This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2005 proceedings.

Minimization of Rekeying Cost for Contributory Group Communications

Wei Yu*, Yan Sun† and K. J. Ray Liu*
*Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742
Email: weiyu, kjrliu@umd.edu
†Department of Electrical and Computer Engineering, University of Rhode Island, 4 East Alumni Ave, Kingston, RI 02881
Email: yansun@ele.uri.edu

***Abstract*— While contributory group key agreement is a promising solution to achieve access control in collaborative and dynamic group applications, the existing schemes have not achieved the performance lower bound in terms of rekeying overhead. In this paper we introduce a contributory group key agreement that achieves the performance lower bound by utilizing a novel logical key tree structure, called PFMH, and the concept of phantom user position. In particular, the proposed scheme only needs $O(1)$ rounds of two-party Diffie-Hellman upon any single user join event and $O(\log n)$ rounds of two-party Diffie-Hellman upon any single user leave event. Both theoretical bound analysis and simulation studies show that the proposed scheme achieves much lower rekeying cost than the existing tree-based contributory group key agreement schemes.**

I. Introduction

Contributory group key agreement is a promising solution to achieve access control in collaborative and dynamic group applications [1]–[8]. However, the early design of contributory group key agreement schemes mostly considers the efficiency of initial group key establishment [1]–[3], which encounter high rekeying cost upon group membership changes. Recently, logical tree structures are used to improve the scalability of contributory key agreements [7], [8]. Kim et al. proposed a tree-based contributory group key agreement protocol called TGDH, where binary balanced tree is adopted to maintain the keying material [7]. In TGDH, the group key can be updated by performing $\log n$ rounds of two-party Diffie-Hellman (DH) upon any single user join or departure, where n is the group size. Mao et al. proposed another tree-based contributory key agreement scheme called DST [8]. By using a special join-tree/exit-tree topology and exploiting cost amortization, DST can reduce the average time cost to $\Theta(\log \log n)$ rounds of two-party DH for single user join or departure. However, DST has an unrealistic requirement that members need to know other members' exact departure time in advance.

The theoretical analysis in [9] indicates that for any tree-based contributory group key management scheme, the lower bound of the worst case cost is $\Theta(\log n)$ rounds two-party DH for either user adding or deleting. That is, either the cost for adding a user or the cost for deleting a user is no less than $\Theta(\log n)$. In addition, it is obvious that at least one round of two-party DH needs to be performed for adding or deleting a user in any circumstance. Therefore, lowest possible cost for contributory group key agreement is (1) $\Theta(\log n)$ for user join and $O(1)$ for user departure; or (2) $\Theta(\log n)$ for user departure and $O(1)$ for user join. Both TGDH and DST do not achieve these lower bounds.

To achieve the lower bound of the rekeying cost, in this paper we propose a novel and efficient logical key tree structure, called PFMH tree, as well as a cost-minimizing PFMH tree-based contributory group key agreement protocol suite (PACK) that handles dynamic group membership events. The optimality of the proposed PACK protocol suite lies in that it only needs $O(1)$ rounds of two-party DH upon any single user join event and $O(\log n)$ rounds of two-party DH upon any single user leave event, which achieves the lower bound. Both theoretical analysis and simulation studies show that the PACK has much lower rekeying cost than the existing tree-based contributory group key agreements.

The rest of this paper is organized as follows. Section II introduces security requirement and performance metrics. Section III presents the proposed PFMH tree structure and two basic operations to manage PFMH key trees. Section IV describes the proposed PFMH tree-based contributory group key agreement protocol suite. Performance evaluation is presented in Section V. Finally, conclusion is drawn in Section VI.

II. Security Requirement and Performance Metric

Group key management schemes must be able to adjust group secrets subsequent to membership changes, including *user addition* and *user deletion* [7]. The security requirements with dynamic membership include *group key secrecy*, *forward secrecy*, *backward secrecy*, and *key independence* [7]. The overhead of group key agreement involves *computation cost*, *communication cost* and *time cost*. Since most existing contributory key agreement schemes use two-party DH protocol [10] as a basic building module, the computation cost comes mainly from the cryptographic primitives that are needed to perform two-party DH, such as modular exponentiation, and the communication cost comes from sending/receiving rekeying messages. In contributory group key agreement, many operations can be performed in parallel. Thus, the time cost is used to describe the latency in key generation.

Next we analyze the cost corresponding to the implementation of two-group DH (two-party DH among two groups), which is the basic building module for most tree-based

This work was supported in part by the Army Research Office under URI Award No. DAAD19-01-1-0494

contributory schemes. Let C_{cast} denote the communication cost needed per multicast/broadcast, and let C_{me} denote the computation cost of a modular exponentiation operation. In this paper, the implementation method proposed in [7] for two-group DH is used. Then for each round of two-group DH, the computation cost is no more than $(n+2)C_{me}$ with n being the total number of users in both subgroups, and the communication cost is $2C_{cast}$.

III. PFMH Key Tree Structure

In tree-based contributory group key agreement schemes, keys are organized in a logical tree structure, referred to as the *key tree*. In the key tree, the root node represents the group key, leaf nodes represent members' private keys, and each intermediate node corresponds to a subgroup key shared by all the members (leaf nodes) under this node. The key of each non-leaf node is generated by performing two-party DH between the two subgroups represented by its two children [7]. Since two-group DH is used, the obtained key tree is binary. For each node in the key tree, *key-path* denotes the path from this node to the root, and *co-path* denotes the sequence of siblings of each node on its key-path. In order to compute the group key, a node only needs to know its own key and all the blinded keys on its co-path [7].

In general the departure users can leave the key tree from arbitrary positions. In fact, for user departure, the best tree structure that reduces the worst-case rekeying overhead is a balanced key tree, when group members have similar computation and communication capability. When using the balanced key tree, the rekeying cost for user departure can be $O(\log n)$. Thus, in order to achieve the lower bound of the rekeying cost, the cost for user join must be $O(1)$. One strategy that achieves such a low user join cost is to insert the join user always at the root of the key tree. However, with multiple join users, this strategy results in a maximum height (MH) key tree (which is defined next), which is sort of the opposite of balanced key tree.

Therefore, in order to achieve the lower bound for user join and departure simultaneously, we take into consideration of both trees and propose a novel and efficient key tree structure for contributory group key agreement schemes, which we refer to as PFMH tree. PFMH tree is a combination of two special key tree structures: *partially-full* (PF) key tree and *maximum height* (MH) key tree. In the following of this paper, the size of a key tree is defined as the total number of leaf nodes in this tree, the function $\log()$ and $\log_2()$ will be used exchangeably, and when we say a "full (key) tree", we always mean a fully balanced binary (key) tree with certain size 2^n, where n is a non-negative integer.

Definition 1 *Let T be a binary key tree of size n, and let $n' = 2^{\lfloor \log n \rfloor}$. T is a PF key tree if and only if it satisfies one of the following properties: 1) T is a full key tree; 2) the left subtree of T is a full key tree with size n', and the right subtree of T is a PF key tree with size $(n-n')$.*

Definition 2 *A key tree T of size n is a MH key tree if and only if it satisfies one of the following properties: 1) $n=1$, and T is a tree with only one leaf node; 2) the right subtree of T is a leaf node, and the left subtree of T is a MH key tree with size $n-1$.*

Definition 3 *A key tree T of size n is a PFMH key tree if and only if it satisfies one of the following properties: 1) T is a PF key tree; 2) the left subtree of T is a PF tree, and the right subtree of T is a MH tree.*

Procedure 1 $unite(\{T_1, \ldots, T_L\})$

▷ $\mathcal{T} = \{T_1, \ldots, T_L\}$;
while ($\exists\, T, T' \in \mathcal{T}$ with $T \neq T'$ and $|T| = |T'|$) **do**
  Perform two-group DH between T and T' to generate a new full key tree;
  Remove T and T' from $\mathcal{T}$, and add the new generated full key tree to $\mathcal{T}$.
end while
while ($|\mathcal{T}| > 1$) **do**
  Let T and T' be the two trees with the smallest sizes in $\mathcal{T}$, and $|T| \geq |T'|$; perform two-group DH between them to generate a new key tree with the left subtree being T and the right subtree being T'; remove T and T' from $\mathcal{T}$, and add the new generated key tree to $\mathcal{T}$.
end while
The remaining tree T in $\mathcal{T}$ is the PF tree.

According to these definitions, we can see that the height of a PF key tree with size n is $\lceil \log n \rceil$, the height of a MH tree with size n is $n-1$. In this paper, without introducing ambiguity, we will use $\lceil \log n \rceil$ and $\log n$ exchangeably. Also, given a PFMH key tree T, we will use *main tree* to refer to the PF subtree of T, denoted by T_{main}, and use *join tree* to refer to the MH subtree of T, denoted by T_{join}. It is easy to see that the height of T_{main} is always bounded by $\log n$.

Next we describe two basic operations to manage PFMH trees: *unite* and *split*. Given a set of full key trees $\mathcal{T} = \{T_1, \ldots, T_L\}$, the *unite* operation is to combine these key trees into a single PF key tree. Given a key tree T, the *split* operation is to partition T into a set of full key trees with minimum set size. In general, given a set of full key trees, the result of *unite* operation may not be unique, but all the obtained PF trees have the same tree structure. Procedure 1 presents one possible implementations of *unite* operation, and Procedure 2 presents one possible implementation of *split* operation.

For *split* operation, each leaf node only needs to truncate the current tree to remove those nodes that do not belong to the full tree where this leaf node belongs to, so no extra cost is needed to recalculate keys. For *unite* operation, extra cost will be introduced when performing a sequence of two-group DH to generate new keys. Let L be the number of full key trees before performing *unite* operation, and let n be the total number of leaf nodes in all these trees, then we can have:

Proposition: Let $\mathcal{T} = \{T_1, \ldots, T_L\}$ be a set of full key trees with $\sum_{i=1}^{L} |T_i| = n$ and $|T_1| \geq |T_2| \geq \cdots \geq |T_L|$, by exploiting possible parallelism, the costs associated to $unite(\mathcal{T})$ operation are as follows:

1) The time cost, which is the maximum rounds of two-group DH, is bounded by $\log n$.
2) The total communication cost is $2(L-1)C_{cast}$.

Procedure 2 $split(T)$

if (T is a full tree) **then**
  Return $\{T\}$;
else if (T is empty) **then**
  Return $\emptyset$.
else
  Let T_{left} and T_{right} be the left and right subtrees of T;
  Return $split(T_{left}) \bigcup split(T_{right})$.
end if

3) If $|T_i| = 1$ for all $1 \leq i \leq L$, the total computation cost is bounded by $n(\log n + 1)C_{me}$.
4) If $|T_i| \neq |T_j|$ for any $i \neq j$, the total computation cost is bounded by $2(n + \log n)C_{me}$.
5) Consider the special situation that for each tree T_i, no more than 1 other tree has the same size with T_i in $\mathcal{T}$. If $|T_1| \geq n/2$, then the total computation cost can be approximately bounded by $2.5nC_{me}$, otherwise, the total computation cost can be approximately bounded by $3nC_{me}$.

IV. Description of PACK

In this section we describe the proposed PFMH tree-based contributory group key agreement protocol suite, referred to as PACK, to update the group key upon group membership change events.

A. User Join Protocol

When a prospective user M wants to join the group $\mathcal{G}$, it initiates the *user join* protocol by broadcasting a request message that contains its own blinded key. After receiving the join request message, the current group members check whether M has the privilege to join the group based on certain group policies. If M has the authorization to join, the group key needs be updated in order to incorporate a secret share from M and to guarantee group keys' backward secrecy. Procedure 3 describes the single user join protocol in PACK.

Procedure 3 $join(\mathcal{G}, M)$

▷ T is the PFMH key tree of group $\mathcal{G}$, T_{main} is the main tree of T, T_{join} is the join tree of T.

Create a leaf node to host user M;
if (T_{join} is empty) **then**
  Perform two-group DH between M and $\mathcal{G}$ to generate a new group key K. Create a node for K with its right child being M and its left child being T_{main}.
else
  Round 1: Perform two-group DH between M and T_{join} to generate a new subgroup key K_{join}. Create a node for K_{join} with its right child being M and its left child being the old T_{join}. Now K_{join} becomes the new root of the join tree.
  Round 2: Perform two-group DH between T_{main} and the new T_{join} to generate a new group key K, with its right child being the new T_{join}, and its left child being T_{main}.
end if

In PACK, the rekeying upon single user join needs to perform at most 2 rounds of two-group DH. If the join tree is not empty, a new join tree is generated by performing two-group DH between the new member and the old join tree, with the left subtree being the old join tree and the right subtree being the node hosting the new member. If the join tree is empty, the node hosting the new member becomes the join tree. The group key is generated by performing two-group DH between the new join tree and the main tree. In another words, the new member will be become the shallowest node in the join tree.

TABLE I
REKEYING COST UPON SINGLE USER JOIN EVENT

	time cost	communication cost	computation cost
case 1	1	$2C_{cast}$	nC_{me}
case 2	2	$4C_{cast}$	$(n + \|T_{join}\|)C_{me}$

Table I list the rekeying cost upon single user join event in PACK. Case 1 denotes the situation that the join tree is empty, and the protocol only needs to perform one round of two-group DH. Case 2 denotes the situation that the join tree is not empty, and the protocol needs to perform two rounds of two-group DH. For case 2, the $|T_{join}|$ term in the computation cost comes from performing two-group DH between the new member and the old join tree.

B. User Leave Protocol

When a current group member M wants to leave the group, it broadcasts a leave request message to initiate the user leave protocol. Once M leaves the group, the group key should be updated to remove M's share, and all the keys on M's key-path should be updated to maintain the group key's forward secrecy. In PACK, to reduce the rekeying cost upon user leave event, we introduce the concept of *phantom* node which allows an existing member to simultaneously occupy more than one leaf node in the key tree. In particular, when member M leaves the group, another group member M' is moved to M's position, generates a new secret key, and recursively updates the keys on M's previous key-path. However, after moving M' to M's position, the node which M' previously occupied will not be deleted immediately. As a result, now M' occupies two leaf nodes in the key tree. We call the node associated to M''s previous position as *phantom* node, which are known by all group members, and should be deleted no later than M' leaving the group to maintain group keys' forward secrecy. Procedure 4 describes the protocol upon single user leave event in PACK.

In PACK, upon single user leave event, based on the situation, different rekeying procedure is applied. The first situation happens when the leaving member M is in the join tree, and the size of join tree is no larger than $\log n$. For this situation, PACK simply removes the leaving member from the key tree, changes certain node's secret share, and recursively updates all the keys on M's key-path. Let h be M's level in T. Since at most h rounds of two-group DH operations need to be performed, the time cost is h, the communication cost is no more than $2hC_{cast}$, and the computation cost is about $C_{me}(n + \sum_{i=\log n - h}^{\log n} i)$, which is bounded by $C_{me}(n + 0.5\log^2 n)$.

Procedure 4 $Leave(\mathcal{G}, M)$

▷ T is the PFMH key tree of $\mathcal{G}$, and n is the current tree size.
▷ T_{main} is the main tree of T, T_{join} is the join tree of T.

if ($M \in T_{join}$) **then**
  if ($|T_{join}| \leq \log n$) **then**
    If M is the root of T_{join}, then simply remove M and the root from T. Otherwise, let P be M's sibling, remove M and M's parent from the key tree. If P has no children, change P's secret share, otherwise, change P's right child's secret share. Recursively update all the keys on P's key-path.
  else
    Remove all phantom nodes from T, remove M from T, and change M's sibling's secret share.
    $unite(split(T))$.
  end if
else if (M has a phantom node in T or $|T_{join}| > 1$ or M is the rightmost non-phantom leaf node) **then**
  Remove all phantom nodes from T, remove M from T, and change M's sibling's secret share.
  $unite(split(T))$.
else
  Find the rightmost non-phantom leaf node M' in T. Let P_{new} denote the node occupied by M, and P_{old} denote the node occupied by M'. M' moves to node P_{new} and generates a new secret share for it. If P_{old} lies in the join tree, then remove P_{old} and the root of T, otherwise, let P_{old} be M''s phantom node. Recursively update all the keys on P_{new}'s key path.
end if

The second situation happens when any of the following conditions is satisfied: 1) the leaving member M is in the join tree, and the size of join tree is larger than $\log n$; 2) M is in the main tree, and the size of join tree is larger than 1; 3) M is in the main tree, and is the rightmost non-phantom leaf node; 4) M is in the main tree, and has a phantom node. For this situation, the key tree will be reorganized to generate a new PF tree as the main tree, and the join tree will become empty. Let T_{join} and T_{main} be the join tree and the main tree after removing M and all phantom nodes but before reorganizing the tree, and let T_{left} be the left subtree of T_{main}. According to the **Proposition**, the time cost is bounded by $\log n$, the communication cost is bounded by $2(\log(n)+|T_{join}|)C_{cast}$ in most cases (condition 1, 2, and 3), where $\log n+|T_{join}|$ is the maximum number of full subtrees after split operation. When condition 4 is satisfied, the communication cost is bounded by $2(2\log(n) + |T_{join}|)C_{cast}$.

To analyze the computation cost for the second situation, we first consider the most general case where the height of the main tree does not increase after the *unite* operation, that is, $T_{left} \geq n/2$. According to the **Proposition**, in this case, the computation cost can be approximately bounded by $C_{me}(2.5n + T_{join}\log(|T_{join}|))$, where the term $T_{join}\log(|T_{join}|)$ comes from merging the nodes from the join tree into a set of full subtrees with different sizes. If $T_{left} < n/2$, which is a rare case, the approximate upper bound becomes $(3n + |T_{join}|\log(|T_{join}|))C_{me}$.

The third situation happens when neither of the first two situations is applicable. For this situation, the leaving member M is removed from the key tree, and M', which is the member who occupies the right-most non-phantom leaf node, moves to M's previous position, generates a secret share for this node, and recursively updates all the keys on this node's key-path. Now, M' occupies two positions, and the original position is called M''s phantom position. It is easy to see that the time cost is bounded by $\log n$, the communication cost is bounded by $2\log(n)C_{cast}$, and the computation cost can be approximately bounded by $(n + 2|T_{left}|)C_{me}$, where T_{left} is T_{main}'s left subtree.

TABLE II
REKEYING COST UPON SINGLE USER LEAVE EVENT

	time cost	communication cost	computation cost
case 1	h	$2hC_{cast}$	$(n + \frac{1}{2}n_j^2)C_{me}$
case 2	$\log n$	$2(\log n + n_j)C_{cast}$	$(2.5n + n_j \log n_j C_{me}$
case 3	$\log n$	$2(\log n)C_{cast}$	$(n + 2n_j)C_{me}$

TABLE III
REKEYING COST COMPARISON AMONG DIFFERENT SCHEMES

	time cost	communication cost	computation cost
Upon Single User Join Event			
PACK	$1 \sim 2$	$2 \sim 4C_{cast}$	nC_{me}
TGDH	$\log n$	$2(\log n)C_{cast}$	$2nC_{me}$
DST	$1 + \log\log n$	$(1 + \log\log n)C_{cast}$	$(n + \log n)C_{me}$
Upon Single User Leave Event			
PACK	$\log n$	$2(\log n)C_{cast}$	$(1 \sim 2.5)nC_{me}$
TGDH	$\log n$	$2(\log n)C_{cast}$	$2nC_{me}$
DST	$1 + \log nn$ $+ \log\log n$	$2(1 + \log n$ $+ \log\log n)C_{cast}$	$3nC_{me}$

Table II summarizes the rekeying cost upon single user leave events under different situations, where n_j denotes the size of T_{join}. Usually we have $|T_{left}| \geq n/2$, $h \simeq \frac{1}{2}\log n$, $n_j \ll n$, and the average size of T_{left} is about $0.75n$. For the second and third situation, in most cases we can simplify the upper bound of computation cost as $2.5nC_{me}$. For the first situation, we can simplify the bound of computation cost as nC_{me}.

V. Performance Evaluation and Comparison

Similar arguments as in [7] can be used to show that PACK also satisfies all four security requirements. Next we compare the rekeying cost in PACK upon single user join and leave events with two existing tree-based contributory group key agreement schemes: TGDH [7] and DST [8]. All three types of cost are considered: time, computation, and communication. Since in general members' leaving time is not known in advance, in DST, only join-tree is used. Table III lists the approximate bounds of different costs for the three schemes.

From the above comparison, we can see that PACK has the lowest cost in terms of time, computation, and communication. For example, for user join, only 1 or 2 rounds are needed in time cost, while DST needs $1 + \log\log n$ rounds and TGDH needs $\log n$ rounds. Similar results can also be seen in communication cost for user join. For the total computation cost computed as the average of user join cost and leave cost, DST has similar cost as TGDH, which is an order of $2n$, while for PACK, the order is from n to $1.75n$, with the saving ranging from 15% to 50% compared with DST and TGDH.

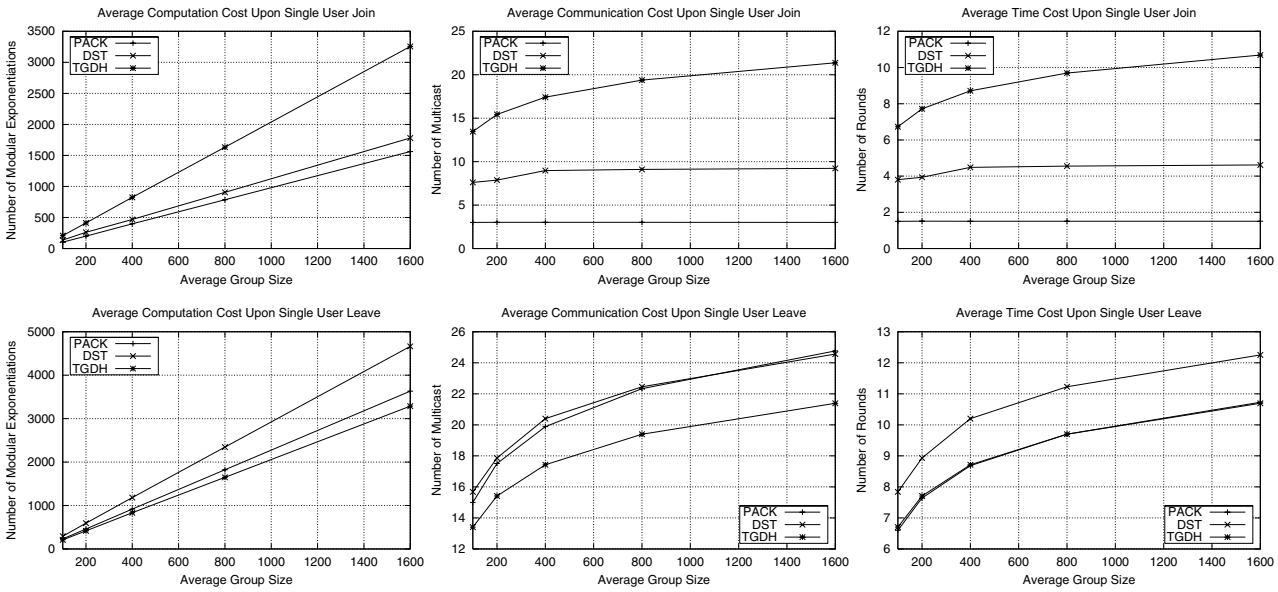

Fig. 1. Comparison of rekeying cost among PACK, TGDH and DST

In our simulations we generate the user activities according to the following probabilistic models: users join the group according to a Poisson process with average arrival rate λ, and users' staying time in the group follows an exponential distribution with mean μ. Then $\lambda\mu$ is the average number of users in the group, that is, the average group size. For each simulation, we initialize the group size to be 0, fix λ, and vary μ to get different average group size configuration. For each configuration (different average group size), a sequence of $100\lambda\mu$ users join the group according to the Poisson process with rate λ, and each user's staying time is drawn independently from an exponential distribution with mean μ. In the simulations, we have compared the rekeying cost of the following three schemes: PACK, TGDH [7] and DST [8], in all three aspects: computation, communication and time.

The simulation results are presented in Fig. 1. From these results we can see that upon single user join event, PACK has the lowest cost among all three schemes. Compared with DST, PACK has more than 10% reduction in computation cost, more than 65% reduction in communication cost and time cost. Compared with TGDH, the reduction is even more, about 50% in computation cost and about 80% in time and communication cost. Upon single user leave event, compared with DST, PACK has about 25% reduction in computation cost, about 15% reduction in time cost, and has similar communication cost. Although PACK has slightly higher computation and communication cost than TGDH upon single user leave event, when averaged over both join and leave events, the reduction is still significant, with 20% reduction in computation cost, 35% reduction in communication cost, and 40% reduction in time cost.

VI. Conclusion

In this paper, we proposed PACK, a highly efficient contributory key agreement scheme to achieve access control in collaborative and dynamic group applications. Upon single user join, PACK has the time cost as 1 or 2 rounds of two-group DH, the communication cost as 2 or 4 multicast, and the average computation cost as 1 modular exponentiation per user. Upon single user leave event, PACK takes at most $\log n$ rounds of two-group DH in terms of time cost, $O(\log n)$ multicast in communication cost, and an average of 2 modular exponentiations per user in computation cost, where n is the current group size. Both theoretical bound analysis and simulation results have shown that PACK has much lower rekeying cost in terms of communication, computation and cost than the existing schemes.

References

[1] I. Ingemarsson, D. T. Tang, and C. K. Wong, "A conference key distribution system," *IEEE Transactions on Information Theory*, vol. IT-28, no. 5, pp. 714–720, Sep. 1982.

[2] D. G. Steer, L. Strawczynski, W. Diffie, and M. Wiener, "A secure audio teleconference system," in *Proceedings on Advances in cryptology*, 1990, pp. 520–528.

[3] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution scheme," *Advances in Cryptology-Eurocrypt*, pp. 275–286, 1994.

[4] M. Steiner, G. Tsudik, and M. Waidner, "Diffie-Hellman key distribution extended to group communication," in *ACM Conference on Computer and Communication Security*, 1996, pp. 31–37.

[5] K. Becker and U. Wille, "Communication complexity of group key distribution," in *ACM Conference on Computer and Communication Security*, 1998, pp. 1–6.

[6] M. Steiner, G. Tsudik, , and M. Waidner, "Key agreement in dynamic peer groups," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 8, pp. 769–780, Aug 2000.

[7] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based group key agreement," *ACM Transactions on Information and System Security*, vol. 7, no. 1, pp. 60–96, Feb. 2004.

[8] Y. Mao, Y. Sun, M. Wu, and K. J. R. Liu, "Dynamic join-exit amortization and scheduling for time-efficient group key agreement," in *IEEE INFOCOM*, 2004.

[9] Jack Snoeyink, Subhash Suri, and George Varghese, "A lower bound for multicast key distribution," in *IEEE INFOCOM*, 2001.

[10] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, Nov. 1976.