A Distributed Key Management Protocol for Dynamic Groups

Jen-Chiun Lin* Chun-Yen Chou† Feipei Lai‡ Kuen-Pin Wu§

Abstract

With the popularity of group-oriented applications, secure and efficient communication among all group members has become a major issue. In this paper, we propose a distributed key management protocol for group members to share a secret group key, which can be used to protect shared information. The protocol manages the group key of a dynamic group, where members can freely join or leave, and each time the key is updated using two broadcast messages. The protocol algorithms provide group key secrecy, forward group key secrecy, and backward group key secrecy. The complexities of the group key computation time, the storage space for every member, and the total communication bandwidth to update the group key are approximately of logarithmic order of the group size, which make the protocol attractive for environments with less computation power and smaller storage.

1 Introduction

Information confidentiality is an important issue in communication. With the popularity of group-oriented applications, such as teleconferencing, chat rooms, and collaborative groupware, to name a few, there are increasing demands for secure and efficient information sharing among group members. Though numerous schemes have been developed for two communicating entities, the solutions for a group of more than two entities, in particular, key management, remain research challenges.

When concerning the information confidentiality problem for groups, rather than directly applying point-to-point solution to each pair of the group members, it is often more desirable to reduce the performance overhead by sharing keys, or by partitioning the groups into smaller subgroups. In both cases, key management plays a crucial role. Current solutions can be classified into two categories: the centralized approach [16, 10] where keys are managed by servers, and the distributed approach [2, 4, 7] where keys are distributed among all members. While both approaches have their own advantages and disadvantages, the centralized approach often suffers from the fact that servers need great computation power, large communication bandwidth, and considerable storage space. These servers tend to be the performance bottleneck of the entire system. If the servers are not group members, the entire system needs additional resources for them. If the servers themselves are also group members, then the imbalance of server members' duty and non-server members' duty might cause problems in fairness. The distributed approach is attractive if the above issues are major concerns.

For the above reasons, we focus on providing a distributed group key management solution. In this paper, we propose a secure, efficient and distributed key management protocol for dynamic groups using broadcast network. Our goal is that all group members can securely and efficiently share a common group key, which can be used to protect shared information. The protocol supports operations for dynamic groups, including group key updating, member joining, and member leaving. The protocol algorithms are based on the extension of two-party Diffie-Hellman key exchange (2DH). Our solution uses a variation of key trees to speed up key computation. Each group member keeps only partial group information to run the protocol, which makes the communication bandwidth and storage requirement low.

The rest of the paper is organized as follows. Section 2 discusses the general requirements and issues in secure group communication systems. Section 3 introduces related work of this field. Section 4 gives the protocol terminology and algorithms. Section 5 discusses the proto-

*Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan, *simon@orchid.ee.ntu.edu.tw*.

†Department of Mathematical Education, National Hualien Teachers College, Hualien 970, Taiwan, *choucy@sparc2.Nhltc.edu.tw*.

‡Department of Electrical Engineering & Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan, *flai@cc.ee.ntu.edu.tw*.

§Institute of Information Science, Academia Sinica, Taipei, Taiwan, *kpw@iis.sinica.edu.tw*.

Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN'02)
0742-1303/02 $17.00 © 2002 IEEE

col security, complexity and other issues. The last section gives conclusions and explores possible future research directions.

2 Overview

The protocol uses *atomic* and *reliable* network broadcasts to send request messages and updating messages. Here "a broadcast service is atomic and reliable" means that either a broadcast message is successfully received by all entities or the operation fails. The reason using broadcasts is that, most of the time, one group member, or a new joining member, does not know all other group members, but it still has to send messages to all group members without relying on servers.

The protocol supports three kinds of operations: group key updating operations, member joining operations, and member leaving operations. Two broadcast messages, a request message and an updating message, are required to perform any one of such operations. A request message is broadcast for a member to initiate a group operation. An updating message is then broadcast to all members to compute the new group key.

Since the order of these operations are crucial to the consistency of the group, every group member must receive the request messages in the same order. The protocol itself does not provide any ordering mechanism, and it depends on the underlying network support for total ordered message delivery. More rigorous discussion of the communication models can be found in [11].

All messages of the protocol operations should be signed and authenticated. We use signatures for two reasons. First, signatures provides mechanisms for source authentication, that is, group members receiving the message can verify if the message really comes from the claimed group member. This ensures that the messages cannot be faked by malicious entities. Another reason is that message integrity can be guaranteed by checking the signatures, which is also important that the message information is not altered during transmission. There are several public key algorithms, such as RSA, that can produce strong digital signatures. There are also popular authentication systems based on signatures, including Pretty Good Privacy (PGP) [17] and Public Key Infrastructure (PKI) [13]. For more general discussions of signatures and authentication, please refer to [12].

Conceptually, each group member uses two threads to run the protocol. One thread, the *command thread*, waits for client commands and uses appropriate algorithms to send request messages. Another thread, the *protocol thread*, receives request messages and uses appropriate algorithms to handle them. While handling the request, one of the group members broadcasts an updating message, and all group members, including the updating message sender, receive the message and update their member information. The pseudo-code of these two threads is listed in Figure 1. The algorithms used by these threads are discussed in section 4.

```
command_thread
   while the protocol works correctly do
      accept a command from the client
      select a protocol algorithm to send request message

protocol_thread
   while the protocol works correctly do
      receive a request message
      select a protocol algorithm to handle the message
```

Figure 1. Pseudo-code of the threads

3 Related Work

The distributed schemes are often based on the two-party Diffie-Hellman key exchange (2DH) or its extensions. In [14], Steer et al. proposed a key agreement protocol STR for secure audio conferencing. The solution is not suitable for large groups because its computation and message size grow linearly with the group size. Burnmester and Desmedt [6] present several conference key distribution systems. Among them, the tree-based system and the broadcast system are particularly interesting. All N members in the tree based system agree upon a group key in $O(\log N)$ rounds, and in each round, two members perform the 2DH protocol to compute a common key for the next round. In the broadcast system, only two rounds are needed to reach the agreement, but each round needs N broadcast messages. Steiner, Tsudik, Ateniese, and Waidner have a series of work about the dynamic peer groups key agreement system, the CLIQUE protocol suite [15, 1, 3, 2], based on the group Diffie-Hellman (GDH) protocol. However, the solution aims at small dynamic peer groups, and it does not scale well when the group size becomes large.

Becker and Wille [4] study contributory key distribution systems based on the 2DH key exchange protocol, with or without broadcasts. They give theoretical lower bounds of the total number of messages, the total number of exchanges, and the total number of simple rounds. In addition, they proposed the 2^d-octopus and the 2^d-cube protocols. While these protocols can deal with group key agreements efficiently in terms of the number of messages or

rounds, the member leaving operation is not handled well according to [7]. Kim et al. [7] propose a tree based group Diffie-Hellman protocol (TGDH). One potential problem is the broadcast bandwidth required, since it is proportional to the group size. Each member must store the whole group tree, which requires considerable storage.

4 The Protocol Algorithms

A group key management protocol must update the group key according to group key updating requests or group membership changes. The protocol supports two kinds of group membership operations, the joining operations and the leaving operations. A new member initiates the joining operation to be added to the group. A member leaves the group after it has successfully notified all group members. The protocol uses an extended Diffie-Hellman key agreement algorithm based on two-party Diffie-Hellman key exchange [8, 9]. Every group member keeps partial group information to compute the group key and maintain membership consistency. The data structures and algorithms of the protocol will be explained in the rest of the section. The notations used throughout this paper are listed in Table 1.

Table 1. Notation

p	a large prime number
$GF(p)$	the finite field of order p
g	a generator of $GF(p)^*$
m	a group member
x	a secret key
k	a key computed by the 2DH protocol
h	an agreement key used in the 2DH protocol
$\varphi(x)$	$g^x \bmod p$
V	the ordered tuples of agreement keys
v	an internal node
G	the group
I	a unique identity in G
W	a weight factor
D	node information $(I_s, W_s, k_s, I_t, W_t, h_t)$
E	node updating information $(I_s, W_s, h_s, I_t, W_t, h_t)$
P	path information, a sequence of Ds
U	path updating information, a sequence of Es
M	member information (I, x, k_G, P)
L	the length of the specified path
i, j	indices

4.1 Basic Concepts

The key management mechanism is based on a variation of key trees. A leaf node of the key tree represents a group member or the secret key it holds, and an internal node represents the information shared by members whose paths to the root contain the node. Figure 2 shows a sample key tree.

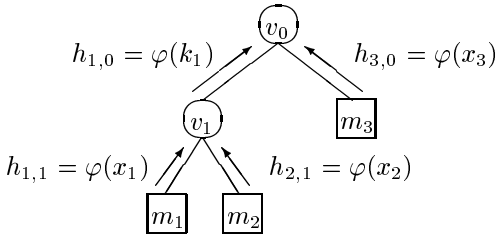

Figure 2. A key tree

In Figure 2, assume that there are three members in the group, m_1, m_2 and m_3, and each m_i holds a secret key x_i. The internal nodes are labeled v_0, v_1. First, m_1 and m_2 perform a two-party Diffie-Hellman key exchange (2DH) and produce a shared key $k_1 = \varphi(x_1 x_2)$, which can be treated as the secret key of v_1. Then v_1 and m_3 perform a 2DH and produce a shared key

$$k_0 = \varphi(k_1 x_3) = g^{g^{x_1 x_2} \cdot x_3} \bmod p.$$

The key k_0 can be viewed as the group key shared by m_1, m_2 and m_3. The sample implies that if all group members can be arranged into leaf nodes of a binary key tree, the group key can be easily computed by successive applications of 2DH.

Without centralized servers, it is not suitable for every group member to keep a copy of the key tree, since the cost is considerably high considering the space used to store it and the bandwidth used to transmit it. Instead, every group member just keeps the information of those nodes on the path from the root node of the key tree to the leaf node representing itself. Every group member keeps partial group information $M \equiv (I, x, k_G, P)$, where I is its identity, x is its secret key, k_G is the group key, and P is the path information as illustrated in Figure 3. Each node information is a six-tuple $D \equiv (I_s, W_s, k_s, I_t, W_t, h_t)$. The values k_s, h_t are used to compute the group key, and I_s, W_s, I_t, W_t are used for group membership management. These nodes are indexed from the root node to the leaf node, beginning from 0. The subscripts s and t mean *self* and *peer* respectively. The concept of *self* and *peer* can be explained using

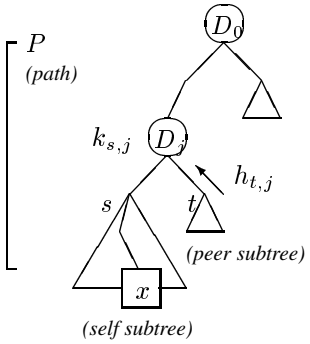

Figure 3. Path notation

Figure 2. From m_1's point of view, the *self* and *peer* subtrees of v_1 are m_1 and m_2 respectively; from m_2's point of view, they are m_2 and m_1 respectively. The weight factor W_s and W_t are the number of group members in the *self* and *peer* subtrees respectively. These data are used to keep the average path length of all group members small, so that the key tree constructed by these paths is more balanced. Every node is associated with two identity I_s and I_t, each denotes the identity of the representative group member in the two subtrees labeled as "self" and "peer". The protocol uses these identities to identify subtrees.

To compute the group key, we observe that, for a node, the value k_s can be viewed as its secret key or a key shared by the members in its *self* and *peer* subtrees, and the 2DH agreement key h_t is $\varphi(k_t) = g^{k_t} \pmod p$, which can be used to further compute a shared key $\varphi(k_s k_t) = g^{k_s k_t} \pmod p = (g^{k_t})^{k_s} \pmod p = h_t^{k_s} \pmod p$. In general, given a path with L nodes, we can compute $k_{s,0}$, the group key, using the following recurrence relations:

$$\begin{aligned} h_{t,j} &= \varphi(k_{t,j+1}) = g^{k_{t,j+1}} \bmod p, \\ k_{s,j} &= \varphi(k_{s,j+1} k_{t,j+1}) = h_{t,j}^{k_{s,j+1}} \bmod p, \end{aligned}$$

where $0 \le j \le L-1$. The algorithm in Figure 4 computes the group key and updates the weight factors along the path.

algorithm ComputeGroupKey()
  $L \leftarrow |P|$; $k \leftarrow x$
  for $j = L-1$ **down to** 0 **do**
    $k_{s,j} \leftarrow h_{t,j}^{k} \bmod p$
    $k \leftarrow k_{s,j}$
    if $j \neq L-1$ **then** $W_{s,j} \leftarrow W_{s,j+1} + W_{t,j+1}$
  $k_G \leftarrow k$

Figure 4. ComputeGroupKey algorithm

Every time a new member joins the group or a member leaves the group, all members' path information (which containing some 2DH agreement keys) must be updated to reflect group membership changes. To update those information, the path updating information U is broadcast. U is similar to the path information, except that it is composed of a sequence of node updating information E, which is a six-tuple $(I_s, W_s, h_s, I_t, W_t, h_t)$. The only difference between E and D is h_s, which stores the 2DH agreement key computed by the message sender. The algorithms manipulating U and P are shown in Figure 5.

algorithm CreateU()
  $U' \leftarrow P$; $L \leftarrow |P|$; $k \leftarrow x$
  for $j = L-1$ **down to** 0 **do**
    $h'_{s,j} \leftarrow g^{k} \bmod p$
    $k \leftarrow h'^{\,k}_{t,j} \bmod p$
    if $j \neq L-1$ **then** $W'_{s,j} \leftarrow W'_{s,j+1} + W'_{t,j+1}$
  return U'

algorithm UpdatePath($\tilde{U}$)
  $L \leftarrow |P|$
  for $j = 0$ **to** $L-1$ **do**
    if $(I_{s,j} = \tilde{I}_{s,j})$ **and** $(I_{t,j} = \tilde{I}_{t,j})$ **then**
      $W_{s,j} \leftarrow \tilde{W}_{s,j}$; $W_{t,j} \leftarrow \tilde{W}_{t,j}$
    else if $(I_{s,j} = \tilde{I}_{t,j})$ **and** $(I_{t,j} = \tilde{I}_{s,j})$ **then**
      $W_{s,j} \leftarrow \tilde{W}_{t,j}$; $W_{t,j} \leftarrow \tilde{W}_{s,j}$; $h_{t,j} \leftarrow \tilde{h}_{s,j}$
      break

Figure 5. Algorithms manipulating U and P

Algorithm *CreateU* is used to create updating information U from a member's path P and its secret key x. Every group member uses algorithm *UpdatePath* to update its own path according to the path updating information. It can be observed that the agreement keys are computed in algorithm *CreateU*, and every member runs algorithm *UpdatePath* to replace agreement keys in P by corresponding agreement keys in $\tilde{U}$. If the agreement keys in a path are changed, the computed group key is also changed.

4.2 The Group Key Updating Algorithms

The group key updating algorithms are used when a group member wishes to update the group key. A group member uses algorithm *SendRekeyRequest* to initiate a group key updating operation. All group members, including the sender, then proceed to finish the operation using algorithm *HandleRekeyRequest*. In the algorithm, the sender selects a new secret key, and broadcasts the path updating information U. All group members modify their path information and compute the new group key. The algorithms are listed in Figure 6.

Take the key tree in Figure 2 as an example. Assume that the identity of each member m_i is I_i, and the representative

```
algorithm SendRekeyRequest()
  broadcast REKEY_REQUEST[I]

algorithm HandleRekeyRequest(Ĩ)
  if I = Ĩ then
    x ← randomly select a new secret key
    U ← CreateU()
    broadcast REKEY_UPDATE[U]
  receive REKEY_UPDATE[Ũ]
  UpdatePath(Ũ)
  ComputeGroupKey()
```

Figure 6. The group key updating algorithms

member of m_1, m_2 is m_1 (at node v_1). Let P_i be the path information of m_i, then we have:

$$\begin{aligned} P_1 &= ((I_1, 2, \varphi(x_3\varphi(x_1x_2)), I_3, 1, \varphi(x_3)), \\ &\quad (I_1, 1, \varphi(x_1x_2), I_2, 1, \varphi(x_2))), \\ P_2 &= ((I_1, 2, \varphi(x_3\varphi(x_1x_2)), I_3, 1, \varphi(x_3)), \\ &\quad (I_2, 1, \varphi(x_1x_2), I_1, 1, \varphi(x_1))), \\ P_3 &= ((I_3, 1, \varphi(x_3\varphi(x_1x_2)), I_1, 2, \varphi(x_1x_2))). \end{aligned}$$

Assume that m_1 broadcasts a REKEY_REQUEST message. After receiving the request message, m_1 itself selects a new secret key x_1' and broadcasts the REKEY_UPDATE message including

$$\begin{aligned} U &= ((I_1, 2, \varphi(\varphi(x_1'x_2)), I_3, 1, \varphi(x_3)), \\ &\quad (I_1, 1, \varphi(x_1'), I_2, 1, \varphi(x_2))). \end{aligned}$$

Then all group members update their path information, and the new path information are:

$$\begin{aligned} P_1' &= ((I_1, 2, \varphi(\varphi(x_1'x_2)x_3), I_3, 1, \varphi(x_3)), \\ &\quad (I_1, 1, \varphi(x_1'x_2), I_2, 1, \varphi(x_2))), \\ P_2' &= ((I_1, 2, \varphi(\varphi(x_1'x_2)x_3), I_3, 1, \varphi(x_3)), \\ &\quad (I_2, 1, \varphi(x_1'x_2), I_1, 1, \varphi(x_1'))), \\ P_3' &= ((I_3, 1, \varphi(\varphi(x_1'x_2)x_3), I_1, 2, \varphi(\varphi(x_1'x_2))). \end{aligned}$$

4.3 The Member Joining Algorithms

The member joining algorithms are used when a new member wishes to join the group. To become a group member, one uses algorithm *SendJoinRequest* to select a secret key and broadcast the request message. The JOIN_REQUEST message includes an additional field, the agreement key h, which will be used by the group members to compute the new group key. All group members will receive the request message, and they use algorithm *HandleJoinRequest* to handle it. In algorithm *HandleJoinRequest*, one of the group members will find itself to be responsible for the joining operation. It selects a new secret key, computes the path updating information U, and then broadcast it to all group members including the new member. Group members use U to update their path information, and the new member uses algorithm *CreatePath* to create its path information from U. Now the new member becomes a group member, and it can compute the group key with its path and secret key. The member joining algorithms are listed in Figure 7.

```
algorithm SendJoinRequest()
  x ← randomly select a new secret key
  h ← g^x mod p
  broadcast JOIN_REQUEST[I, h]

algorithm HandleJoinRequest(Ĩ, h̃)
  if I ≠ Ĩ and IsLeader() then
    x ← randomly select a new secret key
    D ← (I, 1, 0, Ĩ, 1, h̃)
    append D to P
    U ← CreateU()
    broadcast JOIN_UPDATE[U]
  receive JOIN_UPDATE[Ũ]
  if I = Ĩ then CreatePath(Ũ)
  else UpdatePath(Ũ)
  ComputeGroupKey()

algorithm IsLeader()
  L ← |P|
  for j = 0 to L − 1 do
    if W_{s,j} > W_{t,j} then return false
    if W_{s,j} = W_{t,j} and I_{s,j} < I_{t,j} then return false
  return true

algorithm CreatePath(Ũ)
  P ← Ũ; L ← |P|
  I_{s,L−1} ← Ĩ_{t,L−1}; I_{t,L−1} ← Ĩ_{s,L−1}; h_{t,L−1} ← h̃_{s,L−1}
```

Figure 7. The member joining algorithms

As can be seen in algorithm *IsLeader*, only one of the group members will get a positive result after evaluating their paths. The member is responsible to make the JOIN_UPDATE message and to broadcast it. The algorithm *IsLeader* examines the path nodes from the root node to the leaf node. For those members who are in a subtree with greater weight factor, they are not considered as candidates of the temporary leader. The idea is that, to place a new member as a leaf node in a key tree, the node should be placed in a subtree with less leaf nodes to make the tree more balanced. As shown in algorithm *HandleJoinRequest*, it is simple for the temporary group leader to change its path to reflect the addition of a new member. It only has to

append a new node to its path, and to update the path information henceforth. These two members are peers of each other afterwards. Figure 8 illustrates the concept.

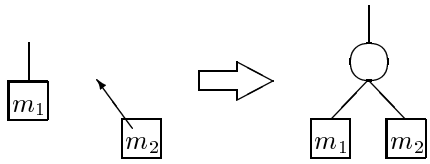

Figure 8. Handling the joining member

To demonstrate the algorithms, we assume that initially there are two members m_1,m_3 in the group with $I_1 < I_3$, and a new member m_4 wishes to join the group. Let the original path information of the group be:

$$\begin{aligned} P_1 &= ((I_1, 1, \varphi(x_1 x_3), I_3, 1, \varphi(x_3))), \\ P_3 &= ((I_3, 1, \varphi(x_1 x_3), I_1, 1, \varphi(x_1))). \end{aligned}$$

Now the new member m_4 is ready to join the group. Because the weight factors of both child subtrees are the same, and $I_3 > I_1$, m_3 will be the temporary leader. Member m_3 selects a new secret key x_3' and broadcasts the path updating information

$$\begin{aligned} U &= ((I_3, 2, \varphi(\varphi(x_3' x_4)), I_1, 1, \varphi(x_1)), \\ &\quad (I_3, 1, \varphi(x_3'), I_4, 1, \varphi(x_4))). \end{aligned}$$

The final path information would be:

$$\begin{aligned} P_1' &= ((I_1, 1, \varphi(x_1 \varphi(x_3' x_4)), I_3, 2, \varphi(\varphi(x_3' x_4)))), \\ P_3' &= ((I_3, 2, \varphi(x_1 \varphi(x_3' x_4)), I_1, 1, \varphi(x_1)), \\ &\quad (I_3, 1, \varphi(x_3' x_4), I_4, 1, \varphi(x_4))), \\ P_4' &= ((I_3, 2, \varphi(x_1 \varphi(x_3' x_4)), I_1, 1, \varphi(x_1)), \\ &\quad (I_4, 1, \varphi(x_3' x_4), I_3, 1, \varphi(x_3))). \end{aligned}$$

If another new member m_2 wishes to join the group, this time, m_1 will be the temporary leader since $W_{1,0} < W_{3,0}$. It is easily verified that the resulting path length of every member is 2.

4.4 The Member Leaving Algorithms

The member leaving algorithms are used when a group member wishes to leave the group. The leaving member uses algorithm *SendLeaveRequest* to notify all group members that it is going to leave. The algorithms require the leaving member to remain in the group until the request of leaving is successfully handled. It is because that the protocol relies on the underlying network system to order requests from different members. The leaving member cannot just broadcast the request and quit, since it may have to handle other requests broadcast prior to its leaving request, but not received yet. The peer member of the leaving one is responsible to select a new secret key, to make the path updating information U, and to broadcast it. Algorithm *HandleLeaveRequest* uses algorithm *ReplaceMember* to replace the leaving member with its peer member. These algorithms are listed in Figure 9.

```
algorithm SendLeaveRequest()
  broadcast LEAVE_REQUEST[I]

algorithm HandleLeaveRequest(Ĩ)
  if I = Ĩ then leave the group
  if IsPeer(Ĩ) then
    x ← randomly select a new secret key
    ReplaceMember(Ĩ, I)
    U ← CreateU()
    broadcast LEAVE_UPDATE[U]
  receive LEAVE_UPDATE[Ũ]
  L ← |Ũ|; I' ← Ĩ_{t,L-1}
  ReplaceMember(Ĩ, I')
  UpdatePath(Ũ)
  ComputeGroupKey()

algorithm IsPeer(Ĩ)
  L ← |P|
  for j = 0 to L − 1 do
    if I_{s,j} = I and I_{t,j} = Ĩ and W_{t,j} = 1 then return true
  return false

algorithm ReplaceMember(Ĩ, I')
  L ← |P|
  for j = 0 to L − 1 do
    if I_{s,j} = Ĩ then I_{s,j} ← I'
    else if I_{t,j} = Ĩ then
      if W_{t,j} = 1 then remove D_j from P
      else I_{t,j} ← I'
```

Figure 9. The member leaving algorithms

Conceptually, if a group member leaves the group, the leaf node representing the member must be removed from the key tree. As illustrated in Figure 10, the new tree becomes (a) when the leaving member itself forms one subtree, or (b) when the leaving member is in one subtree with more than two members. If algorithm *ReplaceMember* finds that the peer subtree identity of a node is the same as the leaving member's identity and its weight is one, the situation belongs to case (a) and the node should be removed from the path. Otherwise, it just replaces the leaving member's identity with its peer member's identity.

Following the previous sample, we assume that m_3 wishes to leave. Member m_3 leaves after it receives the

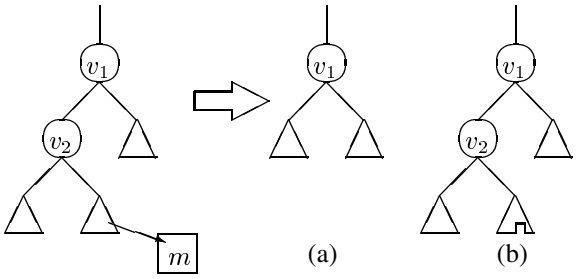

Figure 10. Handling the leaving member

LEAVE_REQUEST message it broadcast. All other members receive the message, and m_4 finds itself to be the peer member of m_3. Member m_4 then selects a new key x_4'', fixes its path using algorithm *ReplaceMember* to

$$P_4'' = ((I_4, 2, \varphi(x_1\varphi(x_3'x_4)), I_1, 1, \varphi(x_1)),$$

and broadcasts the path updating information

$$U'' = ((I_4, 1, \varphi(x_4''), I_1, 1, \varphi(x_1))).$$

Then both m_1 and m_4 receive the updating message, and only m_1's path is modified and

$$P_1'' = ((I_1, 1, \varphi(x_1\varphi(x_3'x_4)), I_4, 2, \varphi(\varphi(x_3'x_4)))).$$

Finally, m_1 and m_4 update their path information to

$$P_1''' = ((I_1, 1, \varphi(x_1x_4''), I_4, 1, \varphi(x_4''))),$$
$$P_4''' = ((I_4, 1, \varphi(x_1x_4''), I_1, 1, \varphi(x_1'))).$$

5 Discussion

We first give a security analysis by proving several theorems and propositions. Then we discuss the time, space complexities, and communication costs of the protocol.

5.1 Security Analysis

The underlying algorithm of our key management protocol is an extended Diffie-Hellman (EDH) key agreement algorithm, which is based on iterations of the two-party Diffie-Hellman key exchange (2DH). It is clear from section 4 that EDH is *contributory*, that is, each group member makes an independent contribution to the group key. The protocol also provides *key integrity* that the group key is a function of every member's secret key, and no other entities contribute to the computation of the group key. If EDH provides group key secrecy, forward group key secrecy and backward group secrecy, then we will show that our protocol is secure for dynamic groups.

Group key secrecy means that it is computationally infeasible for a passive adversary to compute the group key. Forward group key secrecy guarantees that it is computationally infeasible for passive adversaries to derive the new group key from old secret keys and old group keys. Backward group key secrecy guarantees that it is computationally infeasible for passive adversaries to derive old group keys from new secret keys and new group keys. These properties together ensure that a group member only has the knowledge of its own secret keys and the group keys computed when it is in the group.

The security of EDH is based on two more fundamental problems: the difficulty to the solve discrete logarithm problem (DL), and the 2DH problem. The DL problem is hard to solve in general, but if p is a composite of small primes, efficient algorithms, such as the Silver-Pohlig-Hellman algorithm [8], exist. The security of 2DH relies on the decisional Diffie-Hellman assumption (DDH), which states that given arbitrary a, b, c in $GF(p)$, and the tuples $(\varphi(a), \varphi(b), \varphi(ab))$ and $(\varphi(a), \varphi(b), \varphi(c))$ in random order, no polynomial time algorithms can determine which one is the 2DH tuple with probability substantially greater than $1/2$ [5, 9].

Since all information an adversary can get is in exponential form, to derive the exponents, and hence the secret keys of group members, is as hard as to solve the DL. It follows that every group member's secret key is hard to derive for any adversary. In order to prove that the protocol provides group key secrecy, forward group key secrecy and backward forward secrecy, we use an equivalent form of the EDH, which looks like the key computation algorithm in section 4.1 rewritten in recursive form.

algorithm EDHS(S, X_n)
  if $n = 1$ **then return** $((\varphi(x_1)), x_1)$
  else if $n = 2$ **then return** $((\varphi(x_1), \varphi(x_2)), \varphi(x_1x_2))$
  else
    partition X_n *to* X_i, X_j *using* S
    $(V_{S,i}, k_{S,i}) \leftarrow$ EDHS(S, X_i)
    $(V_{S,j}, k_{S,j}) \leftarrow$ EDHS(S, X_j)
    return $((V_{S,i}, V_{S,j}, \varphi(k_{S,i}), \varphi(k_{S,j})), \varphi(k_{S,i}k_{S,j}))$

Figure 11. EDH simulation algorithm (*EDHS*)

Figure 11 shows the simulation algorithm *EDHS* for EDH. *EDHS* has two inputs: $X_n \equiv (x_1, x_2, \ldots, x_n)$ is an n-tuple and each tuple is the secret key of a group member; S is a protocol dependent algorithm to partition the n-tuple X_n to an i-tuple X_i and a j-tuple X_j. It controls the output of the simulation algorithm. To simulate the EDH used in our protocols, S must be deterministic and independent

of the values of X, since all our algorithms are not probabilistic and run independently of the secret key values or computed key values. S is open to adversaries since the protocol is public. Note that the order of X_i and X_j is not important, since $\varphi(k_i k_j) = \varphi(k_j k_i)$. Switching X_i and X_j only affects the order of the tuples of V_n. There are two parts of the output:

- $k_{S,n}(X_n)$ simulates the key value computed by all group members;
- $V_{S,n}(X_n)$ is the ordered $2(n-1)$-tuple that simulates all values needed to be transmitted to compute $k_{S,n}(X_n)$.

For instance, in the example given in section 4.1, we have $n = 3$, and $X_3 = (x_1, x_2, x_3)$. The algorithm S will partition X_3 to two ordered tuples, (x_1, x_2) and (x_3). The outputs of $EDHS(S, X_3)$ are:

$$\begin{aligned} k_{S,3}(X_3) &= \varphi(\varphi(x_1 x_2) x_3), \\ V_{S,3}(X_3) &= (\varphi(x_1), \varphi(x_2), \varphi(\varphi(x_1 x_2)), \varphi(x_3)). \end{aligned}$$

We follow a similar approach used in [4]. To prove that the EDH provides group key secrecy, we only need to show that the key generated by *EDHS* is polynomial time indistinguishable from a random number of $GF(p)$, with all the values transmitted by the protocols are known. For a random value $y \in GF(p)$, consider

$$\begin{aligned} A_{S,n}(X_n) &= (V_{S,n}(X_n), y), \\ F_{S,n}(X_n) &= (V_{S,n}(X_n), k_{S,n}(X_n)). \end{aligned}$$

We write $A_n \approx_{poly} F_n$ to mean that A and F are polynomial time indistinguishable, no matter what protocol dependent algorithm S is.

Theorem 1 *The computational infeasibility to solve DDH implies $A_n \approx_{poly} F_n$ for all $n \geq 2$.*

Proof: We prove it by an alternative form of induction. Given $n = 2$, x_1, x_2 belonging to $GF(p)$, and the protocol dependent algorithm S, we have $A_{S,2}(X_2) = (V_{S,2}(X_2), y) = (\varphi(x_1), \varphi(x_2), y)$, and $F_{S,2}(X_2) = (V_{S,2}(X_2), k_{S,2}(X_2))) = (\varphi(x_1), \varphi(x_2), \varphi(x_1 x_2))$. In algorithm *EDHS*, S is ignored for $n = 2$, so the assumption that it is computationally infeasible to solve DDH implies that A_2 and F_2 are polynomial time indistinguishable. Assume that for all $n = 2, 3, \ldots, m-1$, $m > 2$, $A_{m-1} \approx_{poly} F_{m-1}$ is true. Now we only have to show that for $n = m$, $m > 2$, $A_m \approx_{poly} F_m$ to complete the proof. In order to do so, we rewrite A_m, F_m and define several ordered tuples:

$$\begin{aligned} A_{S,m}(X_m) &= (V_{S,i}(X_i), V_{S,j}(X_j), \\ &\quad \varphi(k_{S,i}(X_i)), \varphi(k_{S,j}(X_j)), y), \\ B_{S,m}(X_m) &= (V_{S,i}(X_i), V_{S,j}(X_j), \\ &\quad \varphi(a), \varphi(k_{S,j}(X_j, S)), y), \\ C_{S,m}(X_m) &= (V_{S,i}(X_i), V_{S,j}(X_j), \varphi(a), \varphi(b), y), \\ D_{S,m}(X_m) &= (V_{S,i}(X_i), V_{S,j}(X_j), \varphi(a), \varphi(b), \varphi(a \cdot b)), \\ E_{S,m}(X_m) &= (V_{S,i}(X_i), V_{S,j}(X_j), \varphi(a), \\ &\quad \varphi(k_{S,j}(X_j)), \varphi(a \cdot k_{S,j}(X_j))), \\ F_{S,m}(X_m) &= (V_{S,i}(X_i), V_{S,j}(X_j), \varphi(k_{S,i}(X_i)), \\ &\quad \varphi(k_{S,j}(X_j)), \varphi(k_{S,i}(X_i) \cdot k_{S,j}(X_j))), \end{aligned}$$

where $a \neq k_{S,i}(X_i)$ and $b \neq k_{S,j}(X_j)$ are random values chosen from $GF(p)$. Since $A_m \approx_{poly} B_m$, $B_m \approx_{poly} C_m$, $C_m \approx_{poly} D_m$, $D_m \approx_{poly} E_m$, $E_m \approx_{poly} F_m$ together imply $A_m \approx_{poly} F_m$, it only has to be shown that each one of them is true to complete the proof.

Proposition 1: $A_m \approx_{poly} B_m$, for all $m > 2$.
Assume that for some protocol algorithm S, there exists a polynomial time algorithm M that distinguishes between $A_{S,m}$ and $B_{S,m}$. For some instance $\chi = (V_{S,i}(X_i), V_{S,j}(X_j), \varphi(z), \varphi(k_{S,j}(X_j)), y)$, M distinguishes whether χ belongs to $A_{S,m}(X_m)$ or $B_{S,m}(X_m)$. The algorithm *EDHS* uses S to partition X_m to two tuples X_i and X_j when $m > 2$, and we choose i and j such that $i \geq j$ (recall that the order of X_i and X_j is not important). Now we want to determine whether $\lambda = (V_{S,i}(X_i), z)$ belongs to $A_{S,i}(X_i)$ or $F_{S,i}(X_i)$. We choose random value y', and j random values from $GF(p)$ to make X'_j. Run $EDHS(S, X'_j)$ to generate $V'_{S,j}(X'_j)$, and the value $k'_{S,j}(X'_j)$. The new instance $\chi' = (V_{S,i}(X_i), V'_{S,j}(X'_j), \varphi(z), \varphi(k'_{S,j}(X_j)), y')$ is fed to M. If M tells us that χ' belongs to $A_{S,m}$, then λ belongs to $F_{S,i}$ (since in this case $z = k_{S,i}(X_i)$). If M tells us that χ' belongs to $B_{S,m}$, then λ belongs to $A_{S,i}$ (since in this case $z = a$). It is shown that M is a polynomial time distinguishing algorithm for $A_{S,i}$ and $F_{S,i}$, where $i \geq j \geq 1$ and $i + j = m > 2$. It contradicts $A_i \approx_{poly} F_i$, for $2 \leq i \leq m-1$, so we conclude that $A_m \approx_{poly} B_m$, for all $m > 2$.

Proposition 2: $C_m \approx_{poly} D_m$, for all $m > 2$.
Since $X_{S,m}$, $V_{S,m}$ are independent of a, b, y, these values are redundant and can be ignored, the problem can be directly reduced to DDH, which implies $C_m \approx_{poly} D_m$.

Proposition 3: $B_m \approx_{poly} C_m$, $D_m \approx_{poly} E_m$ and $E_m \approx_{poly} F_m$, for all $m > 2$.
It can be proved analogously to proposition 1, with suitable choices of χ and λ.

Theorem 1 confirms us that the EDH used by our algorithms provides group key secrecy. For more powerful adversaries, such as the left group members or current valid members, we have to show that EDH provides both forward group key secrecy and backward group key secrecy. We use $\bar{X}_n$ to denote the n-tuple of secret keys known by the adversary, and $F_{S,i,n}(\bar{X}_i, V_{S,n}(X_n))$ is an ordered tuple of values computable from $\bar{X}_i$ and $V_{S,n}(X_n)$ in polynomial time. We further define the ordered tuples:

$$\begin{aligned} H_{S,i,n}(\bar{X}_i, X_n) &= (\bar{X}_i, V_{S,n}(X_n), \\ &\quad F_{S,i,n}(\bar{X}_i, V_{S,n}(X_n)), y), \\ K_{S,i,n}(\bar{X}_i, X_n) &= (\bar{X}_i, V_{S,n}(X_n), \\ &\quad F_{S,i,n}(\bar{X}_i, V_{S,n}(X_n)), k_{S,n}(X_n)), \end{aligned}$$

where y is chosen from $GF(p)$ randomly.

Theorem 2 *$A_n \approx_{poly} F_n$ implies $H_{i,n} \approx_{poly} K_{i,n}$, for all $n \geq 2$ and $i > 0$.*

Proof: Suppose for some protocol algorithm S, there exists a polynomial time algorithm M distinguishing between $H_{S,i,n}$ and $K_{S,i,n}$. For the instance $\lambda = (V_{S,n}(X_n), z)$, we select i random values from $GF(p)$ to make the tuples $\bar{X}'_i$ and compute $F'_{S,i,n}(\bar{X}'_i, V_{S,n}(X_n))$. The instance $\chi' = (\bar{X}'_i, V_{S,n}(X_n), F'_{S,i,n}(\bar{X}'_i, V_{S,n}(X_n)), z)$ is fed to M. If M tells us that χ' belongs to $H_{S,i,n}$, λ belongs to $A_{S,n}$ (since in this case $z = y$). If M tells us that χ' belongs to $K_{S,i,n}$, λ belongs to $F_{S,n}$ (since in this case $z = k_{S,n}(X_n)$). Using the procedure, M is able to distinguish between $A_{S,n}$ and $F_{S,n}$ for protocol algorithm S, which contradicts theorem 1. So there is no such M for any protocol algorithm S, and $H_{i,n} \approx_{poly} K_{i,n}$.

From theorem 2, knowing $\bar{X}_i$, $V_{S,n}$ and algorithm S, the adversary can compute more information in polynomial time, but the information is useless against k_n. It follows that the EDH used by our algorithms provides forward and backward group secrecy. Theorem 2 tells us that it is infeasible to break those group keys without corresponding secret keys, and it implies that the protocol resists collusion attacks of several adversaries.

5.2 Complexity Analysis

The computation complexity of the algorithms is measured in terms of loop operations, and all these loops are single loops and the iteration counts depend on every member's path length. The storage complexity is measured by the space required for every member to keep the group information M. For every member, it only depends on the path information length, since all other information takes constant space. The communication complexity is measured by the number of messages, and the total bandwidth required. All kinds of request messages use only constant bandwidth, and all kinds of updating messages contain path updating information U, whose size is proportional to the message sender's path length. The group operations costs are summarized in Table 2, and they are compared to GDH.2 [15] and TGDH [7].

Table 2. Costs of group operations

The group key updating operation			
protocol	GDH.2	TGDH	our scheme
computation	$O(N^2)$	$O(\log N)$	$O(L)$
unicasts	$N-1$	0	0
broadcasts	1	1	2
total bandwidth	$O(N^2)$	$O(N)$	$O(L)$

(a)

The member joining operation			
protocol	GDH.2	TGDH	our scheme
computation	$O(N)$	$O(\log N)$	$O(L)$
unicasts	1	0	0
broadcasts	1	2	2
total bandwidth	$O(N)$	$O(N)$	$O(L)$

(b)

The member leaving operation			
protocol	GDH.2	TGDH	our scheme
computation	$O(N)$	$O(\log N)$	$O(L)$
unicasts	0	0	0
broadcasts	1	1	2
total bandwidth	$O(N)$	$O(N)$	$O(L)$

(c)

The member storage requirement			
protocol	GDH.2	TGDH	our scheme
storage	$O(N)$	$O(N)$	$O(L)$

(d)

From Table 2, the average length of all group members' paths should be minimized for our scheme to work efficiently. Algorithm *IsLeader* in subsection 4.3 places new members to subtrees with less leaf nodes by examining weight factors, and thus makes the average path length

small. For a sequence of N new members joining an initially empty group, each member's path length is at most $\lceil \log N \rceil$. We can roughly estimate that the average path length of all group member is of order $O(\log N)$, where N is the number of group members.

6 Conclusion and Future Work

In this paper, we propose a distributed key management protocol for dynamic groups, and each operation only takes two broadcast messages. The protocol algorithms are provably secure. The orders of the computation time to derive the group key, the storage space for every member, and the bandwidth for all kinds of messages are all small compared to the number of group members.

We are planning to improve the protocol to support subgroups. Scalability and efficiency can be improved, if every group operation only affects members in a subgroup. We are devising the protocol for non-broadcasting network environments, and developing more fault-tolerant algorithms.

References

[1] G. Ateniese, M. Steiner, and G. Tsudik. Authenticated group key agreement and friends. In *5th ACM CCS*, pages 17–26, Nov. 1998.

[2] G. Ateniese, M. Steiner, and G. Tsudik. Key agreement in dynamic peer groups. *IEEE Transactions on PDS*, 11(8), Aug. 2000.

[3] G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE JSAC*, Apr. 2000.

[4] K. Becker and U. Wille. Communication complexity of group key distribution. In *Proceedings of the 5th ACM CCS*, Nov. 1998.

[5] D. Boneh. The decision Diffie-Hellman problem. In *Third Algorithmic Number Theory Symposium*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, 1998.

[6] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 275–286. Springer-Verlag, 1995.

[7] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative group. In *Proceedings of ACM CCS (CCS-7)*, Nov. 2000.

[8] N. Koblitz. *A Course in Number Theory and Cryptography, 2nd edition*. Springer-Verlag, 1994.

[9] U. M. Maurer and S. Wolf. The Diffie-Hellman protocol. *Designs, Codes and Cryptography*, 19(2/3):147–171, 2000.

[10] S. Mittra. Iolus: A framework for scalable secure multicasting. In *ACM SIGCOMM*, pages 277–288, Sept. 1997.

[11] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *International Conference on Distributed Computing Systems*, pages 56–65, 1994.

[12] B. Schneier. *Applied Cryptography, 2nd edition*. John Wiley & Sons, Inc., 1996.

[13] W. Stallings. *Cryptography and Network Security: Principles and Practice, 2nd edition*. Prentice-Hall Inc., 1999.

[14] D. Steer, L. Strawczynski, W. Diffie, and M. Wiener. A secure audio teleconference system. In *Crypto' 88*, volume 403 of *Lecture Notes in Computer Science*, pages 520–528. Springer-Verlag, 1988.

[15] M. Steiner, G. Taudik, and M. Waidner. CLIQUES: A new approach to group key agreement. In *IEEE ICDCS*, pages 380–387, 1998.

[16] C. Wong, M. Gouda, and S. Lam. Secure group communications using key graphs. *Proceedings of the ACM SIGCOMM'98*, pages 68–79, Sept. 1998.

[17] P. R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.