Dynamic Group Key Agreement in Tree-Based Setting

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 provably secure tree based authenticated group key agreement protocol in dynamic scenario. Bilinear pairing and multi-signature are at the heart of our protocol. We prove that our protocol is provably secure in the standard security model of Bresson *et al.* An appropriate modification of Katz-Yung approach to tree based setting is adopted while proving its security against active adversaries. The protocol has an in-built hierarchical structure that makes it desirable for certain applications.
Keywords: bilinear pairing, multi-signature, group key agreement, dynamic operations, provable security

1 Introduction

A group key agreement protocol allows a group of users to exchange information over public network to agree upon a common secret key from which a session key can be derived. This common session key can later be used to achieve desirable security goals, such as authentication, confidentiality and data integrity.

Tree based group key agreement protocols are typically essential while the users are grouped into a hierarchical structure. The leaves of the tree denote individual users and each internal node corresponds to a user that represents the set of users in the subtree rooted at that node. The representative users have more computational resources than other users in the subtree. In a tree based group key agreement protocol, the set of all users in each subtree agree upon a common secret key. Besides, making optimal use of precomputed values in the previous session, a group of users can save computation and communication in subsequent sessions in which users join or leave the group. Moreover, some subclass of users agree upon multiple common keys in a single session which facilitates a typical subclass of users of the group to securely communicate among themselves. These features make tree based key agreement protocols desirable for certain applications.

In this work, we present a provably secure tree based authenticated group key agreement in the dynamic scenario where a user can join or leave the group as his desire with updating sets of keys. Although designing constant round provably secure group key agreement schemes [13, 25, 26] get much attention in the current research, it does not eliminate the need of tree based key agreement. We can combine constant round protocols and tree based protocols to get hybrid group key agreement which are efficient in terms of both computation and communication. Consider the situation where there are collection of user sets, each having a common key agreed upon by executing an efficient constant round protocol among the users in that user set. Now executing the constant round protocol among these user sets may not always be desirable and executing the protocol among all the users may be expensive from computation or

communication point of view when number of users in each subgroup is large. For instance, if the number of groups is about 20 and each group is large, then a constant round key agreement using the protocol of [25] will involve a large number of computations as well as communications. In contrast, the tree based protocol (with the representatives of each group) will compute the common key in 3 rounds with lesser number of communications and verifications. Thus, a tree based scheme can be incorporated among these user sets to get an efficient multi-party key agreement protocol. There are quite a number of tree based key agreement protocols [6, 24]. A ternary tree based protocol was proposed by Barua *et al.* [5] that extends the basic Joux [21] protocol to multi-party setting. They have shown that the protocol is secure against passive adversaries. Dutta *et al.* [17] authenticate this unauthenticated protocol using multi-signature and provide a concrete security analysis against active adversaries in the standard model as formalized by Bresson *et al.* [14]. This security was achieved by modifying the Katz and Yung [22] technique to tree based setting. The present work further extends this static authenticated protocol [17] to dynamic authenticated protocol and provides a proof of security in the security model of Bresson *et al.* [14]. Our protocol is designed to ensure minimum modification to the computation already precomputed when a user leaves or joins the group. Besides, if the tree structure is not maintained after a join or leave operation, then the subsequent join or leave in the group can not be performed anymore. Thus retaining the tree structure is another important issue of our protocol while such dynamic operations (join or leave) are concerned.

2 Preliminaries

In this section, we put forward the required notations, definitions, models that we use in the discussions of the subsequent sections. We use the notation $a \in_R S$ to denote that a is chosen randomly from the set S.

2.1 Cryptographic Bilinear Maps

Let G_1, G_2 be two groups of the same prime order q. We view G_1 as an additive group and G_2 as a multiplicative group. A mapping $e : G_1 \times G_1 \to G_2$ satisfying the following properties is called a cryptographic bilinear map:

Bilinearity : $e(aP, bQ) = e(P, Q)^{ab}$ for all $P, Q \in G_1$ and $a, b \in Z_q^*$.
Non-degeneracy : If P is a generator of G_1, then $e(P, P)$ is a generator of G_2.
Computablity : There exists an efficient algorithm to compute $e(P, Q)$.

Modified Weil Pairing [9] and Tate Pairing [4, 19] are examples of cryptographic bilinear maps.

2.2 Security Model

We assume that the reader is familier with the model of Bresson *et al.* [14], which is the model in which we prove security of our dynamic key aggreement protocol. For completeness, we review their definitions and refer the reader to [14] for more details.

Let $\mathcal{P} = \{U_1, \ldots, U_n\}$ be a set of n (fixed) users or participants. A user can execute the protocol for group key agreement several times with different partners, can join or leave the group at it's desire by executing the protocols for Insert or Delete. We assume that users do not deviate from the protocol and 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. These queries are as follows, where Π_U^i denotes the i-th instance of user U and sk_U^i denotes the session key after execution of the protocol by Π_U^i.

– $\mathsf{Send}(U, i, m)$: 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_U^i upon receipt of message m. The adversary is allowed to prompt the unused instance Π_U^i to initiate the protocol with partners $U_2, \ldots, U_l, l \leq n$, by invoking $\mathsf{Send}(U, i, \langle U_2, \ldots, U_l \rangle)$.

– $\mathsf{Execute}(\{(V_1, i_1), \ldots, (V_l, i_l)\})$: Here $\{V_1, \ldots, V_l\}$ is a non empty subset of $\mathcal{P}$. This query models passive attacks in which the attacker evesdrops 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.

– $\mathsf{Join}(\{(V_1, i_1), \ldots, (V_l, i_l)\}, (U, i))$: This query models the insertion of a user instance Π_U^i in the group $(V_1, \ldots, V_l) \in \mathcal{P}$ for which $\mathsf{Execute}$ have already been queried. The output of this query is the transcript generated by the invocation of algorithm Insert. If $\mathsf{Execute}(\{(V_1, i_1), \ldots (V_l, i_l)\})$ has not taken place, then the adversary is given no output.

– $\mathsf{Leave}(\{(V_1, i_1), \ldots, (V_l, i_l)\}, (U, i))$: This query models the removal of a user instance Π_U^i from the group $(V_1, \ldots V_l) \in \mathcal{P}$. If $\mathsf{Execute}(\{(V_1, i_1), \ldots (V_l, i_l)\})$ has not taken place, then the adversary is given no output. Otherwise, algorithm Delete is invoked. The adversary is given the transcript generated by the honest execution of procedure Delete.

– $\mathsf{Reveal}(U, i)$: 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 $\mathsf{Execute}$, Join, Leave, Reveal, $\mathsf{Corrupt}$ and Test oracles, is considered to be passive while an active adversary is given access to the Send oracle in addition. We also use notations

sid_U^i : session identity for instance Π_U^i. We set $\mathsf{sid}_U^i = S = \{(U_1, i_1), \ldots, (U_k, i_k)\}$ such that $(U, i) \in S$ and $\Pi_{U_1}^{i_1}, \ldots, \Pi_{U_k}^{i_k}$ wish to agree upon a common key.

pid_U^i : partner identity for instance Π_U^i, defined by $\mathsf{pid}_U^i = \{U_1, \ldots, U_k\}$, such that $(U_j, i_j) \in \mathsf{sid}_U^i$ for all $1 \leq j \leq k$.

acc_U^i : 0/1-valued variable which is set to be 1 by Π_U^i upon normal termination of the session and 0 otherwise.

The adversary can ask Send, $\mathsf{Execute}$, Join, Leave, Reveal and $\mathsf{Corrupt}$ 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. 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 the advantage functions.

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

2.3 Decision Hash Bilinear Diffie-Hellman (DHBDH) Problem

Let (G_1, G_2, e) be as in Section 2.1. We define the following problem. Given an instance (P, aP, bP, cP, r) for some $a, b, c, r \in_R Z_q^*$ and a one way hash function $H : G_2 \to Z_q^*$, to decide whether $r = H(e(P,P)^{abc}) \bmod q$. This problem is termed DHBDH problem as defined in [5] and is a combination of the bilinear Diffie-Hellman(BDH) problem and a variation of the hash Diffie-Hellman(HDH) problem. The DHBDH assumption is that there exists no probabilistic, polynomial time, 0/1-valued algorithm which can solve the DHBDH problem with non-negligible probability of success.

2.4 Multi-signatures

Multi-signatures allow a group of users to sign a message, such that a verifier can verify that all users indeed signed the message. We use the multi-signatures presented by Boldyreva in [12] which is based on the Boneh-Lynn-Shacham [10] (BLS) pairing based short signature. Formally, a multi-signature scheme consists of three algorithms $\mathsf{MSig} = (\mathcal{MK}, \mathcal{MS}, \mathcal{MV})$, where $\mathcal{MK}$ is the key generation algorithm; $\mathcal{MS}$ is the signature generation algorithm and $\mathcal{MV}$ is the signature verification algorithm. We denote by $\mathsf{Succ}_{\mathsf{DSig}}(t)$ the maximum success probability of any adversary running in time t to forge signatures for a standard digital signature scheme $\mathsf{DSig} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$. Similarly, by $\mathsf{Succ}_{\mathsf{MSig}}(t)$ the maximum success probability of any adversary running in time t to break the multi-signature scheme MSig based on DSig.

3 A Provably Secure Dynamic Group Key Agreement Protocol

Our protocol extends the authenticated tree-based multi-party group key agreement protocol of [17] to dynamic case where a user can leave or join the group. The protocol is designed to ensure minimum modification to the computations already precomputed when a user leaves or joins the group. By appropriately modifying the proof in [17], the security of our protocol can be reduced to that of the unauthenticated protocol of [5] which is a provably secure scheme under DHBDH assumption.

Suppose a set of n users $\mathcal{P} = \{U_1, U_2, \ldots, U_n\}$ wish to agree upon a secret key. Let US be a subset of users. Quite often, we identify a user with its instance during the execution of a protocol. In case US is a singleton set, we will identify US with the instance it contains. Each user set US has a representative Rep(US) and for the sake of concreteness we take $\mathsf{Rep}(\mathsf{US}) = U_j$ where $j = \min\{k : \Pi^{d_k}_{U_k} \in \mathsf{US}\}$. We use the

notation $A[1,\ldots,n]$ for an array of n elements $A_1,\ldots,A_n$ and write $A[i]$ or A_i to denote the ith element of array $A[\,]$. Let $G_1 = \langle P\rangle, G_2$ (groups of prime order q) and $e(,)$ be as described in Section 2.1. We choose a hash function $H : G_2 \to Z_q^*$. The public parameters are $\mathsf{params} = (G_1, G_2, e, q, P, H)$. Each user $U_i \in \mathcal{P}$ chooses $s_i \in Z_q^*$ at random which it uses as its ephemeral key. These keys are session specific and determine the final common key for the users in a session.

3.1 Unauthenticated Key Agreement Protocol of [5]

This section presents an informal description of the unauthennticated protocol of [5]. Security of our dynamic key agreement protocol relies on the security of this scheme.

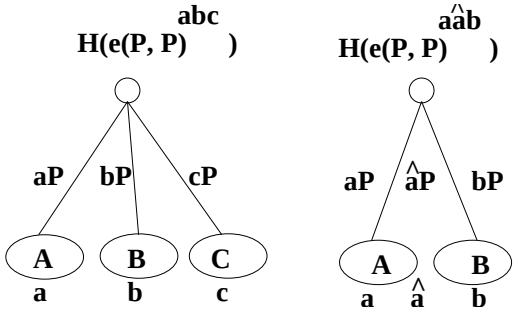

Figure 1: **procedure** CombineThree and **Procedure** CombineTwo

Let $p = \lfloor \frac{n}{3} \rfloor$ and $r = n \bmod 3$. The set of users participating in a session is partitioned into three user sets $\mathsf{US}_1, \mathsf{US}_2, \mathsf{US}_3$ with respective cardinalities being p, p, p if $r = 0$; $p, p, p+1$ if $r = 1$; and $p, p+1, p+1$ if $r = 2$. This top down recursive procedure KeyAgreement is invoked for further partitioning to obtain a ternary tree structure (*cf.* Figure 2). The lowest level 0 consists of singleton users having a secret key. We invoke CombineTwo, a key agreement protocol for two user sets and CombineThree, a key agreement protocol for three user sets in the key tree thus obtained. We demonstrate these two procedure in Figure 1. For more details, we refer [5].

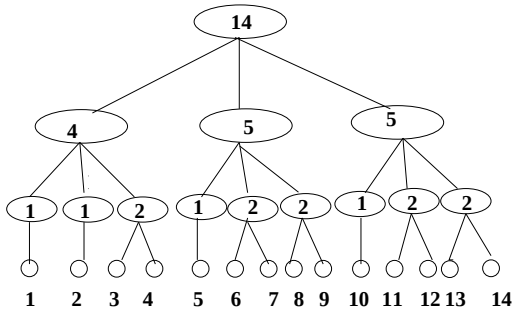

Figure 2: **procedure** KeyAgreement for $n = 14$

All communications are done by representatives and users in each user set have a common agreed key. In CombineThree, a, b, c respectively are the common agreed key of user sets A, B, C. Representative of user set A sends aP to both the user sets B, C. Similarly, representative of B sends bP to both A, C and representative of C sends cP to both A, B. After these communications, each user can compute the common agreed key $H(e(P,P)^{abc})$. In CombineTwo, users in user set A has common agreed key a, users in user set B has common agreed key b. Representative of A sends aP to user set B and representative of

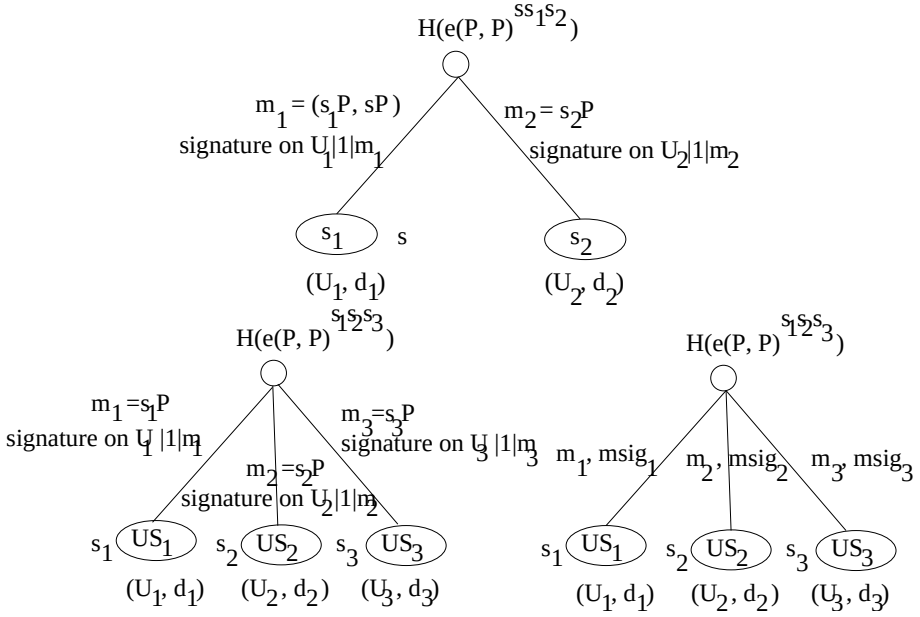

Figure 3: **procedure** AuthCombTwo, AuthCombThree-A and AuthCombThree-B

B sends bP to user set A. Besides representative of user set A generates a random key $\hat{a} \in Z_q^*$ and sends $\hat{a}P$ to all the users in both A, B. After these communications, each user can compute the common agreed key $H(e(P,P)^{a\hat{a}b})$.

3.2 Authenticated Key Agreement Protocol of [17]

This protocol incorporates secure signature based authentication mechanism into the unauthenticated protocol of Barua, Dutta, Sarkar [17]. Each user U_i chooses a signing and a verification key sk_i (or sk_{U_i}) and pk_i (or pk_{U_i}) respectively as part of the basic signature scheme DSig and multi-signature scheme MSig.

Besides, this authentication mechanism uses a variable, partial session-identity $\mathsf{psid}_{U_{i_j}}^{d_j}$ for each instance $\Pi_{U_{i_j}}^{d_j}$ which is initially set to be $\{(U_{i_j}, d_j)\}$ and after completion of the session, $\mathsf{psid}_{U_{i_j}}^{d_j} = \mathsf{sid}_{U_{i_j}}^{d_j} = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$ where instances $\Pi_{U_{i_1}}^{d_1} \ldots, \Pi_{U_{i_k}}^{d_k}$ are involved in this session. The session-identity $\mathsf{sid}_{U_{i_j}}^{d_j}$ uniquely identifies the session and is same for all instances participating in this session. The authenticated protocol consists of an algorithm for two-party authenticated key agreement AuthCombTwo, a three-party authenticated key agreement AuthCombThree-A and an authenticated key agreement AuthCombThree-B among three user sets. These procedures are invoked instead of CombineTwo and CombineThree in the key tree obtained by the procedure KeyAgreement in the unauthenticated protocol described above. We discuss AuthCombTwo, AuthCombThree-A, AuthCombThree-B informally and refer the reader to [17] for details. (See Figure 3.)

In AuthCombTwo, user instance $\Pi_{U_1}^{d_1}$ has a secret key s_1 with partial session-identity $\mathsf{psid}_{U_1}^{d_1} = \{(U_1, d_1)\}$. Besides it generates a random key $s \in Z_q^*$, computes a signature σ_1 using basic signature scheme DSig on $m_1 = (s_1P, sP)$ and sends $U_1|1|m_1|\sigma_1$ to user instance $\Pi_{U_2}^{d_2}$. Similarly, user instance $\Pi_{U_2}^{d_2}$ with secret key s_2 and partial session-identity $\mathsf{psid}_{U_2}^{d_2} = \{(U_2, d_2)\}$ computes a basic signature σ_2 on $m_2 = s_2P$ and sends $U_2|1|m_2|\sigma_2$ to user instance $\Pi_{U_1}^{d_1}$. On receiving the message $U_2|1|m_2|\sigma_2$, user U_1 verifies σ_2 on $U_2|1|m_2$

according to the verification algorithm $\mathcal{V}$ of DSig. If verification fails, it sets $\mathsf{acc}_{U_1}^{d_1} = 0$, $\mathsf{sk}_{U_1}^{d_1} = \mathsf{NULL}$ annd aborts. Otherwise it sets $\mathsf{psid}_{U_1}^{d_1} = \mathsf{psid}_{U_1}^{d_1} \cup \{(U_2, d_2)\}$ and computes the key $H(e(P,P)^{ss_1s_2})$. In a similar way, user U_2 performs verification σ_1 on $U_1|1|m_1$ and computes the key only if verification succeeds. Note that at the end, user instances $\Pi_{U_1}^{d_1}, \Pi_{U_2}^{d_2}$ have the same partial session-identity : $\mathsf{psid}_{U_1}^{d_1} = \mathsf{psid}_{U_1}^{d_1} \cup \mathsf{psid}_{U_2}^{d_2} = \mathsf{psid}_{U_2}^{d_2}$. An analogus description holds for AuthCombThree-A. The algorithm AuthCombThree-B performs key agreement among three user sets $\mathsf{US}_1, \mathsf{US}_2, \mathsf{US}_3$ as follows: Suppose $\Pi_{U_1}^{d_1}$ be the representative of user set US_1 and users in this set have a common agreed key s_1. Similarly, $\Pi_{U_2}^{d_2}$, s_2 are those for user set US_2 and $\Pi_{U_3}^{d_3}$, s_3 for user set US_3. Let $m_1 = \mathsf{psid}_{U_1}^{d_1}|t_1|s_1P$, where t_1 is the next expected message number to be sent by $\Pi_{U_1}^{d_1}$. For each user V in user set US_1, $\mathsf{psid}_V^{d_V}|t_1|s_1P$ is same as m_1. Each user $V \in \mathsf{US}_1$ computes a basic signature σ_V on m_1 using the scheme DSig and sends (V, σ_V) to $\Pi_{U_1}^{d_1}$. After accumulating these basic signatures, the representative $\Pi_{U_1}^{d_1}$ constructs the multi-signature msig_1 on message m_1 using the scheme MSig and sends $m_1|\mathsf{msig}_1$ to $\mathsf{US}_2 \cup \mathsf{US}_3$. Similarly, representative $\Pi_{U_2}^{d_2}$ of user set US_2 sends $m_2|\mathsf{msig}_2$ to $\mathsf{US}_1 \cup \mathsf{US}_3$ and representative $\Pi_{U_3}^{d_3}$ of user set US_3 sends $m_3|\mathsf{msig}_3$ to $\mathsf{US}_1 \cup \mathsf{US}_2$ where $m_2 = \mathsf{psid}_{U_2}^{d_2}|t_2|s_2P$ and $m_3 = \mathsf{psid}_{U_3}^{d_3}|t_3|s_3P$, t_2, t_3 being the next expected message number to be sent by $\Pi_{U_2}^{d_2}$, $\Pi_{U_3}^{d_3}$ respectively.

We define a variable $\mathsf{First}(\mathsf{psid}_U^d)$ for the ease of discussion. If $\mathsf{psid}_U^i = \{(U_{i_1}, d_{i_1}), \ldots, (U_{i_k}, d_{i_k})\}$, then $\mathsf{First}(\mathsf{psid}_U^i) = \{U_{i_1}, \ldots, U_{i_k}\}$. Now on receipt of messages $m_2|\mathsf{msig}_2$ and $m_3|\mathsf{msig}_3$, each user instance $\Pi_V^{d_V} \in \mathsf{US}_1$ (1) checks $\mathsf{First}(\mathsf{psid}_{U_2}^{d_2}) \subseteq \mathsf{pid}_V^{d_V}$ and $\mathsf{First}(\mathsf{psid}_{U_3}^{d_3}) \subseteq \mathsf{pid}_V^{d_V}$; (2) verifies t_2, t_3 are the next expected message number to be sent by $\Pi_{U_2}^{d_2}$, $\Pi_{U_3}^{d_3}$ respectively; (3) verifies $\mathsf{msig}_2, \mathsf{msig}_3$ are multi–signatures on m_2, m_3 respectively. If any of these verification fails, $\Pi_V^{d_V}$ sets $\mathsf{acc}_V^{d_V} = 0$ and $\mathsf{sk}_V^{d_V} = \mathsf{NULL}$ and aborts. Otherwise computes the common key $H(e(P,P)^{s_1s_2s_3})$ and sets $\mathsf{psid}_V^{d_V} = \mathsf{psid}_V^{d_V} \cup \mathsf{psid}_{U_2}^{d_2} \cup \mathsf{psid}_{U_3}^{d_3}$.

Similar verifications are done by each user in user sets US_2 and US_3. Common key $H(e(P,P)^{s_1s_2s_3})$ is computed only if verification holds. Note that at the end of an honest execution of this protocol, each user in the group $\mathsf{US}_1 \cup \mathsf{US}_2 \cup \mathsf{US}_3$ has a common partial session-identity.

3.3 Proposed Authenticated Dynamic Key Agreement Protocol

Dynamic key agreement consists of a key agreement protocol together with two additional algorithms, Insert and Delete. The procedure Insert enables a user to join a group. A user can leave a group by invoking the procedure Delete. In this subsection, we describe protocols for insertion and deletion for the above static tree-based authenticated protocol. Our protocol design makes an optimal use of the data precomputed in the procedure KeyAgreement. When a user joins or leaves a group, the structure of the key tree is disturbed and requires to be updated for any subsequent join or leave operation. Maintaining the tree structure of the key agreement protocol is a crucial part of our scheme. We refer this as the preservation of the structure of the procedure KeyAgreement.

Suppose we have a keytree T of n users $\{1, 2, \ldots, n\}$ according to the key agreement of [5] with $k = R(n)$ rounds. For the sake of easy description, we take the user set $\mathcal{P} = \{1, \ldots, n\}$ instead of the set $\{U_1, \ldots, U_n\}$ and introduce some more notations. For $1 \leq l \leq k$, let $U_i^{(l)}$ be the i-th user set at level l and $s_i^{(l)}$ be the common agreed key of users in the user set $U_i^{(l)}$ at level l. Initially, $U_i^{(0)} = \{i\}$, $s_i^{(0)}$ is the private key randomly chosen by user i from Z_q^*.

3.3.1 Insertion

Let a new user $\{u\}$ with private key x requests for join in the group $\{1, \ldots, n\}$. He joins the tree in such a way that the structure of KeyAgreement is still preserved and updation of key path is optimal in the

sense that minimum updation or recomputation is required. For instance, consider a group key agreement with $n = 10$ members. In this case, the root node will have 3 subtrees, with the left and middle subtrees having 3 leaves each and the right subtree having 4 leaves. Now suppose a new user wants to join this group. He cannot join the first subgroup (of subusers) since this is contrary to the way we partition the user sets. So the entire group of users will have to be repartitioned. Similarly, he cannot join the third subgroup (of 4 users) without causing repartitioning. But if he joins the second subtree, then there is no need of repartitioning and so key updation is minimal and is only along a single path. This is illustrated in Figure 4. The following Lemma determines uniquely such a path that we call the optimal path of joining of the new user.

Lemma 3.1 *For $1 \leq l \leq k$, $k = R(n)$, define the following:*
$i_l :=$ *the index of the node at level l whose subtree will contain the new user $\{u\}$ as a leaf,*
$\eta_l :=$ *the number of leaf nodes in the subtree at i_l,*
$N_l :=$ *number of leaf nodes to the left of node i_l and*
$r_l = \eta_l \bmod 3$.
Clearly $i_k = 1; \eta_k = n; r_k = n \bmod 3; N_k = 0$. Then, for $2 \leq l \leq k$, we have $i_{l-1} = 3i_l - r_l$; $\eta_{l-1} = \lfloor \frac{\eta_l}{3} \rfloor$; and $N_{l-1} = N_l + (2 - r_l)\eta_{l-1}$. Thus user u joins the subgroup at i_1 and $i_k, i_{k-1}, \ldots, i_1$ determine the optimal path along which the subgroup keys needs to be updated or recomputed.

Proof : By induction. The case $l = k$ is straightforward. Suppose the results hold for $l < k$. The node i_{l-1} is the $(3 - r_l)$-th child of the subtree T at i_l and there are $3(i_l - 1)$ nodes at level $(l - 1)$ to the left of subtree T in the original tree. Hence $i_{l-1} = 3(i_l - 1) + (3 - r_l) = 3i_l - r_l$.

The second one is easy to check. For the third one, we argue as follows: There are η_l leaf nodes in the subtree T at i_l and i_l has exactly three offsprings with subtrees T_L (left), T_M (middle) and T_R (right). If $r_l = 0$, *i.e.* if $|T_L| = |T_M| = |T_R|$, then i_{l-1} is the third child; if $r_l = 1$, then i_{l-1} is the second child and if $r_l = 2$, then i_{l-1} is the first child. In any case, the number of leaf nodes in the subtree at i_{l-1} is η_{l-1}.

In case 1 above, there are $2\eta_{l-1}$ leaf nodes to the left of node i_{l-1} in the subree T at i_l. In case 2, there are η_{l-1} leaf nodes to the left and in case 3, there are zero leaf nodes to the left. In each case, there are $(2 - r_l)\eta_{l-1}$ leaf nodes to the left of i_{l-1} in T. Since there are N_l leaf nodes to the left of i_l in the original tree, the total number of leaf nodes to the left of i_{l-1} in $N_l + (2 - r_l)\ \eta_{l-1} = N_{l-1}$, by definition of N_{l-1}. This completes the proof. ■

We now describe our protocol Insert. This algorithm invokes a procedure FindKeyPath to find the optimal key path path of joining of a new member and updates path according to the algorithm UpdateKeyPath. The new user who has permission to join the group for a group key agreement computes it's optimal key path of joining $i_k, \ldots, i_1; N_k, \ldots, N_1$ by using algorithm FindKeyPath, communicate this to all other members of the group and invokes algorithm UpdateKeyPath to update this path. The formal description of the procedures FindKeyPath and UpdateKeyPath are given below.

procedure FindKeyPath(n)
1. $k = R(n); i_k = 1; N_k = 0; \eta_k = n; r_k = n \bmod 3;$
2. $l = k$ downto 2 **do**
3. $\quad i_{l-1} = 3i_l - r_l;\ \eta_{l-1} = \lfloor \frac{\eta_l}{3} \rfloor;\ r_{l-1} = \eta_{l-1} \bmod 3;\ N_{l-1} = N_l + (2 - r_l)\eta_{l-1};$
4. **end do**
5. **return** $(i_k, \ldots, i_1; \eta_k, \ldots, \eta_1; N_k, \ldots, N_1);$

end FindKeyPath

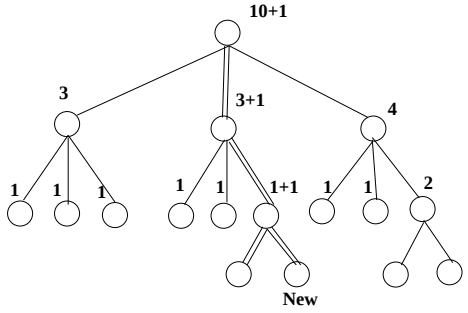

Figure 4: **procedure** Insert

procedure UpdateKeyPath(path)

1. parse path $= (i_k, \ldots, i_1; \eta_k, \ldots, \eta_1; N_k, \ldots, N_1)$;
2. The node i_1 at level 1 has η_1 children which are clearly leaf nodes. Note that $\eta_1 \leq 3$.
3. **if** $\eta_1 = 1$ **then**
4. user $N_{i_1} + 1$ chooses a new private key $s^{(0)}_{N_{i_1}+1} \in_R Z_q^*$;
5. $\hat{U}_1 = U^{(0)}_{N_{i_1}+1}$; $\hat{s}_1 = s^{(0)}_{N_{i_1}+1}$; $\hat{U}_2 = \{u\}$; $\hat{s}_2 = x$;
6. **call** AuthCombTwo$(\hat{U}[1,2], \hat{s}[1,2])$;
7. $s^{(1)}_{i_1}$ is assigned the agreed key between user sets $\hat{U}_1, \hat{U}_2$ and $\hat{U}^{(1)}_{i_1} = \hat{U}[1,2]$;
8. **end if**
9. **if** $\eta_1 = 2$ **then**
10. users $N_{i_1} + 1, N_{i_1} + 2$ choose new private keys $s^{(0)}_{N_{i_1}+1}, s^{(0)}_{N_{i_1}+2} \in_R Z_q^*$ respetively;
11. $\hat{U}_1 = U^{(0)}_{N_{i_1}+1}$; $\hat{s}_1 = s^{(0)}_{N_{i_1}+1}$; $\hat{U}_2 = U^{(0)}_{N_{i_1}+2}$; $\hat{s}_2 = s^{(0)}_{N_{i_1}+2}$; $\hat{U}_3 = \{u\}$; $\hat{s}_3 = x$;
12. **call** AuthCombThree $-$ A$(\hat{U}[1,2,3], \hat{s}[1,2,3])$;
13. $s^{(1)}_{i_1}$ is assigned the agreed key between user sets $\hat{U}_1, \hat{U}_2, \hat{U}_3$ and $\hat{U}^{(1)}_{i_1} = \hat{U}[1,2,3]$;
14. **end if**
15. **if** $\eta_1 = 3$ **then**
16. users $N_{i_1} + 1, N_{i_1} + 2, N_{i_1} + 3$ choose new private keys $s^{(0)}_{N_{i_1}+1}, s^{(0)}_{N_{i_1}+2}, s^{(0)}_{N_{i_1}+3}$ respectively;
17. $\hat{U}_1 = U^{(0)}_{N_{i_1}+3}$; $\hat{s}_1 = s^{(0)}_{N_{i_1}+3}$; $\hat{U}_2 = \{u\}$; $\hat{s}_2 = x$;
18. **call** AuthCombTwo$(\hat{U}[1,2], \hat{s}[1,2])$;
19. $s^{(0)}_{N_{i_1}+3}$ is assigned the agreed key between user sets $\hat{U}_1, \hat{U}_2$
20. $\hat{U}_3 = \hat{U}[1,2]$; $\hat{s}_3 = s^{(0)}_{N_{i_1}+3}$; $\hat{U}_2 = U^{(0)}_{N_{i_1}+2}$; $\hat{s}_2 = s^{(0)}_{N_{i_1}+2}$; $\hat{U}_1 = U^{(0)}_{N_{i_1}+1}$; $\hat{s}_1 = s^{(0)}_{N_{i_1}+1}$;
21. **call** AuthCombThree $-$ B$(\hat{U}[1,2,3], \hat{s}[1,2,3])$;
22. $s^{(1)}_{i_1}$ is assigned the agreed key between user sets $\hat{U}_1, \hat{U}_2, \hat{U}_3$ and $\hat{U}^{(1)}_{i_1} = \hat{U}[1,2,3]$;
23. **end if**
24. $l = 1$ to $k - 1$ **do**
25. $j = i_l$;
26. **if** $i_l = 3i_{l+1} - 2$ *i.e.* $r_{l+1} = 2$ **then**
27. **call** AuthCombThree $-$ B$(U^{(l)}[j, j+1, j+2], s^{(l)}[j, j+1, j+2])$;
28. $s^{(l+1)}_{i_{l+1}}$ is assigned the agreed key among user sets $\hat{U}^{(l)}_j, \hat{U}^{(l)}_{j+1}, \hat{U}^{(l)}_{j+2}$; $U^{(l+1)}_{i_{l+1}} = U^{(l)}[j, j+1, j+2]$;

29. **end if**
30. **if** $i_l = 3i_{l+1} - 1$ *i.e.* $r_{l+1} = 1$ **then**
31. **call** AuthCombThree − B$(U^{(l)}[j-1, j, j+1], s^{(l)}[j-1, j, j+1])$;
32. $s_{i_{l+1}}^{(l+1)}$ is assigned the agreed key among user sets $\hat{U}_{j-1}^{(l)}, \hat{U}_j^{(l)}, \hat{U}_{j+1}^{(l)}$; $U_{i_{l+1}}^{(l+1)} = U^{(l)}[j-1, j, j+1]$;
33. **end if**
34. **if** $i_l = 3i_{l+1}$ *i.e.* $r_{l+1} = 0$ **then**
35. **call** AuthCombThree − B$(U^{(l)}[j-2, j-1, j], s^{(l)}[j-2, j-1, j])$;
36. $s_{i_{l+1}}^{(l+1)}$ is assigned the agreed key among user sets $\hat{U}_{j-2}^{(l)}, \hat{U}_{j-1}^{(l)}, \hat{U}_j^{(l)}$; $U_{i_{l+1}}^{(l+1)} = U^{(l)}[j-2, j-1, j]$;
37. **end if**
38. **end do**

end UpdateKeyPath

The algorithm UpdateKeyPath works as follows to update keys in level 1 on joining of the new user. In the key tree T with n users, the number of children of the node i_1 (node at level 1) in the optimal key path is either 1 or 2 or 3. If i_1 has 1 leaf node, then the user corresponding to this leaf node chooses a new private key, agree upon a common key with the new user by invoking algorithm AuthCombTwo and the corresponding user set for level 1 is modified to a set that includes these two users. In case i_1 has 2 leaf nodes, the users corresponding to these leaves choose new private keys, a new user set for level 1 is constructed that contains these two users and the new user and algorithm AuthCombThree-A is invoked to agree upon a common key among them. If i_1 has 3 leaves, then the users corresponding to these leaves choose new private keys. The rightmost user agree upon a common key with the new user by invoking algorithm AuthCombTwo and constructs a new user set that consists of the new user and itself. Then AuthCombThree-B is invoked for this new user set and the the other two leaves of i_1 to agree upon a common key. Finally the corresponding user set for level 1 is modified to a set that includes these users.

The subsequent user sets are accordingly changed by algorithm UpdateKeyPath and key updates in level $l+1$ $(1 \leq l \leq k-1)$ are done by invoking algorithm AuthCombThree-B among the three user sets which are subtrees of node i_{l+1}. The modified user set corresponding to the node i_l invokes AuthCombThree-B to agree upon a common key with the user sets corresponding to the other two subtrees (siblings of i_l) of node i_{l+1} and a new user set for level $l+1$ is constructed that is the union of these three user sets. We proceed in this way and finally a common key is agreed among all the $n+1$ users. At the end, we newly index the members (leaves) as $\{1, 2, \ldots, n+1\}$.

Remark 3.2 *The insertion algorithm presented above is an authenticated algorithm. If we invoke* CombineTwo *in lines 6, 18 instead of* AuthCombTwo *and* CombineThree *in lines 12, 21, 27, 31, 35 instead of* AuthCombThree-A *and* AuthCombThree-B*, we get an unauthenticated version of the insertion algorithm. We require* $k = \lfloor \log_3 n \rfloor + 1$ *key updates in the execution of* Insert*. After insertion of the new member,* $R(n+1) = k+1$ *if* $n = 3^k$*, and* k *otherwise.*

3.3.2 Deletion

Suppose key tree T with n leaf nodes $\{1, 2, \ldots, n\}$ has the tree structure used in the procedure KeyAgreement and suppose a member j_0, $1 \leq j_0 \leq n$, wants to leave the group. For this we first introduce a procedure Extract which outputs the identity or index of a leaf node in T such that the structure of KeyAgreement is preserved in the tree after removal of this node. We take a designated user, called group controller (GC) to initiate the operation Delete. To be specific, we take one sibling of the node leaving the group as the GC which is trusted only for this purpose.

The procedure Extract works as follows. If all the three subtrees T_L, T_M, T_R of the tree T have equal number of leaf nodes, then removal of a leaf node from the leftmost subtree T_L will not disturb the tree structure and so we can extract a leaf node from T_L. Similarly, if the number of leaves in both T_L, T_M are same, say p, and that of right subtree T_R is $p+1$, then we can extract a leaf from T_R without disturbing the tree structure. If the number of leaves in T_M, T_R are same, say, $p+1$ and that of T_L is p, then we can extract a leaf from T_M retaining the tree structure. We recursively apply this procedure on T_L, T_R or T_M chosen in this manner and finally reach to a leaf node. The index of the user corresponding to this leaf node is outputed to the GC.

procedure Extract (T, n)
1. **if** $n = 1$ **then return** the index of the leaf node to GC;
2. **if** $n = 2$ **then return** the index of the left leaf node to GC;
3. Let T_L, T_M, T_R be respectively the left, middle and right subtree of T;
4. $p = \lfloor \frac{n}{3} \rfloor$; $r = n \bmod 3$;
5. **while** $(p \geq 1)$ **do**
6. **if** $r = 0$ *i.e.* $|T_L| = |T_M| = |T_R| = p$ **then**
7. **call** Extract (T_L, p);
8. **end if**
9. **if** $r = 1$ *i.e.* $|T_L| = |T_M| = p, |T_R| = p + 1$ **then**
10. **call** Extract $(T_R, p + 1)$;
11. **end if**
12. **if** $r = 2$ *i.e.* $|T_L| = p, |T_R| = |T_M| = p + 1$ **then**
13. **call** Extract $(T_M, p + 1)$;
14. **end if**
15. **end do**

end Extract

Now a user corresponding to leaf j_0 leaves the group, the tree structure is disturbed. We first find the highest level, say i, for which the subtree rooted at the internal node at, say j_i, lacks the tree structure. Consequently, j_i is the root of the highest level subtree with disturbed tree structure on removal of j_0. To retain the the tree structure, we may need to extract suitably a leaf node l_0 from the tree T in such a way that the key updates required are minimal. We do this by using an algorithm FindExtractNode that uses the procedure Extract as a subroutine and removes the leaving member j_0, still preserving the structure of KeyAgreement in the resulting key tree. This algorithm outputs the index of the extracted leaf node l_0 to GC and also the index of the internal node j_i. The procedure FindExtractNode is formally described below.

procedure FindExtractNode (T, n, j_0)
1. $k = R(n)$;
2. After removing node j_0, let $i(\leq k)$ be the highest level for which the subtree rooted at j_i lacks the structure of KeyAgreement. (In case $i < k$, this structure is retained in the other two siblings of j_i);
3. Let the left, middle and right subtrees of offsprings of the node j_i be T_L, T_M, T_R respectively;
4. **Case 1 :** Before removing j_0, let $|T_L| = |T_M| = |T_R| = p$; remove node j_0;
5. **if** (j_0 is leaf node of T_M or T_R) **then**
6. **call** Extract (T_L, p);
7. **end if**

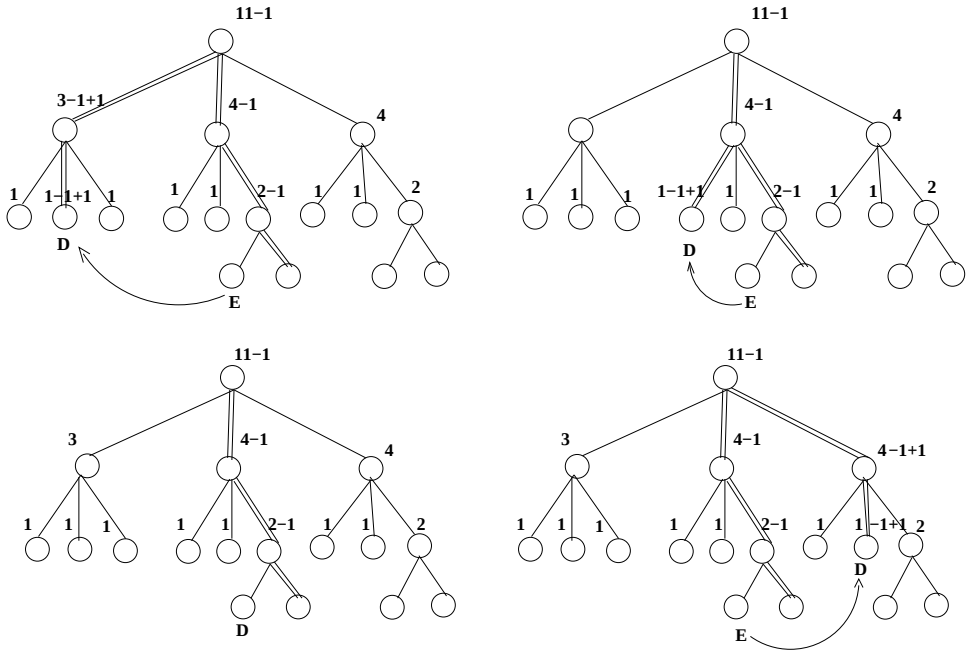

Figure 5: Different cases of **procedure** Delete with $n = 11$ (D denotes the node to be deleted and E denote the node to be extracted to maintain the key structure)

8. **Case 2 :** Before removing j_0, let $|T_L| = |T_M| = p; |T_R| = p + 1$; remove node j_0;
9. **if** (j_0 is leaf node of T_L or T_M) **then**
10. **call** Extract $(T_R, p + 1)$;
11. **end if**
12. **Case 3 :** Before removing j_0, let $|T_L| = p$; $|T_M| = |T_R| = p + 1$; remove node j_0;
13. **if** (j_0 is leaf node of T_L or T_R) **then**
14. **call** Extract $(T_M, p + 1)$;
15. **end if**
16. Let l_0 be the extracted node to be inserted as the j_0-th leaf in the tree T;
17. **return** (l_0, j_i);

end FindExtractNode

Next we describe the algorithm Delete below. Our algorithm Delete invokes FindExtractNode to obtain the index l_0 of the node to be extracted (if required), finds from the tree T the path from root to the parent of the leaving leaf node j_0 and also the path from root to the parent of the extracted leaf node l_0. Two new user sets are constructed in level 1: one user set includes the user corresponding to l_0 together with the users corresponding to brothers of j_0 and another user set includes only the users corresponding to brothers of l_0. All the users in these two new user sets choose new private keys. The subsequent higher level user sets are modified accordingly and appropriate algorithms AuthCombTwo, AuthCombThree-A or AuthCombThree-B are invoked for successive key agreements in the key tree. We invoke AuthCombTwo for two party authenticated key agreement, AuthCombThree-A for three party authenticated key agreement and AuthCombThree-B for authenticated key agreement among three user sets. At the end of the procedure Delete, we newly index the users (leaves) as $\{1, 2, \ldots, n-1\}$. Some particular cases are shown in Figure 5 (See Appendix.)

procedure Delete (T, n, j_0)

1. Let $k = R(n)$.
2. **call** FindExtractNode(T, n, j_0);
3. Let (l_0, j_i) be it's output.
4. **if** $l_0 = j_0$ **then**
5. let $\mathsf{path}_D = (j_k, \ldots, j_1)$ be the path from root to parent of j_0. The siblings of user j_0 choose random new keys and form a new user set $U_{j_1}^{(1)}$. The subsequent user sets $U_{j_t}^{(t)}$ are accordingly modified replacing old $U_{j_1}^{(1)}$ by the new one and the keys updated as in UpdateKeyPath.
6. **end if**
7. **else**
8. let $\mathsf{path}_D = (j_k, \ldots, j_i, j_{i-1}, \ldots, j_1)$ and $\mathsf{path}_E = (j_k, \ldots, j_i, l_{i-1}, \ldots, l_1)$ be the paths from root to the parent of the leaving leaf node j_0 and extracted leaf node l_0 respectively, where j_t denotes the index of the node at level t whose subtree contains the leaving node j_0 and l_t denotes the index of the node at level t whose subtree contains the extracted node l_0.
9. User l_0 and its brothers choose random new keys; also brothers of user j_0 choose random new keys;
10. User l_0 and brothers of j_0 construct new user set $U_{j_1}^{(1)}$; brothers of user l_0 constructs new set $U_{l_1}^{(1)}$.
11. The subsequent user sets $U_{j_t}^{(t)}$, $U_{l_t}^{(t)}$, for $2 \leq t \leq k$, are modified accordingly replacing the old $U_{j_1}^{(1)}$ and $U_{l_1}^{(1)}$ values by their new values. Note that $|U_{l_1}^{(1)}| \neq 0$.
12. $t = 1$ to $i - 1$ **do in parallel**
13. • user sets of level $t-1$ in $U_{j_t}^{(t)}$ agree upon a common key by invoking appropriate subroutine for authenticated key agreement depending upon the cardinality of $U_{j_t}^{(t)}$.
14. • user sets of level $t-1$ in $U_{l_t}^{(t)}$ agree upon a common key by invoking appropriate subroutine for authenticated key agreement depending upon the cardinality of $U_{l_t}^{(t)}$.
15. **end do**;
16. $t = i$ to k **do**
17. user sets of level $t-1$ in $U_{j_t}^{(t)}$ agree upon a common key by invoking appropriate subroutine for authenticated key agreement depending upon the cardinality of $U_{j_t}^{(t)}$.
18. **end do**;
19. **end else**

end Delete

Remark 3.3 *The invocation of* AuthCombTwo, AuthCombThree-A *and* AuthCombThree-B *in the above procedure make the protocol* Delete *authenticated. If instead, we use unauthenticated* CombineTwo *and* CombineThree, *an unauthenticated version of the deletion algorithm is obtained. At most* $2\lfloor \log_3(n+1) \rfloor$ *key updates are required in the execution of* Delete. *After deletion of a member,* $R(n-1) = k+1$ *if* $n = 3^k + 1$, *and* k *otherwise.*

4 Security Analysis

We will show that our dynamic authenticated key agreement protocol DAP is secure in the model as described in Subsection 2.2. In fact, we can convert any active adversary attacking the protocol DAP into a passive adversary attacking the unauthenticated protocol UP assuming that both DSig and MSig are secure and DHBDH problem is hard. No Corrupt query appears since long term secret keys are not used.

So our protocol trivially achieves forward secrecy. We state the security results of the unauthenticated protocol [5] UP and authenticated protocol [17] in Theorem 4.1 and Theorem 4.2. For the proofs of these results, see [5, 17]. We prove our main result in Theoren 4.3 that states the security of our dynamic authenticated key agreement protocol DAP.

Theorem 4.1 *[5] The group key agreement protocol* UP *described in Section 3.1 is secure against passive adversaries under the assumption that DHBDH problem is hard.*

Theorem 4.2 *[17] The group key agreement protocol* AP *described in Section 3.2 satisfies the following:*

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

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

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

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

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,* q_E, q_J, q_L *and* q_S *are respectively the maximum number of execute, join, leave and send queries that an adversary can make.*

Proof (Sketch) : 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 [17], we have the following claim.
Claim : Let Forge be the event that a signature (either of DSig or of MSig) is forged by $\mathcal{A}'$. Then $\mathsf{Prob}[\mathsf{Forge}] \leq |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{MSig}}(t') + |\mathcal{P}|\ \mathsf{Succ}_{\mathsf{DSig}}(t')$.

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 oracle. We provide details below.
Execute and Send queries: These queries are simulated as in [17]. Apart from the usual send queries, there are two special type of send queries, Send_J and Send_L. If an unused instance Π^d_U wants to join the group $\Pi^{d_1}_{U_{i_1}}, \ldots, \Pi^{d_k}_{U_{i_k}}$, then $\mathcal{A}'$ will make $\mathsf{Send}_J(U, d, \langle U_{i_1}, \ldots, U_{i_k}\rangle)$ query. This query initiates $\mathsf{Join}(\{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}, (U, d))$ query . $\mathcal{A}$ first finds a unique entry of the form (S, T) in Tlist with $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$. If no such entry, $\mathcal{A}$ makes an execute query to its own execute oracle on S and gets a transcript T. $\mathcal{A}$ then stores $(S, U|d, T)$ in Jlist. Similarly, when $\mathsf{Send}_L(U, d, \langle U_{i_1}, \ldots, U_{i_k}\rangle)$ query is made, $\mathcal{A}$ stores $(S, U|d, T)$ in Llist.
Join queries : Suppose $\mathcal{A}'$ makes a query $\mathsf{Join}(\{(U_{i_1}, d_1), \ldots, , (U_{i_k}, d_k)\}, (U, d))$. $\mathcal{A}$ finds an entry of the form $(S, U|d, T)$ in Jlist where $S = \{(U_{i_1}, d_1), \ldots, (U_{i_k}, d_k)\}$. If no such entry, then the adversary $\mathcal{A}'$ is given no output. Otherwise, $\mathcal{A}$ modifies T as follows: $\mathcal{A}$ can find the path of joining of U in the key tree with leafs $U_{i_1}, \ldots, U_{i_k}$ and detect the positions in T where the new messages are to be injected or where the old messages are to be replaced by new messages. $\mathcal{A}$ does these modifications in T according to the unauthenticated version (see Remark 3.2) of the algorithm Insert described in Section 3.3.1 and gets

a modified transcript T_M. It then patches appropriate basic signatures and multi-signatures with each message in T_M according to the modifications described in Section 3.2. Thus $\mathcal{A}$ expands the transcript T_M into a transcript T' for DAP. It returns T' to $\mathcal{A}'$.

Leave queries : These queries are simulated as Join queries with modified transcript T_M obtained from unauthenticated transcript T according to the algorithm Delete in Section 3.3.2.

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

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

This yields the statement of the theorem. ■

5 Conclusion

We describe a dynamic group key agreement in tree-based setting and prove that the scheme is provably secure under the assumption that DHBDH problem is hard. Our protocol is an extension of the protocols of Barua *et al.* [5] and Dutta *et al.* [17]. The bilinear pairing based Joux [21] protocol and multi-signature by Boldyreva [12] are used. The protocol is proven to be secure in the standard formalized security model of Bresson *et al.* [14] by appropriately modifying the Katz-Yung [22] technique to tree-based setting. One can be benifited in both communication and computation power by combining our protocol with an efficient constant round protocol. Designing such hybrid group key agreement protocols may be desirable for certain applications, in particular when large number of user groups are concerned.

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.

[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. *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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

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

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

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

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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