Secure and Efficient Tree-based Group Diffie-Hellman Protocol

Sunghyuck Hong
Department of International Affairs, Texas Tech University
Lubbock, Texas 79409 – USA
[e-mail: sunghyuck.hong@ttu.edu]

Received February 20, 2009; revised March 27, 2009; accepted April 4, 2009; published April 25, 2009

Abstract

Current group key agreement protocols (often tree-based) involve unnecessary delays because members with low-performance computer systems can join group key computation. These delays are caused by the computations needed to balance a key tree after membership changes. An alternate approach to group key generation that reduces delays is the dynamic prioritizing mechanism of filtering low performance members in group key generation. This paper presents an efficient tree-based group key agreement protocol and the results of its performance evaluation. The proposed approach to filtering of low performance members in group key generation is scalable and it requires less computational overhead than conventional tree-based protocols.

Keywords: Group key agreement, group key management, secure group communication

The research was partially supported by the Department of International Affairs at Texas Tech University. I deeply appreciate the unknown reviewers' invaluable comments and very quick responses during the review time. Also I would like to express my great thanks to Jane Bell who is an International Cultural Center director.

DOI: 10.3837/tiis.2009.02.004

1. Introduction

An Enhanced Tree-based Group Diffie-Hellman (ETGDH) protocol is proposed that improves the existing TGDH, a tree-based group key agreement protocol that is currently the most efficient such protocol. The overall efficiency of TGDH is always determined by the slowest member's performance in generating a group key, because group communication under the TGDH protocol does not start until all members have completed generation of the group key. Under ETGDH, unnecessary delays are avoided, as low performance members are not allowed to participate in group key generation.

2. Enhanced Tree-Based Group Diffie-Hellman

To generate a group key efficiently, there must be a group controller. The group controller is the last member to join the group in the Enhanced Tree-based Group Diffie-Hellman (ETGDH) protocol. The newest member always plays the role of the group controller. The group controller is responsible for managing group key generation. He/she initiates group key generation for all members by sending them his/her blind key. Each member M_i selects a random private number r_i and computes $M_i = g^{ri} \bmod p$. All members participate in group key generation. When the group key has been computed, the highest performance member computes the final group key and distributes it to all members, and then group communication begins. Whenever membership changes, the group key must be regenerated. The definitions given in **Table 1** are used in the subsequent figures and equations.

Table 1. Definitions

Symbol	Definition
n	The number of group members
G	A set of current members
G_P	A set of partitioned group members
T	A key tree
T^*	A modified tree after membership operations
T_s	The sub tree of T
G_c	A group controller
M_{sc}	A subgroup controller
M_d	A leaving member
M_i	i^{th} group member; $i \in [1,n]$
Z_p^*	Integer set; $Z_p^* = \{1,2,\ldots, p\}$
p, g	A large prime number; $p \in Z_p^*$, exponentiation base; $1 < g < p$ *and* $g \in Z_p^*$
$<l, v>$	v^{th} node at level l in a tree (where $0 \le v \le 2^l - 1$)
$K_{<l, v>}$	$<l,v>^{th}$ node's random private key
$SK_{<l,v>}$	$<l,v>^{th}$ node's session key
$f(x)$	$g^x \bmod p$
$BK_{<l, v>}$	$<l,v>^{th}$ node's blind (public) key, $f(K_{<l, v>}) = g^{K<l, v>} \bmod p$
$T_c(M_i)$	i^{th} group member's response time for generating a key pair.

Fig. 1 and **Fig. 2** show a conventional tree-based group key agreement protocol used to

generate a group key. An example of key tree-based group key generation is shown in **Fig. 2**. Each node <*l, v*>'s private key and public key represent $K_{<l, v>}$ and $BK_{<l, v>} = g^{K<l, v>} \bmod p$ respectively, where *g* and *p* are public parameters. Every member holds the secret keys on the key path. For simplicity, each member knows the blind keys on the key path. The blind keys are the shaded nodes in **Fig. 2**.

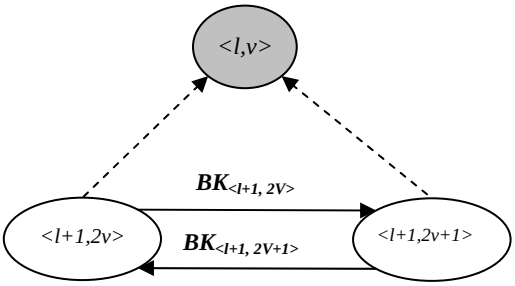

Fig. 1. Node <l,v>'s Session Key Generation

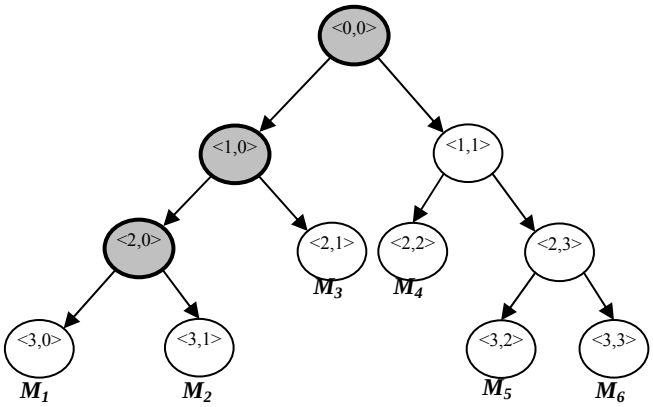

Fig. 2. Binary Key Tree

The random Session Key (SK) of the non-leaf node <*l, v*> in **Fig. 1** is generated by:

$$SK_{<l, v>} = (BK_{<l+1, 2v>})^{K<l+1, 2v+1>} = (g^{K<l+1, 2v>})^{K<l+1, 2v+1>} \bmod p \quad (1)$$

$$SK_{<l, v>} = (BK_{<l+1, 2v+1>})^{K<l+1, 2v>} = (g^{K<l+1, 2v+1>})^{K<l+1, 2v>} \bmod p \quad (2)$$

$$SK_{<l, v>} = g^{K<l+1, 2v>K<l+1, 2v+1>} \bmod p \quad (3)$$

Eq. (1) and Eq. (2) contain two exponents, which are the blind keys of node <*l+1,2v*> and node <*l+1,2v+1*> in Fig. 1. Node <*l,v*>'s session key is given by Eq. (3). Each member in a leaf node randomly selects a private key and generates a blind key. **Fig. 2** shows a tree for six members, where M_1 can compute $SK_{<2,0>}$, $SK_{<1,0>}$, $SK_{<0,0>}$ using the blind keys $BK_{<3,0>}$, $BK_{<3,1>}$, $BK_{<2,1>}$, and $BK_{<1,1>}$ [1].

The final group key G is computed as follows:

$$G = g^{g^{(g^{K<3,0>K<3,1>}})^{K<2,1>}} (K<2,2>)^{K<3,2>K<3,3>} \bmod p \quad (4)$$

Note that the group key G in Eq. (4) is computed in terms of all the blind keys on the key path in the key generation tree. The Group Diffie-Hellman (GDH) key agreement protocol [2] is an extension of the Diffie-Hellman (DH) key exchange protocol [3].

3. Group Key Secrecy and Security Model

Group key (GK) management is concerned with efficient generation of a GK in order to achieve and maintain secure communication. This process has four security requirements: (1) GK secrecy, (2) Backward secrecy, (3) Forward secrecy, and (4) Key independence [4]. Each of these security requirements are defined as follows:

Assume that a GK is changed m times and the sequence of GKs is given by $\{K_0, K_1,\ldots, K_{m-1}, K_m\}$.

1. GK Secrecy guarantees that it is computationally infeasible for a passive adversary to discover any GK, when $K_i \in K$ for all i.
2. Forward Secrecy guarantees that a passive adversary who knows a contiguous subset of old GKs (for example $\{K_0,K_1,\ldots,K_i\}$) cannot discover any subsequent GK, K_j when $j > i$ for all i and j
3. Backward Secrecy guarantees that a passive adversary who knows a contiguous subset of GKs (for example $\{K_i,K_{i+1},\ldots,K_j\}$) cannot discover a preceding GK, K_l when $l < i < j$ for all l, j, k,
4. Key Independence guarantees that a passive adversary who knows a proper subset of GKs $K_{subset} \subset K$ cannot discover any other GK, when $K_i \in (K - K_{subset})$.

Before examining GK secrecy, the possible types of security attacks must be defined.

There are two types of security attacks in group communication: active attacks and passive attacks. Active attacks involve injecting, deleting, delaying, and modifying protocol messages. Protecting against active attacks is beyond the scope of this paper. The security of network protocols such as Public Key Infrastructure (PKI) is assumed and we focus on protection from passive attacks.

When considering passive attacks, eavesdropping is the most significant threat. Efficient generation of GKs that meet the aforementioned four security requirements addresses the threat of eavesdropping. Secure GKs prevent attackers from discovering the GK and using it to decrypt the message.

A GK is a common secret key, which means that one key can encrypt/decrypt messages during communication, (i.e., it is a symmetric cipher). For symmetric ciphers, the most well known passive attack is the brute-force attack [5], which is the process of enumerating all possible keys until the proper key needed to decrypt a given cipher text into the correct plain text is found. All symmetric encryption algorithms will eventually fall to brute-force attacks if there is enough time.

In a brute-force attack, the expected number of trials needed to find the correct key is equal to half the size of the key space (i.e., the expected number of trials is the average of the best- and worst-case number of trials needed to find the right key). Symmetric ciphers with keys of 64 bits or less are vulnerable to brute force attacks [5]. Therefore, the key size must be large enough to prevent such an attack on symmetric ciphers. If the number of possible keys is enough to delay a brute-force attack, then the algorithm is secure [5]. GKs of 1,024 bits are secure with current technology [6]. A brute-force attack needs $2^{1,024}/2$ attempts to find the GK; such an approach is computationally infeasible. A cryptographic protocol is provably secure if

it can be shown to be as difficult to break as solving a well-known number-theoretic problem, such as the computation of discrete logarithms.

Another important security requirement of protecting against passive attacks via GK secrecy is maintaining key freshness. Whenever the group membership changes the GK is collaboratively regenerated. A GK is always fresh after a membership change; there is no chance to use an old key. In addition, GKs can be changed at regular or irregular intervals in order to ensure freshness. The fresher a GK, the more secure it is.

Fig. 3 shows an overview of a secure group communication protocol that relies on the *Secure Spread Library* tool key [7]. The *Secure Spread Library* is a group communication toolkit for wide and local area networks; it provides all the services of traditional group communication systems, including a multicast of messages to a group, network failure detection, and group membership services [8]. In addition, the *Spread* provides the members' login status to all members, so they can update the group key generation tree structure used to generate the group key. After member authentication, secure group communication is ready to start, and this must have a group key agreement protocol that can generate a group key in order to protect message integration.

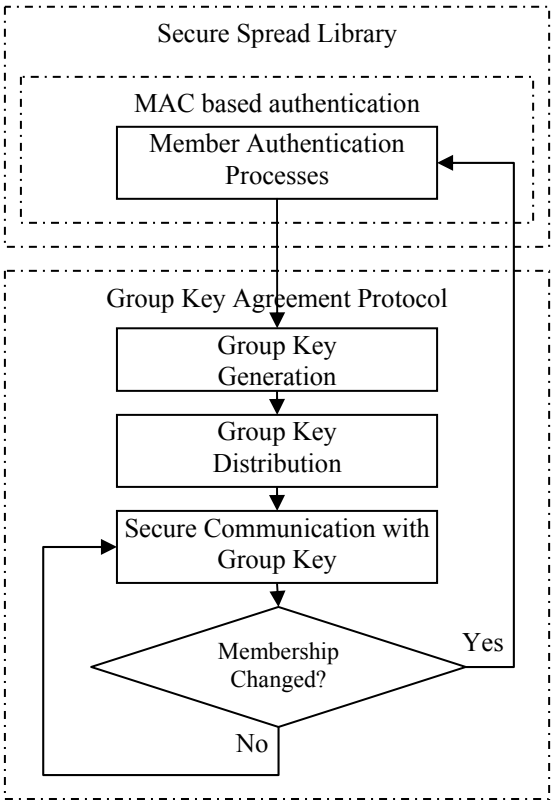

Fig. 3. Overview of Secure Group Communication

The principle of the ETGDH protocol relies on the premise that members in distributed computing do not have equal computing power. In other words, this algorithm admits diversity in member performance. Thus, during group communication, different roles should be allocated to the proper members in the group. This depends on each member's computing performance, because a higher level of the group key generation tree consumes more

computing time than lower levels.

Fig. 4 shows filtering of low performance members in the key tree. Leaf nodes represent members (M_1, M_2, M_3, M_4, M_5, M_6, M_7, and M_8). The sibling nodes in the tree are <M_1, M_2>, <M_3, M_4>, <M_5, M_6>, and <M_7, M_8>. Each member M_i generates a secret key $K_{<l, v>}$ as well as a blind key $BK_{<l, v>} = g^{K<l, v>} \bmod p$. In addition, the response time for generating the keys is measured, and then each member starts using Diffie-Hellman key exchange. For example, M_1 and M_2 exchange M_1 ($g^{K<3,0>} \bmod p$) and M_2 ($g^{K<3,1>} \bmod p$) in order to generate the sub-group key $g^{K<3,0>K<3,1>} \bmod p$.

Table 2. Key Generation

Key node	Member	Key Computation Process
<3,0>	M_1	$g^{K<3,0>} \bmod p$
<3,1>	M_2	$g^{K<3,1>} \bmod p$
<3,2>	M_3	$g^{K<3,2>} \bmod p$
<3,3>	M_4	$g^{K<3,3>} \bmod p$
<3,4>	M_5	$g^{K<3,4>} \bmod p$
<3,5>	M_6	$g^{K<3,5>} \bmod p$
<3,6>	M_7	$g^{K<3,6>} \bmod p$
<3,7>	M_8	$g^{K<3,7>} \bmod p$
<2,0>	M_2	$g^{<3,0><3,1>} = g^{K_{<3,0>}K_{<3,1>}} \bmod p$
<2,1>	M_4	$g^{<3,2><3,3>} = g^{K_{<3,2>}K_{<3,3>}} \bmod p$
<2,2>	M_6	$g^{<3,4><3,5>} = g^{K_{<3,4>}K_{<3,5>}} \bmod p$
<2,3>	M_8	$g^{<3,6><3,7>} = g^{K_{<3,6>}K_{<3,7>}} \bmod p$
<1,0>	M_4	$g^{<2,0><2,1>} = g^{g^{K_{<3,0>}K_{<3,1>}} g^{K_{<3,2>}K_{<3,3>}}} \bmod p$
<1,1>	M_8	$g^{<2,2><2,3>} = g^{g^{K_{<3,4>}K_{<3,5>}} g^{K_{<3,6>}K_{<3,7>}}} \bmod p$
<0,0>	M_8	$g^{<1,0><1,1>} = g^{g^{g^{K_{<3,0>}K_{<3,1>}} g^{K_{<3,2>}K_{<3,3>}}} g^{g^{K_{<3,4>}K_{<3,5>}} g^{K_{<3,6>}K_{<3,7>}}}} \bmod p$

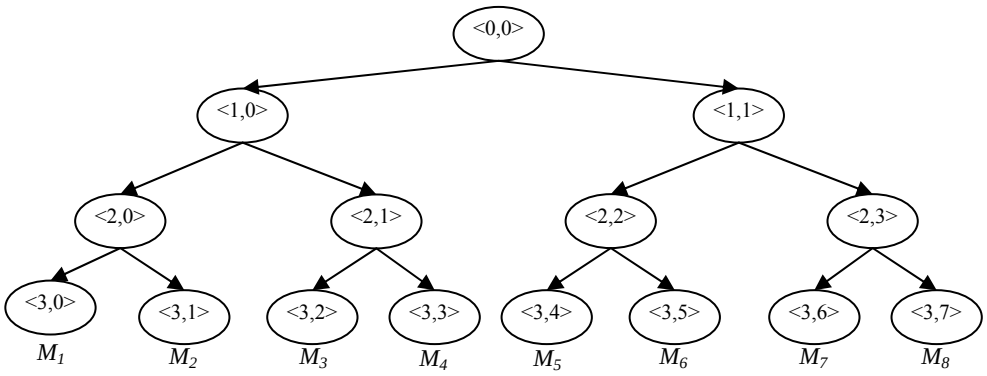

Fig. 4. Key Generation Tree

After completing the leaf level computation, a session key in the next level is ready to be generated. The group controller (who was the last member to join) determines which members join in the next level by comparing each member's response times, $T_c(M_i)$. Assuming the

higher performance members are M_2, M_4, M_6, and M_8, they advance to the next key generation until the final group key is obtained. The rest of the members wait until this has been completed. The next subgroup key pairs are $< M_2, M_4>$ and $< M_6, M_8>$. The higher performance members regenerate the upper level session key. Finally, the highest performance member M_8 (See **Table 2**) generates the final group key and distributes it to the rest of the members. **Table 2** indicates that low performance members do not participate in generation of the group key.

4. Protocol Requirements

The Enhanced Tree-based Group Diffie-Hellman (ETGDH) functions include member authentication, group key generation, and distribution of the group key to all members. The prototyping of ETGDH considers secure communication and algorithm efficiency during normal operations. Most experiments consist of measuring the time required for member authentication and group key generation.

Membership operations of the ETGDH protocol include join, leave, merge, and partition, based on the following preliminary requirements:

- All members must use their own computers to communicate with others.
- All members must have their public keys (certificates) with a MAC address for member authentication.
- The last member to login becomes the group controller and then starts group key generation by sending his/her blind key to current members.
- In case the group controller leaves, the second latest member becomes the group controller.
- Only approved members contribute an equal share to the group key. Non-authorized members should not participate in group key generation while it is in progress.
- Each member is required to know all blind keys in the entire key tree.
- All members are required to generate a private key and a blind key. After the membership changes, they must regenerate the group key for group key secrecy.

5. Membership Operations

A group key agreement protocol needs to provide membership operations to cope with membership changes. The ETGDH includes protocols in support of the following operations:

- Join: a new member is added to group communication
- Leave: a member is removed from group communication
- Partition: a subset of members is split from group communication
- Merge: a partitioned group is merged with current group communication

5.1 Join Protocol

Suppose that the group size is n. The group members are $M_1, M_2, M_3,\ldots, M_{n-1}, M_n$ for $n < 100$. Following any membership change, all members independently update the group key generation tree. Assuming that the underlying group communication system provides *Virtual synchrony* [8], all members correctly compute the key tree after any membership event [1]. The *Virtual Synchrony* protocol is established over the set of core members that can communicate with each other. This protocol provides guaranteed, in-order message delivery

for message streams that potentially involve one sender and multiple receivers. This guarantee is similar to the guarantees that TCP/IP provides for point-to-point message streams.

If the number of members is n, then a binary key tree of height $\lceil log_2 n \rceil$ is generated. All members are assigned from the leftmost to the rightmost leaf node on the tree according to the order of login and the last member (i.e., the group controller G_c) is assigned to the rightmost node $<log_2 n, n\text{-}1>$. **Fig. 5** shows the sequence of steps required for member M_j to join a group, where M_j is a new member who becomes the group controller (G_c).

Events in the algorithms are described as follows:

A ⟶ B

- The left side A is a group and the right side B is a group controller or amember.
- The arrow represents a control message.
- The destination can be either A or B.

Step 1: Each new member becomes a group controller (G_c) and broadcasts a request to generate the group key by sending his blind key.

- G_c sends his blind key to all members to compute the keys.

A blind key

Group G ⟵ G_c

Step 2: Every member:

- Updates the key tree (T^*) by removing all keys on the key-path and adding a new member node and a new intermediate node,
- Generates a private key and a blind key while measuring the response time of generating the key pair $T_c(M_i)$,
- Sends $T_c(M_i)$ to G_c.

Step 3: The group controller (G_c):

- Determines which members will join in the next level of computing the group key by comparing each member's response time $T_c(M_i)$ (i^{th} member, $i<n$) of generating the key pair.

Step 4: Approved members join in order to compute the group key.

Step 5: When the group key has been generated, the group key is sent to all members.

Fig. 5. Join Protocol

Fig. 6 shows an example of member M_5 joining a group. The rightmost node <2,3> in **Fig. 6A** becomes a new intermediate node in the updated Tree T^* in **Fig. 6B**. The detailed steps are as follows:

1. Rename node <2,3> in **Fig. 6A** to <3,0> in **Fig. 6B**
2. Generate a new intermediate node <2,3> and a new member node <3,1> in **Fig. 6B**
3. Promote <2,3> to the parent node of <3,0> and <3,1> in **Fig. 6B**

Since all members know $BK<3,1>$, approved members can compute the new group key $K<0,0>$ in the updated tree T^* in **Fig. 6B**.

5.2 Leave protocol

When a member leaves the group, the group key must be recomputed for group key secrecy. The leaving member broadcasts to all members in order to update the key tree structure. If the approved member leaves, G_c determines a back-up member and allows him/her to join group key generation. If the leaving member has a sibling node, then the sibling node is promoted to

replace the leaving member's parent node. In this case, only approved members contribute to computing the group key. The major steps are shown in **Fig. 7**, where M_d is leaving the group.

Fig. 8 shows an example of member M_3 leaving the group. The detailed steps are as follows:

1. Delete node <2,2> and the intermediate node <2,3> in **Fig. 8A**
2. Rename node <3,0> to <2,2> and <3,1> to <2,3> in **Fig. 8B**
3. Promote node <1,1> to the parent node of <2,2> and <2,3> in **Fig. 8B**

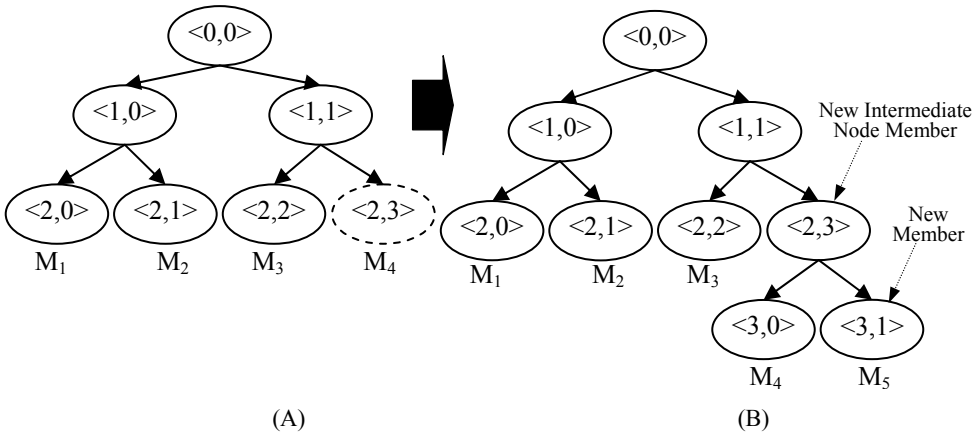

Fig. 6. Join Operation. A) Original Tree T, B) Updated Tree T*

Step 1: The leaving member (M_d):

- Broadcasts a control message to all members in order to update the key tree (T^*).
- The leaving member (M_d) and all the key nodes in the path up to the root are removed.

A broadcast to leave

Group G ← M_d

Step 2: The group controller (G_c):

- Selects a new member to participate in generating the group key if the leaving member M_d is a group key generating member. Otherwise, this is not necessary.

Step 3: Only selected members start computing the group key using the new tree T^*.

Step 4: When the group key has been generated, the group key is sent to all members.

Fig. 7. Leave Protocol

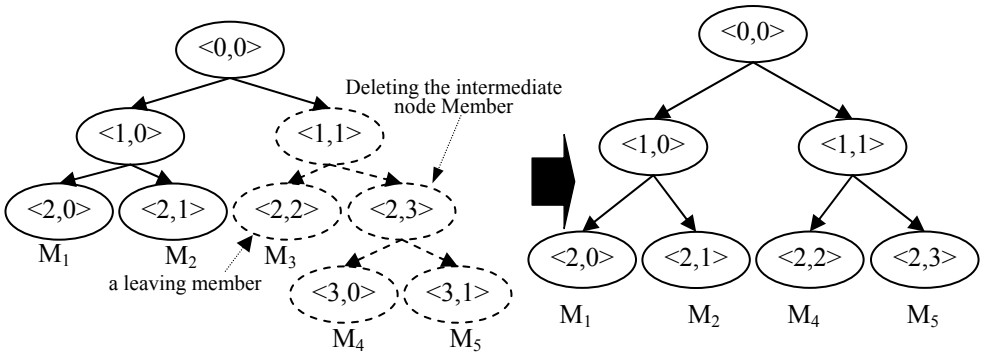

Fig. 8. Leave Operation. A) Original Tree *T*, B) Updated Tree *T**

After updating the tree, approved members can compute the group key. Note that M_3 cannot compute the group key, even though it knows all the blind keys, because its share is no longer part of the group key.

5.3 Partition protocol

Whenever there is a network fault, a group can be divided into several subgroups. The partition protocol is a multi-round protocol which runs until every approved member computes the new group key. If a network fault is detected, each remaining member updates its tree by deleting all partitioned members as well as their respective parent nodes. Each subgroup controller then computes all keys on its key-path as far up the tree as possible. Then, each subgroup controller broadcasts the set of new blind keys. Upon receiving a broadcast, each member checks whether the message contains a new set of blind keys. This procedure is iterated until all members obtain the group key. In case there is a partition protocol, a subgroup controller works as a group controller in the subgroup.

Fig. 9 illustrates the partition operation. Suppose the members M_1 and M_2 are partitioned from M_3, M_4, and M_5 in **Fig. 10A**. The rightmost node in each group becomes the subgroup controller. In **Fig. 10B**, the subgroup controllers M_2 and M_5 broadcast a request to generate the group key, thus, each group continues to communicate with the subgroup members using the group key. The major steps are shown in **Fig. 9**.

Step 1: Every member in the subkey trees:
- Updates the subkey tree (T_s) by removing all leaving member nodes and their parent nodes.

Step 2: The subgroup controller (M_{sc})
- Becomes the rightmost member of the subtree.
- Selects members for participating in generating the group key by checking each member's response time $T_c(M_i)$ for generating the key pair.
- Generates a new share and computes all key pairs on the key path in the subkey tree T_s.

Step 3: Selected members start computing the group key using the updated subkey tree T_s

Step 4: When the group key has been generated, it is sent to all members.

Fig. 9. Partition Protocol

5.4 Merge protocol

After recovering from network failures, the subgroups can be merged into the original group. In the first round of the merge protocol, each subgroup controller broadcasts his/her tree information with all blind keys to the other subgroups. Upon receiving this message, all members can uniquely and independently determine the merge position of the two trees. If the two trees have the same height, one tree is joined to the root node (insertion node) of the other tree. If the trees are of a different height, the smaller tree is joined to the larger.

The insertion node can be: 1) the rightmost smaller node (not necessarily a leaf node), if joining would not increase the height of the tree or 2) the root node, if joining to any other node would increase the height of the key tree. The rightmost member of the sub tree rooted at the joining location becomes the subgroup controller of the key update operation. The rightmost node in the updated tree T^* becomes G_c and computes every key on the key-path and the corresponding blind key. Then, he/she broadcasts the tree with the blind keys to the other members. All members then have the complete set of blind keys, enabling them to compute all keys on their key path. The major steps are shown in **Fig. 11**.

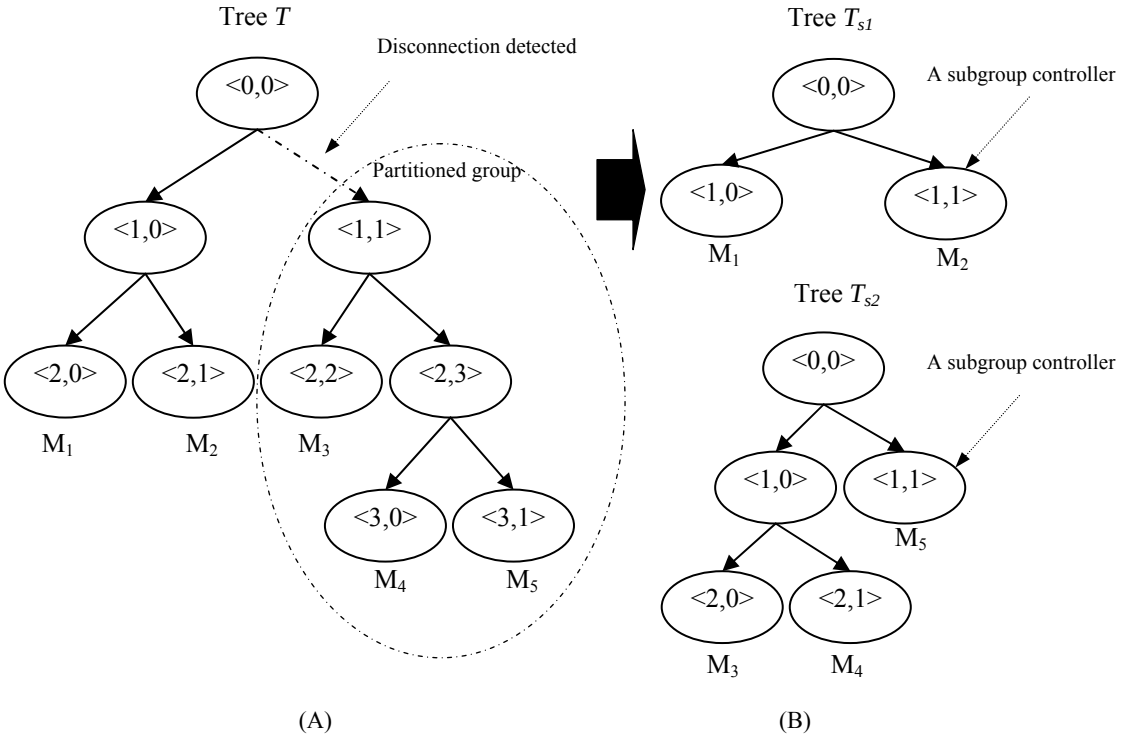

Fig. 10. Partition Operation. A) Original Tree T, B) Subtree T_{s1} and T_{s2}

A merge example is shown in **Fig. 12**. The smaller tree's merge node <0,0> in **Fig. 12A** and the larger tree's merge node <0,0> in **Fig. 12B** merge into the updated tree T^* in **Fig. 12C**. The group controller M_5 in **Fig. 12C** broadcasts a request to generate a new group key. After each member has completed the group key, secure group communication starts.

Step 1: Subgroup controller (M_{sc}):
- Requests subgroup controllers for merger,
- Merges two groups and updates new tree (T^*).

Step 2: The group controller (G_c):
- The G_c who is the rightmost leaf node in T^* is automatically selected.
- Determines the members to participate in generating the group key by checking each member's response time $T_c(M_i)$ for generating a key pair,

Step 3: Current members compute a private key and a blind key in pairs, while measuring the response time for generating the key pair $T_c(M_i)$.
Step 4: Selected members compute the group key using T^*.
Step 5: When the group key has been generated, it is sent to all members.

Fig. 11. Merge Protocol

6. Experimental Comparisons

In this section the proposed protocol (ETGDH) is compared to other contributory group key agreement schemes including TGDH, GDH, BD, and STR.

A cluster of 13 SUN Ultra Spac Stations running Sun Solaris was used for the experiments. The group members were uniformly distributed across the thirteen machines. Therefore, 1-60

processes could be run on a single machine and the response times of computing a group key could be measured. Each process running on a single machine acts like a single member in group key generation. The tests designed were run on the testbed in order to measure the average response times of computing a group key and the communication overhead of sending and delivering messages under the selected protocols, which included GDH (Group Diffie-Hellman) [2], BD (Burmester and Desmedt) [9], STR (Skinny Tree) [10], and TGDH (Tree-based Group Diffie-Hellman) [4].

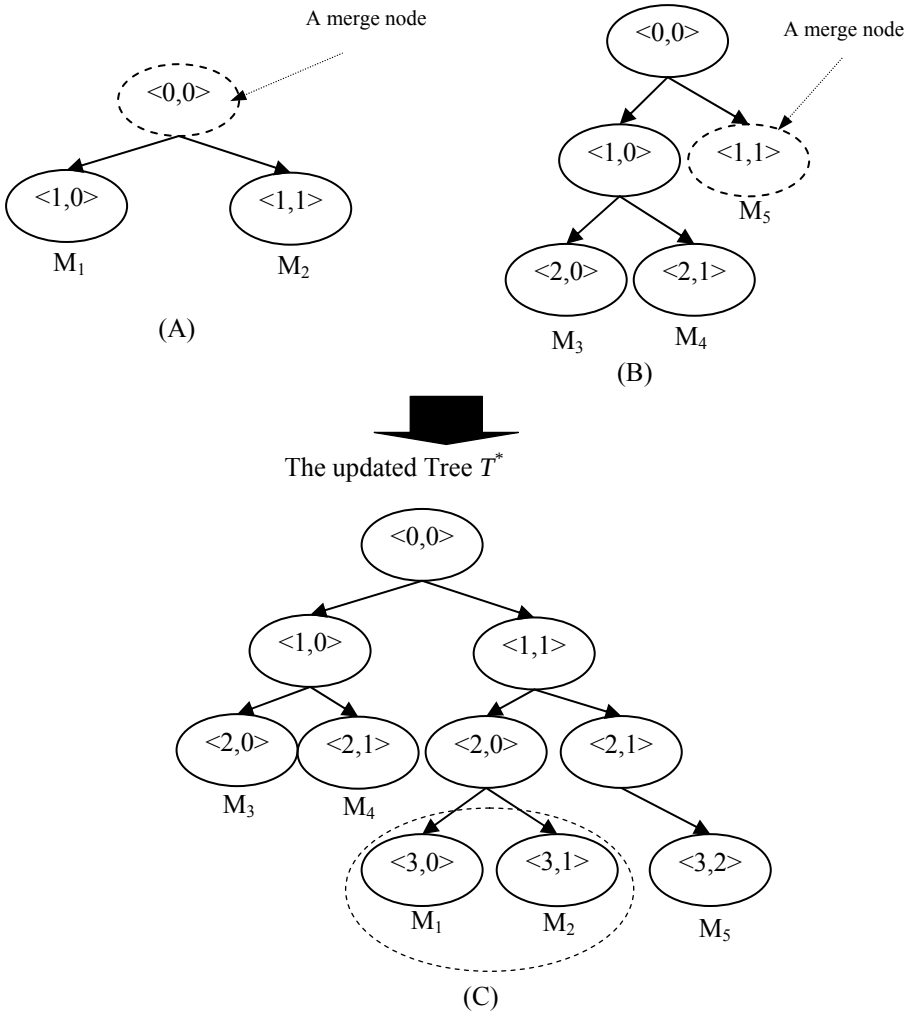

Fig. 12. Merge Operation. A) The smaller tree T_{s1}, B) The larger tree T_{s2}, C) Updated Tree T^*

6.1 Join Operation

Fig. 13 shows that BD and GDH are inefficient, because the join operation exhibits an increasing overhead (i.e., communication and computation costs) with respect to an increasing group size. On the other hand, ETGDH, TGDH, and STR are efficient, because they use a divide-and-conquer algorithm to compute the group key. ETGDH is the most efficient, scaling logarithmically with respect to the number of exponentiations. Both TGDH and STR use a binary tree to compute the GK. ETGDH involves $2n - 2$ more communication messages than

any other protocol. However, the relatively large communication rate does not adversely affect the performance in computing the GK, because it takes a maximum of 20 micro-seconds to exchange messages between group members in GK generation. Therefore, ETGDH is more efficient than the other tree-based GK computation protocols.

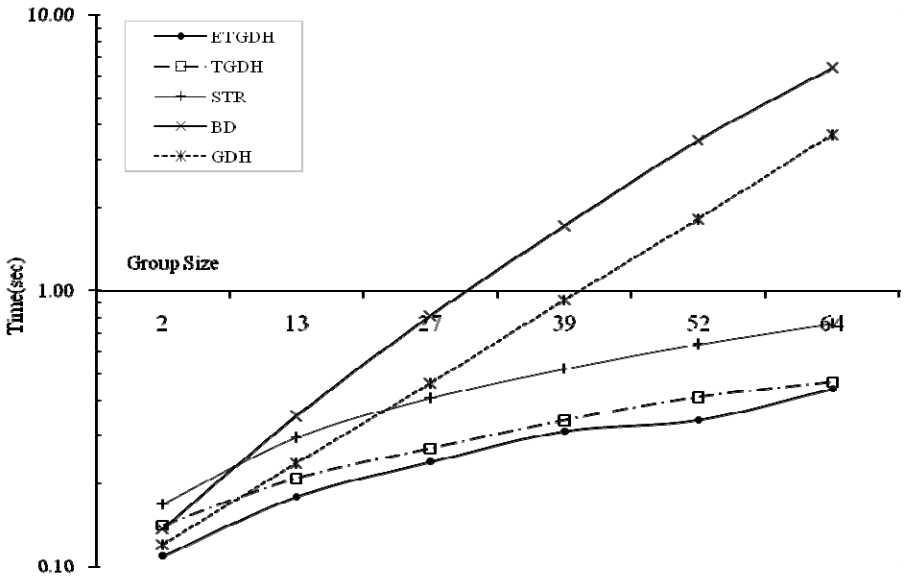

Fig. 13. Comparison of Join Cost: Group Size versus Time

6.2 Leave operation

Fig. 14 shows the computational delay of a random member leaving a group of size n (= 10, 20, 30, 40, 50, and 60). The leave cost depends on the number of remaining members, so it does not matter how many members remain. Thus, the leave cost is almost the same as the join cost. However, the computation cost of STR for a leave event depends on the location of the leaving member in the key generation tree. Therefore, the efficiency of STR in terms of the leave cost is lower than the efficiency in terms of the join cost.

6.3 Partition operation

Fig. 15 shows the partition operation results. Whenever there is a network fault, the group is divided into several subgroups. The group can be divided and their sub group keys must be recomputed. If a network fault is detected in ETGDH and TGDH, each remaining member updates its tree by deleting all partitioned members as well as their respective parent nodes. Each subgroup controller then computes all keys on its key-path as far up the tree as possible. Then, each subgroup controller broadcasts the set of new blind keys. Upon receiving a broadcast, each member checks whether the message contains a new set of blind keys.

For BD and GDH, the location of the partitioning members is irrelevant to the overall performance. However, the partitioning members' location is important in ETGDH, STR and TGDH. Fig. 15 shows that the overhead of TGDH while managing the tree structure increases until half of the group has been partitioned, after which it decreases. Since the group was evenly divided into two groups in TGDH, the maintenance cost of the tree was maximized but more messages were required. On the other hand, the cost of BD and GDH decreases linearly,

because it was inversely proportional to the number of remaining group members. If the number of remaining members is small, then the overhead for all protocols other than TGDH and ETGDH is large.

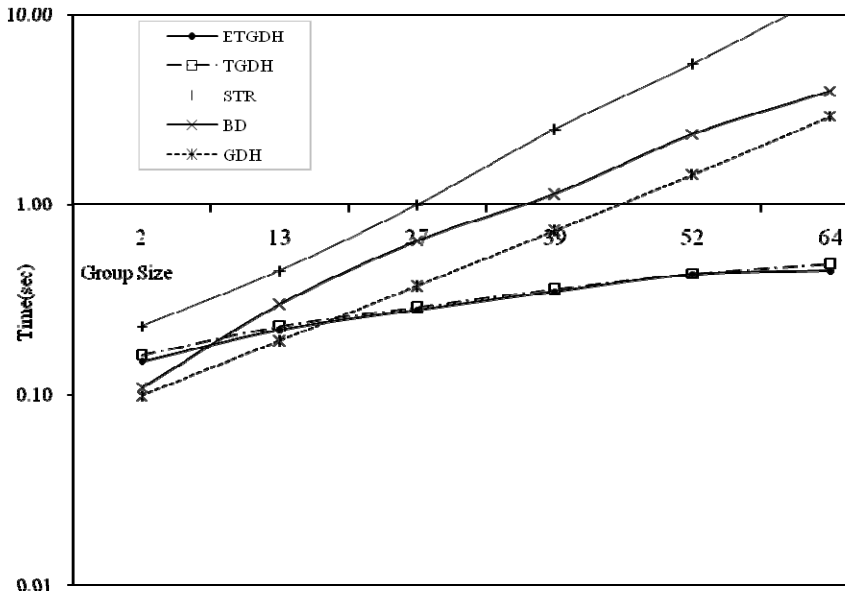

Fig. 14. Comparison of Leave Cost: Group Size versus Time

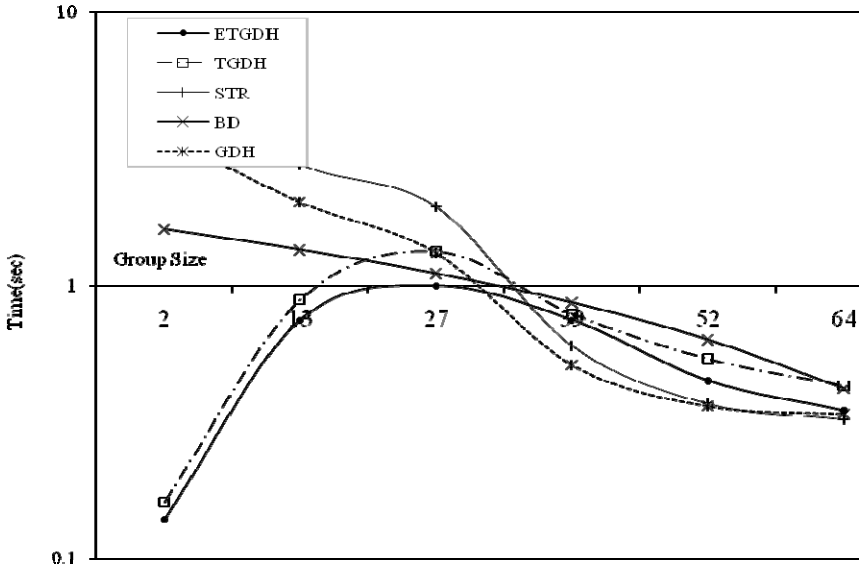

Fig. 15. Comparison of Partition Cost: Group Size versus Time

6.4 Merge operation

Fig. 16 shows the merge operation results. After recovering from a network fault, there is a

merge operation. The overall performance of all protocols other than GDH is good. The performance of GDH strongly depends on the current group size. In addition, it involves $n + 2m + 1$ exponentiations and $n + 2m + 1$ messages. For ETGDH and TGDH, the current group size is irrelevant. Therefore, the performance of the tree-based protocols is almost constant.

Based on the experimental results on computational cost, ETGDH shows the best performance, despite the relatively large number of message exchanges. ETGDH adopts a tree structure in order to generate the GK, which is similar to TGDH. In addition, ETGDH considers user diversity, so only approved members can join in GK generation.

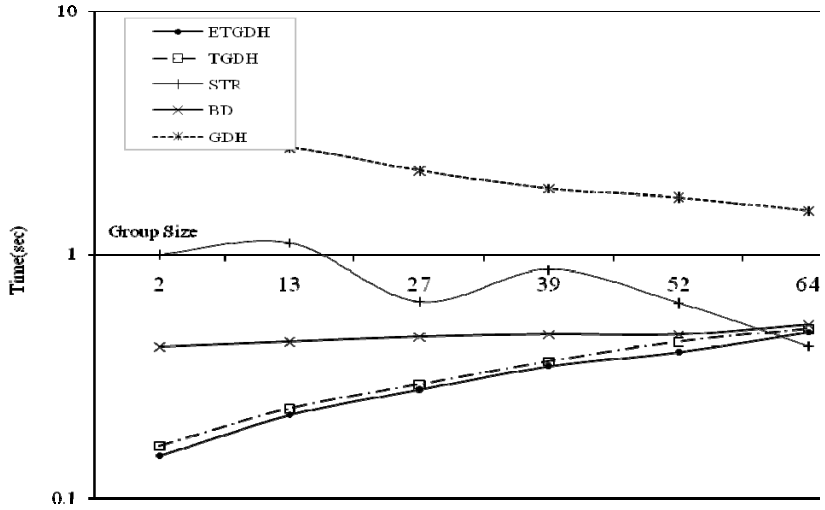

Fig. 16. Comparison of Merge Cost: Group Size versus Time

6.5 Summary

To evaluate the efficiency, the communication and computation overhead of the proposed protocol (ETGDH) were measured in terms of the total response time of generating a group key, and then compared to existing protocols. Results show that ETGDH, which uses a queue structure to determine high performance members, provides the highest efficiency in terms of the response time of generating a group key, even though it involves a relatively large number of message exchanges. Therefore, the factors which must be considered in order to improve efficiency in group key generation include tree maintenance, diversity of computing power, and heterogeneity of network environments. The results were obtained for a small LAN. However, a future research direction is to determine if the communication costs in larger networks (e.g. wide area networks) are greater than the group key generation costs, and to develop a careful experimental design to accurately assess the overhead of the proposed queue-based protocol.

7. Conclusion

Group key generation is an important issue in group key management. Several attempts to enhance group key generation have been reported [11]. Group key management focuses on minimizing the computational overhead of inherently expensive cryptographic operations [10]. As a result of the complexity of group key generation, group key management needs to adopt a key tree structure in order to reduce group key generation times. Key trees have been

suggested for centralized group key distribution systems in order to reduce the complexity of key calculation [12]. One efficient group key generation protocol is TGDH [13]. The TGDH protocol has two necessary preconditions, (a) the key tree is perfectly balanced and (b) all members in the group have equal computing power. If the two preconditions are not satisfied then TGDH cannot perform with an efficiency of O (log n). As mentioned earlier, the members in a distributed computing environment can use any computing machine. Conventional group key agreement protocols require all members to contribute to group key generation. Consequently, it is possible for low performance members to join, which will result in delays. To improve group key generation, (a) low performance members should not participate in generation of the group key and (b) relatively high performance members should be authorized to participate in calculation of the group key. The performance of each member's computing machine is classified as high or low.

This paper proposed an enhanced group key agreement to filter low performance members and achieve maximum efficiency in group key generation.

References

[1] Y. Kim, A. Perrig, and G. Tsudik, "Tree-based Group Key Agreement," *ACM Transaction on Information and System Security*, ACM Press, 2004.

[2] E. Bresson, O. Chevassut, et al., "Provably authenticated group Diffie-Hellman key exchange," in *Proceedings of the 8th ACM conference on Computer and Communications Security*, Philadelphia, PA, 2001.

[3] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, pp.644-654, 1976.

[4] Y. Kim, "Group Key Agreement: Theory and Practice," Ph.D. thesis, Dept. of Computer Science, University of Southern California at LA, 2002.

[5] E. Cole, R. K. Krutz, et al., "Network Security Bible," John Wiley & Sons, 2005.

[6] A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," *Journal of Cryptology*, vol. 14, no. 4, pp.255-293, 2001.

[7] Y. Amir and J. Stanton, "The Spread wide area group communication system," Center of Networking and Distributed Systems: Tech. Rep. 98-4, Johns Hopkins University, 1998.

[8] A. L. N. Fekete and A. Shvartsman, "Specifying and using a partionable group communication service," In *ACM PODC '97*, Santa Barbara, CA, 1997.

[9] M. Burmester and Y. Desmedt, "A secure and efficient conference key distribution system," *Advances in Cryptology - EUROCRYPT'94*, 1994.

[10] Y. Kim, A. Perrig, and G. Tsudik, "Communication-efficient group key agreement," in *17th International Information Security Conference*, 2001.

[11] Y. Amir, Y. Kim, C. Nita-Rotaru, and, G. Tsudik, "On the Performance of Group Key Agreement Protocols," *ACM transactions on information and system security*, vol. 7, no. 3, pp.457- 488, 2004.

[12] D. Wallner, E. Harder, et al., "Key management for multicast: Issues and architecture," in *MILCOM 98*, 1998.

[13] Y. Kim, A. Perrig, et al., "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *7th ACM Conference on Computer and Communications Security*, 2000.

Sunghyuck Hong received his B.A. degree from Myongji University, Korea in 1995. After graduation, he worked at Hyosung Inc. in Seoul, Korea from 1995 to 1999 as a computer programmer and ERP consultant. He has a Ph.D. degree from Texas Tech University in August, 2007, major in Computer Science. Currently, he works at International Affairs in Texas Tech University as a senior programmer/analyst. His current research interests include secure authentication, secure group communication, wireless sensor networks, biometric authentication, and key management protocol.