})$, $\sigma_{j_2} = \mathcal{S}(sk_{U_{j_2}}, M_{j_2})$ where $M_{j_1} = U_{j_1}|1|X_{j_1}$, $M_{j_2} = U_{j_2}|1|X_{j_2}$;
  U_{j_1} sends $M_{j_1}|\sigma_{j_1}$ to U_{j_1-1} and U_{j_2};
  U_{j_2} sends $M_{j_2}|\sigma_{j_2}$ to U_{j_1} and U_{j_2+1} $(U_{n+1} = U_1)$;
end for
(Round 2):
for $i=1$ to m **do in parallel**
  Let $j_1 = l_i - L, j_2 = l_i + R$;
  We set $W = \{j_1-1, j_1, j_2, j_2+1\}$;
  U_{j_1-1}, U_{j_2}, on receiving $M_{j_1}|\sigma_{j_1}$ from U_{j_1}, verifies σ_{j_1} on M_{j_1} using the verification key $pk_{U_{j_1}}$;
  U_{j_1}, U_{j_2+1}, on receiving $M_{j_2}|\sigma_{j_2}$ from U_{j_2}, verifies σ_{j_2} on M_{j_2} using the verification key $pk_{U_{j_2}}$;
  if any of these verifications fail, **then** U_w, $w \in W$, sets $\mathsf{acc}_{U_w}^{d_w} = 0$, $\mathsf{sk}_{U_w}^{d_w} = \mathsf{NULL}$ and aborts;
  else
    U_{j_1} modifies its left key $K_{j_1}^L = X_{j_1-1}^{x_{j_1}}$ and right key $K_{j_1}^R = X_{j_2}^{x_{j_1}}$;
    U_{j_2} modifies its left key $K_{j_1}^L = X_{j_1}^{x_{j_2}}$ and right key $K_{j_2}^R = X_{j_2+1}^{x_{j_2}}$;
    U_{j_1-1} modifies its right key $K_{j_1-1}^R = X_{j_1}^{x_{j_1-1}}$;
    U_{j_2+1} modifies its left key $K_{j_2+1}^L = X_{j_2}^{x_{j_2+1}}$;
  end if
end for
We reindex the $n-m$ users $U[1\ldots n] \setminus \{U_{l_1},\ldots,U_{l_m}\}$. Let $U[1\ldots n-m]$ be the new user set and $\hat{K}^L[1\ldots n-m]$, $\hat{K}^R[1\ldots n-m]$ respectively be the set of corresponding left and right keys.
for $i=1$ to $n-m$ **do in parallel**
  V_i computes $Y_i = \hat{K}_i^R / \hat{K}_i^L$ and signature $\hat{\sigma}_i = \mathcal{S}(sk_{V_i}, \hat{M}_i)$ where $\hat{M}_i = V_i|2|Y_i|d_i$;
  V_i sends $\hat{M}_i|\hat{\sigma}_i$ to the rest of the users in $V[1,\ldots,n-m]$;
end for
Note that $\hat{K}_i^R = \hat{K}_{i+1}^L$ for $1 \le i \le n-m-1$, $\hat{K}_n^R = \hat{K}_1^L$ and $\hat{K}_{i+(n-m-1)}^R = \hat{K}_i^L$.
(Key Computation):
for $i=1$ to $n-m$ **do in parallel**
  for $j=1$ to $n-m$, $j \ne i$ **do**
    V_i, on receiving $\overline{M}_j|\overline{\sigma}_j$ from V_j verifies $\overline{\sigma}_j$ on $\overline{M}_j$ using the verification algorithm $\mathcal{V}$ and the verification key pk_{V_j};
    if verification fails, **then** V_i sets $\mathsf{acc}_{V_i}^{d_i} = 0$, $\mathsf{sk}_{V_i}^{d_i} = \mathsf{NULL}$ and aborts;
    else V_i extracts d_j from $\overline{M}_j$ and sets $\mathsf{psid}_{V_i}^{d_i} = \mathsf{psid}_{V_i}^{d_i} \cup \{(V_j, d_j)\}$;
  end for
  V_i computes $\overline{K}_{i+1}^R = Y_{i+1}\hat{K}_i^R$;

$j = 2$ to $n - m - 1$ **do**
  V_i computes $\overline{K}^R_{i+j} = Y_{i+j}\overline{K}^R_{i+(j-1)}$;
end for
V_i verifies if $\hat{K}^R_{i+(n-m-1)} = \overline{K}^R_{i+(n-m-1)}$ (*i.e.* if $\hat{K}^L_i = \overline{K}^R_{i+(n-m-1)}$);
if verification fails, **then** V_i sets $\mathsf{acc}^{d_i}_{V_i} = 0$, $\mathsf{sk}^{d_i}_{V_i} = \mathsf{NULL}$ and aborts;
else V_i computes the session key $\mathsf{sk}^{d_i}_{V_i} = \overline{K}^R_1\overline{K}^R_2 \ldots \overline{K}^R_{n-m}$, the seed $x = \mathcal{H}(\mathsf{sk}^{d_i}_{V_i})$ and stores $\hat{K}^L_i, \hat{K}^R_i$;
end if
end if
end for
end Leave

4 Security Analysis

4.1 Security of the Unauthenticated Protocol

We will show that our unauthenticated protocol UP is secure against passive adversary under DDH assumption. We state the security result of UP in Theorem 4.1. The proof, although not exactly same, is quite similar to Katz-Yung [25] proof of security against passive adversary of the unauthenticated BD [18] protocol under DDH assumption.

Theorem 4.1 *The unauthenticated protocol* UP *described in Section 3.1 is secure against passive adversary under DDH assumption, achieves forward secrecy and satisfies the following:*

$$\mathsf{Adv}^{\mathsf{KA}}_{\mathsf{UP}}(t, q_E) \le 4\ \mathsf{Adv}^{\mathsf{DDH}}_{G}(t') + \frac{8q_E}{|G|}$$

where $t' = t + O(|\mathcal{P}|\ q_E\ t_{exp})$, t_{exp} *is the time required to perform exponentiation in* G *and* q_E *is the number of* Execute *query that an adversary may ask.*

Proof : Let $\mathcal{A}$ be an adversary for the unauthenticated protocol UP. Using this, we can construct an algorithm $\mathcal{D}$ which solves the DDH problem with non-negligible advantage. We first consider that the adversary $\mathcal{A}$ makes a single Execute query. The number of parties n (≥ 3) among which the adversary $\mathcal{A}$ asks Execute query is chosen by $\mathcal{A}$ itself. Moreover, since we do not use any long term secret key in our protocol UP, Corrupt query may simply be ignored for $\mathcal{A}$ and the protocol trivially achieves forward secrecy. The adversary $\mathcal{A}$ has access to three oracles: Execute, Reveal and Test. To deal with the Execute and Reveal query, we define distributions Real and Fake$'$ for transcript, session key pair (T, sk) as follows where Real is the real execution scenario of the protocol UP and prove the Claim 1 stated below.

$$\mathsf{Real} := \left\{ \begin{array}{ll} x_1, \ldots x_n \longleftarrow Z^*_q; & \\ X_1 = g^{x_1}, X_2 = g^{x_2}, \ldots, X_n = g^{x_n}; & \\ K^R_1 = K^L_2 = g^{x_1x_2}, K^R_2 = K^L_3 = g^{x_2x_3}, \ldots, K^R_n = K^L_1 = g^{x_nx_1}; & : (T, \mathsf{sk}) \\ Y_1 = K^R_1/K^L_1, Y_2 = K^R_2/K^L_2, \ldots, Y_n = K^R_n/K^L_n; & \\ T = (X_1, \ldots, X_n; Y_1, \ldots, Y_n); \mathsf{sk} = K^R_1K^R_2 \ldots K^R_n & \end{array} \right\}$$

$$\mathsf{Fake}' := \left\{ \begin{array}{ll} x_1, \ldots x_n \longleftarrow Z^*_q; & \\ X_1 = g^{x_1}, X_2 = g^{x_2}, \ldots, X_n = g^{x_n}; & \\ K^R_1 = K^L_2 = g^{x_1x_2}, K^R_2 = K^L_3 = g^{x_2x_3}, \ldots, K^R_{n-1} = K^L_n = g^{x_{n-1}x_n}; & : (T, \mathsf{sk}) \\ K^R_n = K^L_1 \longleftarrow G; & \\ Y_1 = K^R_1/K^L_1, Y_2 = K^R_2/K^L_2, \ldots, Y_n = K^R_n/K^L_n; & \\ T = (X_1, \ldots, X_n; Y_1, \ldots, Y_n); \mathsf{sk} = K^R_1K^R_2 \ldots K^R_n & \end{array} \right\}$$

Claim 1 : For any algorithm $\mathcal{A}$ running in time t, we have $|\mathsf{Prob}[(T,\mathsf{sk}) \longleftarrow \mathsf{Real} : \mathcal{A}(T,\mathsf{sk}) = 1] - \mathsf{Prob}[(T,\mathsf{sk}) \longleftarrow \mathsf{Fake}' : \mathcal{A}(T,\mathsf{sk}) = 1]| \leq \mathsf{Adv}_G^{\mathsf{DDH}}(t'') + \frac{1}{|G|}$.

Proof : We construct a distinguisher $\mathcal{D}$ for DDH problem using $\mathcal{A}$, which on an input $(A,B,C) \in G^3$, first generates a pair (T,sk) according to the distribution Dist' described below (which depends on A,B,C), then runs $\mathcal{A}$ on (T,sk) and outputs whatever $\mathcal{A}$ outputs.

$$\mathsf{Dist}' := \left\{ \begin{array}{ll} x_1, \ldots x_n \longleftarrow Z_q^*; & \\ X_1 = A^{x_1}, X_2 = g^{x_2}, \ldots, X_{n-1} = g^{x_{n-1}}, X_n = B^{x_n}; & \\ K_1^R = K_2^L = A^{x_1 x_2}, K_2^R = K_3^L = g^{x_2 x_3}, \ldots, K_{n-2}^R = K_{n-1}^L = g^{x_{n-2} x_{n-1}}; & : (T,\mathsf{sk}) \\ K_{n-1}^R = K_n^L = B^{x_{n-1} x_n}, K_n^R = K_1^L = C^{x_n x_1}; & \\ Y_1 = K_1^R / K_1^L, Y_2 = K_2^R / K_2^L, \ldots, Y_n = K_n^R / K_n^L; & \\ T = (X_1, \ldots, X_n; Y_1, \ldots, Y_n); \mathsf{sk} = K_1^R K_2^R \ldots K_n^R & \end{array} \right\}$$

The distribution Real and the distribution $\{a, b \longleftarrow Z_q^*, A = g^a, B = g^b, C = g^{ab}; (T,\mathsf{sk}) \longleftarrow \mathsf{Dist}' : (T,\mathsf{sk})\}$ are statistically equivalent as long as the exponents x_j used in Dist' are random. On the other hand, the distribution Fake' and the distribution $\{a, b \longleftarrow Z_q^*, c \longleftarrow Z_q^* \setminus \{ab\}, A = g^a, B = g^b, C = g^c; (T,\mathsf{sk}) \longleftarrow \mathsf{Dist}' : (T,\mathsf{sk})\}$ are statistically equivalent but for a factor of $\frac{1}{|G|}$. In distribution Fake', the value of $K_n^R (= K_1^L)$ is chosen uniformly at random from G whereas in Dist', this value is chosen uniformly from $G \setminus \{g^{ab}\}$. These two dristributions are statistically equivalent by the self reducibility property of DDH problem. Hence $|\mathsf{Prob}[(T,\mathsf{sk}) \longleftarrow \mathsf{Real} : \mathcal{A}(T,\mathsf{sk}) = 1] - \mathsf{Prob}[(T,\mathsf{sk}) \longleftarrow \mathsf{Fake}' : \mathcal{A}(T,\mathsf{sk}) = 1]| \leq |\mathsf{Prob}[a, b \longleftarrow Z_q^* : \mathcal{D}(g^a, g^b, g^{ab}) = 1] - \mathsf{Prob}[a, b \longleftarrow Z_q^*, c \longleftarrow Z_q^* \setminus \{ab\} : \mathcal{D}(g^a, g^b, g^c) = 1]| + \frac{1}{|G|} \leq \mathsf{Adv}_G^{\mathsf{DDH}}(t'') + \frac{1}{|G|}$ as the time of $\mathcal{D}$ is dominated by the time t'' of $\mathcal{A}$. ■(of Claim 1)

Next we define the final distribution Fake as follows and prove the Claim 2 stated below:

$$\mathsf{Fake} := \left\{ \begin{array}{ll} x_1, \ldots x_n \longleftarrow Z_q^*; & \\ X_1 = g^{x_1}, X_2 = g^{x_2}, \ldots, X_n = g^{x_n}; & \\ K_1^R = K_2^L, K_2^R = K_3^L, K_3^R = K_4^L, \ldots, K_n^R = K_1^L \longleftarrow G; & : (T,\mathsf{sk}) \\ Y_1 = K_1^R / K_1^L, Y_2 = K_2^R / K_2^L, \ldots, Y_n = K_n^R / K_n^L; & \\ T = (X_1, \ldots, X_n; Y_1, \ldots, Y_n); \mathsf{sk} = K_1^R K_2^R \ldots K_n^R & \end{array} \right\}$$

Claim 2: For any algorithm $\mathcal{A}$ running in time t, we have $|\mathsf{Prob}[(T,\mathsf{sk}) \longleftarrow \mathsf{Fake}' : \mathcal{A}(T,\mathsf{sk}) = 1] - \mathsf{Prob}[(T,\mathsf{sk}) \longleftarrow \mathsf{Fake} : \mathcal{A}(T,\mathsf{sk}) = 1]| \leq \mathsf{Adv}_G^{\mathsf{DDH}}(t'') + \frac{1}{|G|}$.

Proof : Given an adversary, we construct an algorithm $\mathcal{D}$ that takes $(A,B,C) \in G^3$ as input, generates a pair (T,sk) according to the distribution Dist described below (which depends on A,B,C), runs $\mathcal{A}$ on (T,sk) and outputs whatever $\mathcal{A}$ outputs.

$$\mathsf{Dist} := \left\{ \begin{array}{l} x_1, \dots x_n \longleftarrow Z_q^*; \\ \textbf{if } n \text{ is even } \textbf{then} \\ \quad \textbf{for } i = 1\ (2)\ n \textbf{ do} \\ \quad\quad X_i = A^{x_i}, X_{i+1} = B^{x_{i+1}}; \\ \quad \textbf{end for} \\ \textbf{end if} \\ \textbf{if } n \text{ is odd } \textbf{then} \\ \quad \textbf{for } i = 1\ (2)\ n-2 \textbf{ do} \\ \quad\quad X_i = A^{x_i}, X_{i+1} = B^{x_{i+1}}; \\ \quad \textbf{end for} \\ \quad X_n = A^{x_n} \\ \textbf{end if} \\ \textbf{for } i = 1\ (2)\ n-2 \textbf{ do} \\ \quad K_i^R = K_{i+1}^L = C^{x_i x_{i+1}}, K_{i+1}^R = K_{i+2}^L = C^{x_{i+1} x_{i+2}}; \\ \textbf{end do} \\ K_{n-1}^R = K_n^L = C^{x_{n-1} x_n}, K_n^R = K_1^L \longleftarrow G; \\ Y_1 = K_1^R / K_1^L, Y_2 = K_2^R / K_2^L, \dots, Y_n = K_n^R / K_n^L; \\ T = (X_1, \dots, X_n; Y_1, \dots, Y_n); \mathsf{sk} = K_1^R K_2^R \dots K_n^R \end{array} : (T, \mathsf{sk}) \right\}$$

The distribution Fake' and the distribution $\{a, b \longleftarrow Z_q^*, A = g^a, B = g^b, C = g^{ab}; (T, \mathsf{sk}) \longleftarrow \mathsf{Dist} : (T, \mathsf{sk})\}$ are statistically equivalent as long as the exponents x_j used in Dist are random. On the other hand, the distribution Fake and the distribution $\{a, b \longleftarrow Z_q^*, c \longleftarrow Z_q^* \setminus \{ab\}, A = g^a, B = g^b, C = g^c; (T, \mathsf{sk}) \longleftarrow \mathsf{Dist} : (T, \mathsf{sk})\}$ are statistically equivalent but a factor of $\frac{1}{|G|}$. In distribution Fake, the values of $K_i^R (= K_{i+1}^L)$ for $1 \leq i \leq n$ are chosen uniformly at random from G and in Dist, these value are chosen uniformly from $G \setminus \{g^{ab}\}$. Then by the self reducibility property of DDH problem, we have $|\mathsf{Prob}[(T, \mathsf{sk}) \longleftarrow \mathsf{Fake}' : \mathcal{A}(T, \mathsf{sk}) = 1] - \mathsf{Prob}[(T, \mathsf{sk}) \longleftarrow \mathsf{Fake} : \mathcal{A}(T, \mathsf{sk}) = 1]| \leq |\mathsf{Prob}[a, b \longleftarrow Z_q^* : \mathcal{D}(g^a, g^b, g^{ab}) = 1] - \mathsf{Prob}[a, b \longleftarrow Z_q^*, c \longleftarrow Z_q^* \setminus \{ab\} : \mathcal{D}(g^a, g^b, g^c) = 1]| + \frac{1}{|G|} \leq \mathsf{Adv}_G^{\mathsf{DDH}}(t'') + \frac{1}{|G|}$ as the time of $\mathcal{D}$ is dominated by that of $\mathcal{A}$ (which is t''). ■(of Claim 2)

Now we provide the proof of the following claim which deals with the Test query of $\mathcal{A}$.

Claim 3: For any computationally-unbounded adversary $\mathcal{A}$, we have $\mathsf{Prob}[(T, \mathsf{sk}_0) \longleftarrow \mathsf{Fake}; \mathsf{sk}_1 \longleftarrow G; b \longleftarrow \{0,1\} : \mathcal{A}(T, \mathsf{sk}_b) = b] = \frac{1}{2}$.

Proof : In Fake, let $v_i^R := \log_g K_i^R, 1 \leq i \leq n$. Then we have the following system of equations: $\log_g Y_1 = -v_n^R + v_1^R; \log_g Y_2 = -v_1^R + v_2^R; \dots; \log_g Y_n = -v_{n-1}^R + v_n^R$. Besides $\mathsf{sk} = g^{v_1^R + v_2^R + \dots + v_n^R}$ gives the equation $\log_g \mathsf{sk} = v_1^R + v_2^R + \dots + v_n^R$ which is linearly independent from the above system of equations. This implies that the session key sk is independent of the transcript T in Fake. Hence for any computationally unbounded adversary $\mathcal{A}$, $\mathsf{Prob}[(T, \mathsf{sk}_0) \longleftarrow \mathsf{Fake}; \mathsf{sk}_1 \longleftarrow G, b \longleftarrow \{0,1\} : \mathcal{A}(T, \mathsf{sk}_b) = b] = \frac{1}{2}$. ■(of Claim 3)

Now $\mathsf{Adv}_{\mathsf{UP},\mathcal{A}}^{\mathsf{KA}}(t, 1) := |2\mathsf{Prob}[\mathsf{Succ}] - 1| = 2|\mathsf{Prob}[(T, \mathsf{sk}_0) \longleftarrow \mathsf{Real}, \mathsf{sk}_1 \longleftarrow G, b \longleftarrow \{0,1\} : \mathcal{A}(T, \mathsf{sk}_b) = b] - \frac{1}{2}| = 2|\mathsf{Prob}[(T, \mathsf{sk}_0) \longleftarrow \mathsf{Real}, \mathsf{sk}_1 \longleftarrow G, b \longleftarrow \{0,1\} : \mathcal{A}(T, \mathsf{sk}_b) = b] - \mathsf{Prob}[(T, \mathsf{sk}_0) \longleftarrow \mathsf{Fake}, \mathsf{sk}_1 \longleftarrow G, b \longleftarrow \{0,1\} : \mathcal{A}(T, \mathsf{sk}_b) = b]|$ by Claim 3 and using Claim 1 and Claim 2, we obtain $\mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t, 1) \leq 4\mathsf{Adv}_G^{\mathsf{DDH}}(t'') + \frac{4}{|G|}$. Then by applying the self-reducibility property of DDH problem, we get the result stated in the Theorem.

Consider the case for $q_E (> 1)$ $\mathsf{Execute}$ query. The adversary first generates q_E tuples (A_i, B_i, C_i), $1 \le i \le q_E$ with the following properties from the tuple $(A, B, C) \in G^3$ given to the adversary.

1. If $(A, B, C) \longleftarrow \Delta_{\mathsf{Real}}$, then $(A_i, B_i, C_i) \longleftarrow \Delta_{\mathsf{Real}}$ for all i, $1 \le i \le q_E$ with (A_i, B_i) randomly distributed in G^2 (independently of anything else).

2. If $(A, B, C) \longleftarrow \Delta_{\mathsf{Rand}}$, then $(A_i, B_i, C_i) \longleftarrow \Delta_{\mathsf{Rand}}$ for all i, $1 \le i \le q_E$ (independently of anything else) with all but a probability $\frac{q_E}{|G|}$ it will be the case that $\log_g C_i \neq \log_g A_i \log_g B_i$ for all i.

Then proceeding in the similar way as above of defining distributions Real, Fake', Dist', Fake, Dist, we may define distributions Real_{q_E}, Fake'_{q_E}, Dist'_{q_E}, Fake_{q_E} and Dist_{q_E}which simply consist of q_E independent copies of each of the corresponding distributions. In case of Dist'_{q_E} and Dist_{q_E}, we use the corresponding tuple (A_i, B_i, C_i) for the i-th copy. We use notation $(\vec{T}, \vec{\mathsf{sk}})$ to denote the transcript/session key pair generated by these distributions. Then similar to the claims 1, 2 and 3, we can prove the following claims:

Claim 4 : For any algorithm $\mathcal{A}$ running in time t, we have $|\mathsf{Prob}[(\vec{T}, \vec{\mathsf{sk}}) \longleftarrow \mathsf{Real}_{q_E} : \mathcal{A}(\vec{T}, \vec{\mathsf{sk}}) = 1] - \mathsf{Prob}[(\vec{T}, \vec{\mathsf{sk}}) \longleftarrow \mathsf{Fake}'_{q_E} : \mathcal{A}(\vec{T}, \vec{\mathsf{sk}}) = 1]| \le \mathsf{Adv}_G^{\mathsf{DDH}}(t') + \frac{2q_E}{|G|}$. where t' is as in the statement of the Theorem.

Claim 5 : For any algorithm $\mathcal{A}$ running in time t, we have $|\mathsf{Prob}[(\vec{T}, \vec{\mathsf{sk}}) \longleftarrow \mathsf{Fake}'_{q_E} : \mathcal{A}(\vec{T}, \vec{\mathsf{sk}}) = 1] - \mathsf{Prob}[(\vec{T}, \vec{\mathsf{sk}}) \longleftarrow \mathsf{Fake}_{q_E} : \mathcal{A}(\vec{T}, \vec{\mathsf{sk}}) = 1]| \le \mathsf{Adv}_G^{\mathsf{DDH}}(t') + \frac{2q_E}{|G|}$ where t' is as in the statement of the Theorem.

Claim 6: For any computationally-unbounded adversary $\mathcal{A}$, we have $\mathsf{Prob}[(\vec{T}, \vec{\mathsf{sk}}_0) \longleftarrow \mathsf{Fake}; \vec{\mathsf{sk}}_1 \longleftarrow G^{q_E}; b \longleftarrow \{0,1\} : \mathcal{A}(\vec{T}, \vec{\mathsf{sk}}_b) = b] = \frac{1}{2}$.

These three claims yield the result stated in the theorem. ∎

Note : If n is even, then we need not to define the intermediate (T, sk) distribution Fake'. In this case, we can obtain a smaller upper bound of $\mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(1, q_E)$ considering only the distributions Real and Fake and defining Dist as in the proof of Claim 2. Consequently, we get a more tighter upper bound for $\mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t, q_E)$.

4.2 Security of the Authenticated (Static) Protocol

We prove that the security of our static authenticated protocol AP (subsection 3.2) relies on that of UP under the assumption that the underlying signature scheme DSig is secure. In fact, given any active adversary attacking AP, we can construct a passive adversary attacking UP of subsection 3.1. We state the security result of AP below in Theorem 4.2. Our proof technique is based on the proof technique used by Katz and Yung [25]. However, there are certain technical differences of our proof from that of [25].

1. The Katz-Yung technique is a generic technique for converting *any* unauthenticated protocol into an authenticated protocol. On the other hand, we concentrate on one particular protocol. Hence we can avoid some of the complexities of the Katz-Yung proof.

2. Katz-Yung protocol uses random nonces whereas our protocol does not.

3. In our unauthenticated protocol, there are no long term secret keys. Thus we can avoid the Corrupt oracle queries and can trivially achieve forward secrecy.

Theorem 4.2 *The authenticated protocol* AP *described in section 3.2 is secure against active adversary under DDH assumption, achieves forward secrecy and satisfies the following:*

$$\mathsf{Adv}^{\mathsf{AKA}}_{\mathsf{AP}}(t, q_E, q_S) \leq \mathsf{Adv}^{\mathsf{KA}}_{\mathsf{UP}}(t', q_E + \frac{q_S}{2}) + |\mathcal{P}|\ \mathsf{Adv}_{\mathsf{DSig}}(t')$$

where q_E and q_S are respectively the maximum number of Execute *and* Send *query an adversary may ask.*

Proof : Let $\mathcal{A}'$ be an adversary which attacks the authenticated protocol AP. Using this we construct an adversary $\mathcal{A}$ which attacks the unauthenticated protocol UP. We first have the following claim.

Claim : Let Forge be the event that a signature of DSig is forged by $\mathcal{A}'$. Then $\mathsf{Prob}[\mathsf{Forge}] \leq |\mathcal{P}|\ \mathsf{Adv}_{\mathsf{DSig}}(t')$.

Proof of Claim: Suppose the event Forge occurs. Then $\mathcal{A}'$ makes a query of the type $\mathsf{Send}(V, i, Y)$ where Y is either of the form $Y = U_k|1|X_k|\sigma_k$ with $\mathcal{V}(pk_{U_k}, U_k|1|X_k, \sigma_k) = 1$ or of the form $Y = U_k|2|X_k|d_k|\sigma_k$ with $\mathcal{V}(pk_{U_k}, U_k|2|X_k|d_k|, \sigma_k) = 1$ for some instance $\Pi^{d_k}_{U_k}$ with $X_k \in G$ and σ_k was not output by any instance of U_k on the respective messages. Using $\mathcal{A}'$, we construct an algorithm $\mathcal{F}$ that forges a signature for DSig as follows: Given a public key pk, algorithm $\mathcal{F}$ chooses a random $U \in \mathcal{P}$ and sets $pk_U = pk$. The other public keys and private keys for the system are generated honestly by $\mathcal{F}$. The forger $\mathcal{F}$ simulates all oracle queries of $\mathcal{A}'$ by executing protocol AP itself, obtaining the necessary signatures with respect to pk_U, as needed, from its signing oracle. Thus $\mathcal{F}$ provides a perfect simulation for $\mathcal{A}'$. If $\mathcal{A}'$ ever outputs a new valid message/signature pair with respect to $pk_U = pk$, then $\mathcal{F}$ outputs this pair as its forgery. The success probability of $\mathcal{F}$ is equal to $\frac{\mathsf{Prob}[\mathsf{Forge}]}{|\mathcal{P}|}$ and hence $\mathsf{Prob}[\mathsf{Forge}] \leq |\mathcal{P}|\ \mathsf{Adv}_{\mathsf{DSig}}(t')$. ■(of Claim)

Now we describe the construction of the passive adversary $\mathcal{A}$ attacking UP that uses adversary $\mathcal{A}'$ attacking AP. Adversary $\mathcal{A}$ uses a list tlist. It stores pairs of session IDs and transcripts in tlist.

Adversary $\mathcal{A}$ generates the verification/signing keys pk_U, sk_U for each user $U \in \mathcal{P}$ and gives the verification keys to $\mathcal{A}'$. If ever the event Forge occurs, adversary $\mathcal{A}$ aborts and outputs a random bit. Otherwise, $\mathcal{A}$ outputs whatever bit is eventually output by $\mathcal{A}'$. Note that since the signing and verification keys are generated by $\mathcal{A}$, it can detect occurrence of the event Forge.

$\mathcal{A}$ simulates the oracle queries of $\mathcal{A}'$ using its own queries to the Execute *oracle.* The idea is that the adversary $\mathcal{A}$ queried its Execute oracle to obtain a transcript T of UP for each Execute query of $\mathcal{A}'$ and also for each initial send query $\mathsf{Send}_0(U, i, *)$ of $\mathcal{A}'$. $\mathcal{A}$ then patches appropriate signatures with the messages in T to obtain a transcript T' of AP and uses T' to answer queries of $\mathcal{A}'$. Since by assumption, $\mathcal{A}'$ can not forge, $\mathcal{A}'$ is 'limitted' to send messages already contained in T'. This technique provides a good simulation. We discuss details below.

Execute queries: Suppose $\mathcal{A}'$ makes a query $\mathsf{Execute}((U_{i_1}, d_1), \ldots, (U_{i_k}, d_k))$. This means that instances $\Pi^{d_1}_{U_{i_1}}, \ldots, \Pi^{d_k}_{U_{i_k}}$ are involved in this session. $\mathcal{A}$ defines $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$ and sends the execute query to its Execute oracle. It receives as output a transcript T of an execution of UP. It appends (S, T) to tlist. Adversary $\mathcal{A}$ then expands the transcript T for the unauthenticated protocol into a transcript T' for the authenticated protocol according to the modification described in Section 3.2. It returns T' to $\mathcal{A}'$.

Send queries: The first send query that $\mathcal{A}'$ makes to an instance is to start a new session. We will denote such queries by Send_0 queries. To start a session between unused instances $\Pi^{d_1}_{U_{i_1}}, \ldots, \Pi^{d_k}_{U_{i_k}}$, the adversary has to make the send queries: $\mathsf{Send}_0(U_{i_j}, d_j, \langle U_{i_1}, \ldots, U_{i_k} \rangle \setminus U_{i_j})$ for $1 \leq j \leq k$. Note that these queries may be made in any order. When all these queries have been made, $\mathcal{A}$ sets $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$ and makes an Execute query to its own execute oracle. It receives a transcript T in return and stores (S, T) in the list tlist.

Assuming that signatures cannot be forged, any subsequent Send query (i.e., after a Send_0 query) to an instance Π_U^i is a properly structured message with a valid signature. For any such Send query, $\mathcal{A}$ verifies the query according to the algorithm of Section 3.2. If the verification fails, $\mathcal{A}$ sets $\mathsf{acc}_U^i = 0$ and $\mathsf{sk}_U^i = \mathsf{NULL}$ and aborts Π_U^i. Otherwise, $\mathcal{A}$ performs the action to be done by Π_U^i in the authenticated protocol. This is done in the following manner: $\mathcal{A}$ first finds the unique entry (S,T) in tlist such that $(U,i) \in S$. Such a unique entry exists for each instance by assumption. Now from T, $\mathcal{A}$ finds the appropriate message which corresponds to the message sent by $\mathcal{A}'$ to Π_U^i. From the transcript T, adversary $\mathcal{A}$ finds the next public information to be output by Π_U^i and returns it to $\mathcal{A}'$.

Reveal/Test queries : Suppose $\mathcal{A}'$ makes the query $\mathsf{Reveal}(U,i)$ or $\mathsf{Test}(U,i)$ to an instance Π_U^i for which $\mathsf{acc}_U^i = 1$. At this point the transcript T' in which Π_U^i participates has already been defined. Now $\mathcal{A}$ finds the unique pair (S,T) in tlist such that $(U,i) \in S$. Assuming that the event Forge does not occur, T is the unique unauthenticated transcript which corresponds to the transcript T'. Then $\mathcal{A}$ makes the appropriate Reveal or Test query to one of the instances involved in T and returns the result to $\mathcal{A}'$.

As long as Forge does not occur, the above simulation for $\mathcal{A}'$ is perfect. Whenever Forge occurs, adversary $\mathcal{A}$ aborts and outputs a random bit. So $\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}|\mathsf{Forge}] = \frac{1}{2}$. Now

$$
\begin{aligned}
\mathsf{Adv}_{\mathcal{A},\mathsf{UP}} &:= 2\ |\mathsf{Prob}_{\mathcal{A},\mathsf{UP}}[\mathsf{Succ}] - 1/2| \\
&= 2\ |\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \mathsf{Forge}] - 1/2| \\
&= 2\ |\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}|\mathsf{Forge}]\ \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 1/2| \\
&= 2\ |\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \overline{\mathsf{Forge}}] + (1/2)\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 1/2| \\
&= 2\ |\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}] - \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \mathsf{Forge}] + (1/2)\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 1/2| \\
&\geq |2\ \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}] - 1| - |\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Forge}] - 2\ \mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ} \wedge \mathsf{Forge}]| \\
&\geq \mathsf{Adv}_{\mathcal{A}',\mathsf{AP}} - \mathsf{Prob}[\mathsf{Forge}]
\end{aligned}
$$

The adversary $\mathcal{A}$ makes an $\mathsf{Execute}$ query for each $\mathsf{Execute}$ query of $\mathcal{A}'$. Also $\mathcal{A}$ makes an $\mathsf{Execute}$ query for each session started by $\mathcal{A}'$ using Send queries. Since a session involves at least two instances, such an $\mathsf{Execute}$ query is made after at least two Send queries of $\mathcal{A}'$. The total number of such $\mathsf{Execute}$ queries is at most $q_S/2$, where q_S is the number of Send queries made by $\mathcal{A}'$. The total number of $\mathsf{Execute}$ queries made by $\mathcal{A}$ is at most $q_E + q_S/2$, where q_E is the number of $\mathsf{Execute}$ queries made by $\mathcal{A}'$.
Also since $\mathsf{Adv}_{\mathcal{A},\mathsf{UP}} \leq \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t', q_E + q_S/2)$ by assumption, we obtain:

$$\mathsf{Adv}_{\mathsf{AP}}^{\mathsf{AKA}} \leq \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t', q_E + q_S/2) + \mathsf{Prob}[\mathsf{Forge}].$$

This yields the statement of the theorem. ∎

4.3 Security of the Dynamic Authenticated Protocol

In this subsection, we will show that the modifications described in Section 3.3 converts the protocol UP of Section 3.1 into a dynamic authenticated key agreement protocol DAP. Assuming that the signature scheme DSig is secure, we can convert any adversary attacking the protocol DAP into an adversary attacking the protocol UP. We ignore Corrupt queries since our protocol DAP does not use any long-term secret keys. Thus the protocol DAP trivially achieves forward secrecy. We state below our security result in Theorem 4.3.

Theorem 4.3 *The dynamic authenticated key agreement protocol* DAP *described in Section 3.3 satifies the following:*

$$\mathsf{Adv}_{\mathsf{DAP}}^{\mathsf{AKA}}(t, q_E, q_J, q_L, q_S) \leq \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t', q_E + (q_J + q_L + q_S)/2) + |\mathcal{P}|\ \mathsf{Adv}_{\mathsf{DSig}}(t')$$

where $t' \leq t + (|\mathcal{P}|q_E + q_J + q_L + q_S)t_{\mathsf{DAP}}$*, where* t_{DAP} *is the time required for execution of* DAP *by any one of the users.*

Proof : Let $\mathcal{A}'$ be an adversary which attacks the dynamic authenticated protocol DAP. Using this we construct an adversary $\mathcal{A}$ which attacks the unauthenticated protocol UP. As in the previous proof, we have the following claim.

Claim : Let Forge be the event that a signature is forged by $\mathcal{A}'$. Then $\mathsf{Prob}[\mathsf{Forge}] \leq |\mathcal{P}|\ \mathsf{Adv}_{\mathsf{DSig}}(t')$.

Now we describe the construction of the passive adversary $\mathcal{A}$ attacking UP that uses adversary $\mathcal{A}'$ attacking DAP. Adversary $\mathcal{A}$ can execute the unauthenticated protocol UP several times among any subset of $\mathcal{P}$ and also can obtain the session key of the protocol execution by making Reveal queries to any instances involved in the session. *We will show that* $\mathcal{A}$ *itself simulates the* Join *and* Leave *queries of* $\mathcal{A}'$ *using its own* Execute *and* Reveal *oracle.* Adversary $\mathcal{A}$ maintains a list Tlist to store pairs of session IDs and transcripts. It also uses two lists Jlist and Llist to be specified later.

Adversary $\mathcal{A}$ generates the verification/signing keys pk_U, sk_U for each user $U \in \mathcal{P}$ and gives the verification keys to $\mathcal{A}'$. If ever the event Forge occurs, adversary $\mathcal{A}$ aborts and outputs a random bit. Otherwise, $\mathcal{A}$ outputs whatever bit is eventually output by $\mathcal{A}'$. Note that since the signing and verification keys are generated by $\mathcal{A}$, it can detect occurrence of the event Forge. $\mathcal{A}$ simulates the oracle queries of $\mathcal{A}'$ using its own queries to the Execute and Reveal oracles. We provide details below.

Execute queries: These queries are simulated as in the proof of Theorem 4.2.

Send queries: Apart from the usual send queries, there are two special type of send queries, Send_J and Send_L.

If the set $S_1 = \{(U_{i_{k+1}}, d_{k+1}), \ldots, (U_{i_{k+l}}, d_{k+l})\}$ of unused instances wants to join the group $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$, then $\mathcal{A}'$ will make $\mathsf{Send}_J(U_{i_j}, d_j, \langle U_{i_1}, \ldots, U_{i_k}\rangle)$ query for all j, $k+1 \leq j \leq k+l$. These queries initiate $\mathsf{Join}(S, S_1)$ query . Note that the instances in S might have already executed either the unauthenticated (a) key agreement protocol or (b) join protocol or (c) leave protocol. Accordingly, $\mathcal{A}$ first finds any one of the following form of a unique entry: (1) (S, T) in Tlist or (2) (S', S'', T) in Jlist with $S = S' \cup S''$ or (3) (S', S'', T) in Llist with $S = S' \setminus S''$. If no such entry, $\mathcal{A}$ makes an execute query to its own execute oracle on S, gets a transcript T and stores (S, T) in Tlist.

In case $(S, T) \in$ Tlist, $\mathcal{A}$ first makes a Reveal query to any instance in S to obtain the session key sk corresponding to T, computes the seed $x = \mathcal{H}(\mathsf{sk})$ and simulates the algorithm for Join by querying its Execute oracle (making appropriate changes). Then patching up signature in each message, $\mathcal{A}$ obtains a transcript T' and stores (S, S_1, T') in Jlist. $\mathcal{A}$ thus simulates the transcript T' of Join using its own Execute and Reveal oracles. In the remaining cases (2) and (3), T is generated by $\mathcal{A}$ itself and so $\mathcal{A}$ can simulate transcript T' of Join from T.

Similarly, when a set $S_2 = \{(U_{l_1}, d_{l_1}), \ldots, (U_{l_m}, d_{l_m})\}$ of unused instances wants to leave the group $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$, then $\mathcal{A}'$ will make $\mathsf{Send}_L(U_{i_j}, d_j, \langle U_{i_1}, \ldots, U_{i_k}\rangle)$ query for all $j, j \in \{l_1, \ldots, l_m\}$. These queries initiate $\mathsf{Leave}(S, S_2)$ query. As mentioned above in case of member join, $\mathcal{A}$ first finds a unique entry of the form (S, T) in Tlist or a unique entry of the form (S', S'', T) in Jlist with $S = S' \cup S''$ or a unique entry of the form (S', S'', T) in Llist with $S = S' \setminus S''$. If no such entry, then $\mathcal{A}$ makes a query to its own execute oracle on S, gets a transcript T and stores (S, T) in Tlist.

$\mathcal{A}$ then simulates the algorithm for Leave by itself and gets a modified transcript T' from T as follows: A first detects the positions in T where the new messages are to be injected or the old messages are to be replaced by new messages. $\mathcal{A}$ do these modifications in T according to the algorithm Leave described in Section 3.3.1 and gets a modified transcript T' by patching up appropriate signature with each message. Thus $\mathcal{A}$ expands T into a transcript T' for Leave algorithm. $\mathcal{A}$ stores (S, S_2, T') in Llist.

Send_0 queries are answered as in Theorem 4.2. The usual send queries are simulated as in Theorem 4.2 with the following modifications.

Suppose $\mathcal{A}'$ makes a Send query to instance Π_U^i. After proper verification, $\mathcal{A}$ finds a unique entry (S, T) in Tlist such that $(U, i) \in S$. The answer to this query is as in Theorem 4.2. If no such entry is found, then $\mathcal{A}$ finds a unique entry (S, S_1, T') in Jlist such that $(U, i) \in S_1$. This means that the session for Join has already been initiated. $\mathcal{A}$ then obtains the next public information for T' to be output by Π_U^i (provided all necessary information has been received by Π_U^i by send queries from $\mathcal{A}'$) and sends it to $\mathcal{A}'$. If $\mathcal{A}$ finds a unique entry (S, S_2, T') in Llist such that $(U, i) \in S_2$, then as above, the appropriate answer to the query is found from T'.

Join queries : Suppose $\mathcal{A}'$ makes a query $\mathsf{Join}(S, S_1)$ where $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$ and $S_1 = \{(U_{i_{k+1}}, d_{k+1}), \ldots, (U_{i_{k+l}}, d_{k+l})$. The instances $\Pi_{U_{i_{k+1}}}^{d_{k+1}}, \ldots, \Pi_{U_{i_{k+l}}}^{d_{k+l}}$ want to join in the group $\Pi_{U_{i_1}}^{d_1}, \ldots \Pi_{U_{i_k}}^{d_k}$. $\mathcal{A}$ finds an entry of the form (S, S_1, T') in Jlist. If no such entry, then the adversary $\mathcal{A}'$ is given no output. Otherwise, $\mathcal{A}$ returns T' to $\mathcal{A}'$

Leave queries : Suppose $\mathcal{A}'$ makes a query $\mathsf{Leave}(S, S_2)$ where $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, , d_k)\}$ and $S_2 = \{(U_{l_1}, d_{l_1}), \ldots, (U_{l_m}, d_{l_m})\}$ where $(U_{l_j}, d_{l_j}) \in S$ for $1 \leq j \leq m$. The instance $\Pi_{U_{l_1}}^{d_{l_1}}, \ldots \Pi_{U_{l_m}}^{d_{l_m}}$ want to leave the group $\Pi_{U_{i_1}}^{d_1}, \ldots, \Pi_{U_{i_k}}^{d_k}$ where $U_{l_j} \in \{U_{i_1}, \ldots, U_{i_k}\}$ for $1 \leq j \leq m$. $\mathcal{A}$ finds an entry of the form (S, S_2, T') in Llist. If no such entry, then the adversary $\mathcal{A}'$ is given no output. Otherwise, $\mathcal{A}$ returns T' to $\mathcal{A}'$.

Reveal/Test queries : Suppose $\mathcal{A}'$ makes the query $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$ for an instance Π_U^i for which $\mathsf{acc}_U^i = 1$. At this point the transcript T' in which Π_U^i participates has already been defined. If T' corresponds to the transcript of the authenticated protocol, then $\mathcal{A}$ finds the unique pair (S, T) in Tlist such that $(U, i) \in S$. Assuming that the event Forge does not occur, T is the unique unauthenticated transcript which corresponds to the transcript T'. Then $\mathcal{A}$ makes the appropriate Reveal or Test query to one of the instances involved in T and returns the result to $\mathcal{A}'$. Otherwise, T' is the transcript for Join or Leave, as the case may be. Since T' has been simulated by $\mathcal{A}$, $\mathcal{A}$ is able to compute the modified session key and hence send an appropriate reply to $\mathcal{A}'$.

As long as Forge does not occur, the above simulation for $\mathcal{A}'$ is perfect. Whenever Forge occurs, adversary $\mathcal{A}$ aborts and outputs a random bit. So $\mathsf{Prob}_{\mathcal{A}',\mathsf{AP}}[\mathsf{Succ}|\mathsf{Forge}] = \frac{1}{2}$. Using this, one can show

$$\mathsf{Adv}_{\mathcal{A},\mathsf{UP}} \quad \geq \quad \mathsf{Adv}_{\mathcal{A}',\mathsf{DAP}} - \mathsf{Prob}[\mathsf{Forge}]$$

The adversary $\mathcal{A}$ makes an $\mathsf{Execute}$ query for each $\mathsf{Execute}$ query of $\mathcal{A}'$. $\mathcal{A}'$ makes q_J Join queries and q_L Leave queries. These queries are initialized respectively by Send_J and Send_L queries of $\mathcal{A}'$. Now each of Send_J and Send_L query of $\mathcal{A}'$ makes at most one $\mathsf{Execute}$ query of $\mathcal{A}$. Thus there are at most $q_J + q_L$ execute query made by $\mathcal{A}$ to respond all the Send_J and Send_L queries of $\mathcal{A}'$.

Also $\mathcal{A}$ makes an $\mathsf{Execute}$ query for each session started by $\mathcal{A}'$ using Send queries. Since a session involves at least two instances, such an $\mathsf{Execute}$ query is made after at least two Send queries of $\mathcal{A}'$. Thus there are $(q_S - q_J - q_L)/2$ execute queries of $\mathcal{A}$ to respond all other Send queries of $\mathcal{A}'$, where q_S is the number of Send queries made by $\mathcal{A}'$. Hence the total number of $\mathsf{Execute}$ queries made by $\mathcal{A}$ is at most $q_E + q_J + q_L + (q_S - q_J - q_L)/2 = q_E + (q_J + q_L + q_S)/2$, where q_E is the number of $\mathsf{Execute}$ queries made by $\mathcal{A}'$. Also since $\mathsf{Adv}_{\mathcal{A},\mathsf{UP}}(t, q_E, q_J, q_L, q_S) \leq \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t', q_E + q_J/2 + q_L/2 + q_S/2)$ by assumption, we obtain:

$$\mathsf{Adv}_{\mathsf{DAP}}^{\mathsf{AKA}} \leq \mathsf{Adv}_{\mathsf{UP}}^{\mathsf{KA}}(t', q_E + (q_J + q_L + q_S)/2) + \mathsf{Prob}[\mathsf{Forge}].$$

This yields the statement of the theorem. ■

5 Conclusion

We present and analyze a simple and elegant constant round group key agreement protocol and enhance it to dynamic setting where a set of users can leave or join the group at any time during protocol execution with updated keys. The emphasis of this work is to achieve provable security of our scheme under DDH assumption. We provide a concrete security analysis of our protocol against active adversary in the standard security model of Bresson *et al.* [15] adapting Katz-Yung [25] technique. The protocol is forward secure, efficient and fully symmetric.

References

[1] M. Abdalla, M. Bellare and P. Rogaway. *DHIES : An encryption scheme based on the Diffie-Hellman problem*, CT-RSA 2001 : 143-158.

[2] G. Ateniese, M. Steiner, and G. Tsudik. *Authenticated Group Key Agreement and Friends.* In ACM CCS98[1], pages 17-26.

[3] G. Ateniese, M. Steiner, and G. Tsudik. *New Multiparty Authenticated Services and Key Agreement Protocols*, Journal of Selected Areas in Communications, 18(4):1-13, IEEE, 2000.

[4] P. S. L. M. Barreto, H. Y. Kim and M. Scott. *Efficient algorithms for pairing-based cryptosystems.* Advances in Cryptology - Crypto '2002, LNCS 2442, Springer-Verlag (2002), pp. 354-368.

[5] R.Barua, R.Dutta, P.Sarkar. *Extending Joux Protocol to MultiParty Key Agreement.* Indocrypt2003. Also available at http://eprint.iacr.org/2003/062.

[6] K. Becker and U. Wille. *Communication Complexity of Group Key Distribution.* ACMCCS '98.

[7] M. Bellare, R. Canetti, and H. Krawczyk. *A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols.* In Proceedings of the 30th Annual Symposium on the Theory of Computing, pages 419-428. ACM, 1998. http://www.cs.edu/users/mihir/papers/key-distribution.html/.

[8] M. Bellare and P. Rogaway. *Entity Authentication and Key Distribution.* Advances in Cryptology - CRYPTO '93, LNCS Vol. 773, D. Stinson ed., Springer-Verlag, 1994, pp. 231-249.

[9] D. Boneh and M. Franklin. *Identity-Based Encryption from the Weil Pairing.* In Advances in Cryptology - CRYPTO '01, LNCS 2139, pages 213-229, Springer-Verlag, 2001.

Verifiably Encrypted

[10] D. Boneh, B. Lynn, and H. Shacham. *Short Signature from Weil Pairing*, Proc. of Asiacrypt 2001, LNCS, Springer, pp. 213-229, 2001.

[11] C. Boyd and J. M. G. Nieto. *Round-Optimal Contributory Conference Key Agreement*, Public Key Cryptography, LNCS vol. 2567, Y. Desmedt ed., Springer-Verlag, 2003, pp. 161-174.

[12] A. Boldyreva. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. Public Key Cryptography 2003: 31-46.

[13] E. Bresson and D. Catalano. *Constant Round Authenticated Group Key Agreement via Distributed Computing.* In Proceedings of PKC'04, LNCS 2947, pp. 115-129, 2004.

[14] E. Bresson, O. Chevassut, and D. Pointcheval. *Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case.* Advances in Cryptology - Asiacrypt 2001, LNCS vol. 2248, C. Boyd ed., Springer-Verlag, 2001, pp. 290-309.

[15] E. Bresson, O. Chevassut, and D. Pointcheval. *Dynamic Group Diffie-Hellman Key Exchange under Standard Assumptions.* Advances in Cryptology - Eurocrypt '02, LNCS 2332, L. Knudsen ed., Springer-Verlag, 2002, pp. 321-336.

[16] E. Bresson, O. Chevassut, A. Essiari and D. Pointcheval. *Mutual Authentication and Group Key Agreement for low-power Mobile Devices.* Computer Communication, vol. 27(17), 2004, pp. 1730-1737. A preliminary version appeared in Proceedings of the 5th IFIP-TC6/IEEE , MWCN'03.

[17] E. Bresson, O. Chevassut, D. Pointcheval, and J. J. Quisquater. *Provably Authenticated Group Diffie-Hellman Key Exchange.* Proc. 8th Annual ACM Conference on Computer and Communications Security, ACM, 2001, pp. 255-264.

[18] M. Burmester and Y. Desmedt. *A Secure and Efficient Conference Key Distribution System.* In A. De Santis, editor, Advances in Cryptology EUROCRYPT '94, Workshop on the theory and Application of Cryptographic Techniques, LNCS 950, pages 275-286, Springer-Verlag, 1995.

[19] R. Dutta, R. Barua and P. Sarkar. *Pairing Based Cryptographic Protocols : A Survey.* Cryptology ePrint Archive, Report 2004/064, available at http://eprint.iacr.org/2004/064.

[20] R. Dutta, R. Barua and P. Sarkar. *Provably Secure Authenticated Tree Based Group Key Agreement.* Proceedings of ICICS'04, LNCS, Springer-Verlag, 2004. Also available at Cryptology ePrint Archive, Report 2004/090.

[21] W. Diffie and M. Hellman. *New Directions In Cryptography*, IEEE Transactions on Information Theory, IT-22(6) : 644-654, November 1976.

[22] S. Galbraith, K. Harrison and D. Soldera. *Implementing the Tate Pairing*, Algorithm Number Theory Symposium - ANTS V, LNCS 2369, Springer- Verlag (2002), pp. 324-337.

[23] I. Ingemarsson, D. T. Tang, and C. K. Wong. *A Conference Key Distribution System*, IEEE Transactions on Information Theory 28(5) : 714-720 (1982).

[24] A. Joux. *A One Round Protocol for Tripartite Diffie-Hellman*, ANTS IV, LNCS 1838, pp. 385-394, Springer-Verlag, 2000.

[25] J. Katz and M. Yung. *Scalable Protocols for Authenticated Group Key Exchange*, In Advances in Cryptology - CRYPTO 2003.

[26] Y. Kim, A. Perrig, and G. Tsudik. *Simple and Fault-tolerant Key Agreement for Dynamic Collaborative Groups.* In S. Jajodia, editor, 7th ACM Conference on Computation and Communication Security, pages 235-244, Athens, Greece, Nov. 2000, ACM press.

[27] Y. Kim, A. Perrig, and G. Tsudik. *Tree based Group Key Agreement.* Report 2002/009, http://eprint.iacr.org, 2002.

[28] H. J. Kim, S. M. Lee and D. H. Lee. *Constant-Round Authenticated Group Key Exchange for Dynamic Groups*. In the Proceedings of Asiacrypt'04, to appear.

[29] J. Nam, J. Lee, S. Kim and D. Won. *DDH-Based Group Key Agreement For Mobile Computing.* Available at http://eprint.iacr.org, Report 2004/127.

[30] J. Nam, S. Kim, H. Yang and D. Won. *Secure Group Communications over Combined Wired/Wireless Network*. Available at http:eprint.iacr.org, Report 2004/260.

[31] M. Steiner, G. Tsudik, M. Waidner. *Diffie-Hellman Key Distribution Extended to Group Communication*, ACM Conference on Computation and Communication Security, 1996.

$\mathsf{Prob}[\mathsf{Forge}] \leq |\mathcal{P}|\ \mathsf{Adv}_{\mathsf{DSig}}(t')$.