Design and Performance Analysis of Queue-based Group Diffie-Hellman Protocol (QGDH)

Sunghyuck Hong[1] and Sungjin Lee[2]
[1] Division of Information and Communication, Baekseok University
Cheonan, Korea
[2] Division of Information and Communication, Baekseok University
Cheonan, Korea
[e-mail: {shong,lsj}@bu.ac.kr]
*Corresponding author: Sungjin Lee

revised December 7, 2012; accepted December 21, 2012; published January 29, 2013

Abstract

Current group key agreement protocols, which are often tree-based, have unnecessary delays that are caused when members with low-performance computer systems join a group key computation process. These delays are caused by the computations necessary to balance a key tree after membership changes. An alternate approach to group key generation that reduces delays is the dynamic prioritizing mechanism of queue-based group key generation. We propose an efficient group key agreement protocol and present the results of performance evaluation tests of this protocol. The queue-based approach that we propose is scalable and requires less computational overhead than conventional tree-based protocols.

Keywords: Group key management, security, group communication, communication complexity

http://dx.doi.org/10.3837/tiis.2013.01.010

1. Introduction

Group communication on the Internet is exploding in popularity. Videoconferencing, enterprise IM, desktop sharing, and numerous forms of e-commerce are but a few examples of the ways in which the Internet is being used for business. The growing use of group communication has highlighted the need for advances in security. There are several approaches to securing user identities and other information transmitted over the Internet. One of the foundations of secure communication is key management, a building block for encryption, authentication, access control, and authorization [10]. Key management and member authentication processes take place at the beginning of group communication. To establish a secure group, all members are authenticated, then generate and use a common group key (GK) to encrypt and decrypt messages [2].

To achieve a high level of security, the GK should be changed after any member joins or leaves so that new members have no access to previous communications and former group members have no access to current communications [15]. One problem inherent in this process is that the computation of GKs often takes a significant amount of time – even when a group's size is relatively small [14]. To address this problem, a recent focus in key management is the efficient generation of GKs [3][12][14]. It is this need for efficient key generation that we address.

We present an innovative approach to GK generation, the Queue-based Group Diffie-Hellman protocol (QGDH). QGDH provides a queue-based divide and conquer algorithm that is more efficient than the Tree-based Group Diffie-Hellman (TGDH) protocol and Enhanced Tree-based Group Diffie-Hellman (ETGDH) protocol [8], both of which use a key tree in the GK generation process [12]. Our goal is to provide an efficient GK agreement protocol that does not require a high level of computational overhead to maintain secure communication.

This paper is organized as follows. In Section 2, we describe how GKs ensure security of group communications, we introduce conventional GK generation protocols, including TGDH and ETGDH, and we highlight the limitations of these existing protocols. In Section 3, we present QGDH, our protocol for GK generation. We also describe how our protocol executes the join, leave, partition, and merge operations needed to maintain secure communication. Then, in Section 4, we test the performance of QGDH in a series of experiments. QGDH is shown to outperform TGDH and ETGDH in multiple tasks and on multiple types of computer systems. Our conclusions are presented in Section 5.

2. Group Key Agreement Protocols

2.1. Group Key Management

Group key (GK) management focuses on how to efficiently generate a GK to enable and maintain secure communication. In this process, there are four requirements to ensure security: (1) GK secrecy, (2) backward secrecy, (3) forward secrecy, and (4) key independence [10]. Each of these security requirements are defined as follows.

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

1. GK Secrecy guarantees that it is computationally infeasible for a passive adversary to discover any GK $K_i \in K$ for all i.

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

Before examining GK secrecy, possible 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. For the purposes of this paper, we place protecting against active attacks outside of our scope. We assume the security of network protocols such as Public Key Infrastructure (PKI) and now turn our focus to the topic of protection from passive attacks.

When considering passive attacks, eavesdropping is the most significant possible threat. Efficient generation of GKs that meet the four requirements for security described above addresses the possible threat of eavesdropping. Secure GKs prevent attackers from discovering the GK in order to decrypt the message.

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

In a brute-force attack, the expected number of trials before the correct key is found is equal to half the size of the key space (the expected number of trials is the average of the best-case and worst-case number of trials to find the right key). Symmetric ciphers with keys of 64 bits or less are considered vulnerable to brute force attacks [5]. Therefore, key size must be great enough to prevent such an attack on symmetric ciphers. If there are enough possible keys to slow down a brute-force attack, then the algorithm can be considered secure [5]. GKs that are 1,024 bit-long numbers are considered to be secure in current technology [13]. A brute-force attack would need $2^{1,024/2}$ attempts to find the GK; such an approach is considered computationally infeasible. A cryptographic protocol is said to be provably secure if it can be shown to be essentially as difficult as solving a well-known and typically number-theoretic problem, such as the computation of discrete logarithms.

Another important security requirement when protecting against passive attacks by maintaining GK secrecy is to maintain key freshness. Whenever the membership of a group changes, a GK will be collaboratively regenerated. A GK is always fresh after a membership change; the chance to use an old key does not exist. In addition, GKs can be changed on regular or irregular intervals to ensure freshness. The fresher a GK is, the more secure it is considered.

Our protocol will protect against each of these types of passive attacks, as we will explain in the balance of this paper.

2.2. TGDH (Tree-based Group Diffie-Hellman) Protocol

Modular exponentiation is an effective but computationally expensive operation that is used to generate GKs for secure group communication [10]. One of the goals of key management is to minimize the number of modular exponentiations and the number of protocol rounds. To the degree that this goal can be achieved, the key management process can be considered efficient.

TGDH was proposed to increase the efficiency of the GK generation process by using a tree structure [12]. As its name indicates, TGDH is built upon the well-known Diffie-Hellman key exchange protocol [6] and its extension, the Group Diffie-Hellman (GDH) key agreement protocol [2] that enables provably secure, fully distributed GK generation [10]. In TGDH, the number of exponentiations for a membership event depends on the current tree structure and total number of members in the group.

To generate a GK efficiently in TGDH, a group controller must exist to manage the overall GK generation process. The newest member to join a group always fills the role of group controller. This member initiates all members into the GK generation process by sending his/her blinded key to them by using Virtual Synchrony[1] [7].

2.3. Group Key Generation Process

When the GK generation process begins, each member M_i selects a random private number r_i and computes its blinded key, $M_i = g^{ri} \bmod p$. All blinded keys must be shared. When the GK has been generated, then group communication can begin. Whenever membership changes, the GK must be regenerated.

A series of figures will now be presented to describe GK generation in the TGDH protocol. The symbols, characters, and abbreviations used in the figures are summarized in **Table 1**.

Table 1. Definitions

Symbol	Definition
n	The number of group members
GK	Group key
M_i	i^{th} group member; $i \in [1,n]$
Z_p^*	Integer set; $Z_p^*=\{1,2,\ldots,p-1\}$
p, g	A large prime number; $p \in Z_p^*$, exponentiation base; $g \in Z_p^*$
$<l, v>$	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 blinded (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 shows how a Session Key (SK) is generated for a given non-leaf node,$<l,v>$. The SK is an intermediate key that is created as part of the GK generation process. The random SK of a non-leaf node $<l, v>$ in **Fig. 1** is generated by:

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

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

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

Equations (1) and (2) contain two exponents, which are the blinded keys of node $<l+1,2v>$

[1] This assumes that the underlying group communication system provides Virtual Synchrony so that all members correctly compute a group key after any membership events [12]. The Virtual Synchrony protocol provides guaranteed, in-order message delivery for message streams that involve one sender and potentially multiple receivers. This guarantee is similar to the guarantees that TCP/IP provides for point-to-point message streams.

and node <*l+1,2v+1*> in **Fig. 1** Node <*l,v*>'s SK is given by (3). Each member in a leaf node randomly selects a private key and generates a blinded key.

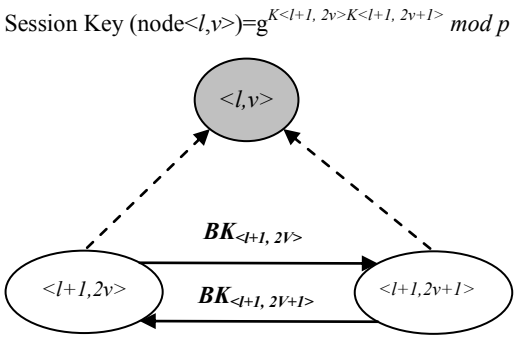

Fig. 1. Node <*l,v*>'s Session Key Generation Process

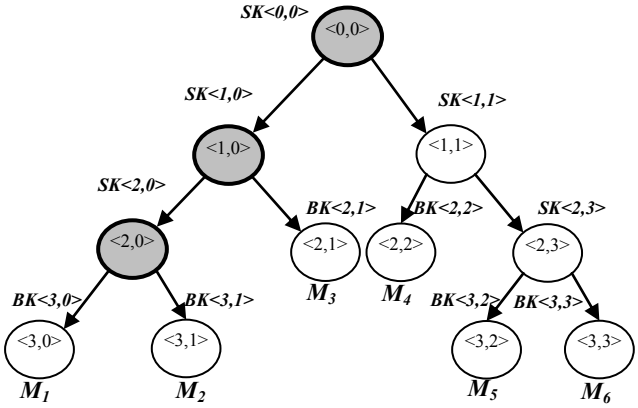

Fig. 2. Binary Key Tree

Fig. 2 extends **Fig. 1** and demonstrates how the GK is computed in a setting with multiple group members. An example of the tree-based GK generation process 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 along the key path. For simplicity, each member knows the blinded keys in the key path. The blinded keys are the shadowed nodes in **Fig. 2**.

Fig. 2 shows a tree for 6 members, where M_1 can compute $SK_{<2,0>}$, $SK_{<1,0>}$, $SK_{<0,0>}$ using the blinded keys $BK_{<3,0>}$, $BK_{<3,1>}$, $BK_{<2,1>}$, and $BK_{<1,1>}$.

The final GK is computed as follows:

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

Note that the *GK* in equation (4) is computed in terms of all the blinded keys on the key path in the key generation tree.

2.4. Shortcomings of TGDH

In spite of the computational advantages of the TGDH protocol, two shortcomings exist. One shortcoming is that TGDH does not generate GKs based on the relative performance of group members' systems. Restated, the TGDH protocol assumes that all members have equal computing power. In actuality, heterogeneity exists in distributed computing environments in that some group members have high-computing-power systems while other group members have lower-computing-power systems. Because secure group communication will not be able to start until every member has a GK, the assumption of equal computing power means that the overall performance of the GK-generation process is limited by the lowest-performance group member.

A second shortcoming is that TGDH uses a tree structure for GK generation. For maximum performance, a tree must be well-balanced. Because group members leave the GK generation tree randomly, such trees are either unbalanced and fail to achieve maximum performance or must be continually re-balanced in a process that generates computational overhead and slows GK generation [12].

To resolve the first shortcoming for TGDH, the Enhanced TGDH (ETGDH) [8] was proposed. We now turn to a discussion of the ETGDH.

2.5. ETGDH (Enhanced Tree-Based Group Diffie-Hellman)

The ETGDH protocol addresses the fact that group members often have computer systems with differing levels of system performance. Performance is measured in terms of the time it takes a member's system to respond to the request for a secret key. Because system performance often differs, there are different roles that should be allocated to certain members in the group. Because the higher levels of the GK generation tree take more computing time than the lower levels of the GK generation tree do, members should be sorted so that the highest-performance member's system begins the GK-generation process.

Leaf-level computation is the first step in the process of generating the GK – and in the process of determining the relative performance of group members' systems. Leaf-level computation is depicted in **Fig. 3**. 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 $K_{<l, v>}$ and generates a blinded key $BK_{<l, v>} = g^{K<l, v>} \bmod p$. The response time for generating the keys is measured, and then each member starts using the Diffie-Hellman key exchange. For example, M_1 and M_2 exchange the values M_1 ($g^{K<3,0>}$ *mod p*) and M_2 ($g^{K<3,1>}$ *mod p*) to generate sub-group key $g^{K<3,0>K<3,1>}$ *mod p.*

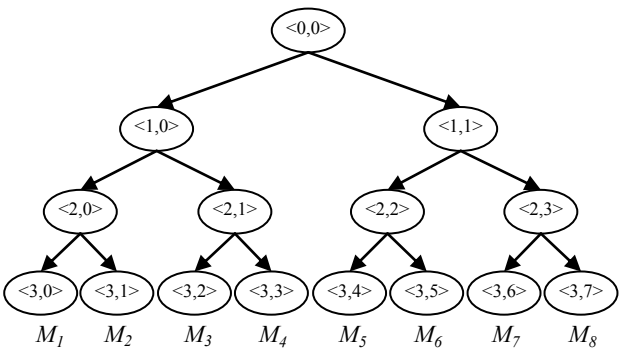

Fig. 3. Key Generation Tree

After completing the leaf level computation, a session key in the next level is ready to be generated. The group controller (the last member to join) determines which member goes to the next level. This determination is made by comparing each member's response times, $T_c(M_i)$. Assuming that the members who use higher-performance systems are M_2, M_4, M_6, and M_8, then each of these members will advance to the next level of the key generation process. The rest of the members will wait until the processes have been completed. The pairs at the next level will be $< M_2, M_4>$ and $<M_6, M_8>$. The highest-performance members, $<M_4, M_8>$, generate the upper level session key. Finally, the highest performance member, M_8, generates a final GK and distributes it to the rest of the members. Lower-performance group members do not participate in the generation of the GK. Each of the steps in the key generation process that is depicted in **Fig. 3** are explicitly listed in **Table 2**.

Table 2. Key Generation Processes

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	$g^{<1,0><1,1>} = g^{g^{g^{K<3,0>K<3,1>} g^{K<3,2>K<3,3>}} g^{g^{K<3,4>K<3,5>} g^{K<3,6>K<3,7>}}} \bmod p$

Efficiency gains in GK generation over TGDH can be realized if GKs are generated by a protocol that: (1) determines which members are high-performance and which are low-performance, (2) prioritizes the generation of GKs for high-performance users, and (3) generates those GKs in descending order from the highest-performance member to the lowest. ETGDH is able to realize these efficiency gains, but it cannot resolve the second shortcoming of TGDH that we noted earlier, the tree maintenance problem. To reiterate, the key tree must be rebalanced every time a GK is generated after group membership changes. This rebalancing is vital to achieve maximum performance. Because ETGDH leaves the shortcoming of tree maintenance unaddressed, we also note that efficiency gains in GK generation can be realized when a protocol (4) uses a structure that is more efficient than the tree-based structure. Our proposed QGDH protocol does just this, providing secure communication more efficiently than any other conventional efficient GK generation protocol.

3. Queue-Based Group Diffie-Hellman

The QGDH protocol that we propose duplicates the efficiency gains of ETGDH by (1) determining which members are high-performance and which are low-performance, (2)

prioritizing the generation of GKs for high-performance members, and (3) generating those GKs in descending order from the highest-performance member to the lowest. In addition, a queue structure is used to determine high- and low-performance members, yielding efficiency gains over tree-based protocols in the generation of GKs.

QGDH, as its name indicates, uses a queue structure rather than a tree structure. When group membership changes and a new GK is required, blinded keys are computed and stored in a blinded key queue (BKQ) on the group controller's system. The highest-performance member's key is stored at the front of the queue and the lowest-performing member's key is stored at the back of the queue (with the remaining members' keys stored somewhere in between, depending on the performance of the member's system and thus, on the time of the key's arrival in the BKQ). This arrangement is advantageous because the first element in the queue (which has been received from the highest-performance member) will be the first one processed. This process avoids delays in generating a GK.

An additional benefit of the queue structure can be seen when comparing it to a tree structure. Queues have a simple linear data structure and can be changed when group members join or leave with minimal computational overheard. In contrast, trees must maintain a balanced structure to achieve and maintain maximum performance. Thus, an advantage of a queue structure over a tree structure is that a queue structure provides a useful way to determine high-performance members, one that avoids the additional computational overhead needed to maintain balance in a key tree.

3.1. Reducing Overhead Costs in GK Management

There are two overhead costs in a GK management protocol: the first is computation cost, and the second is communication cost. Computation and communication costs are important because they impact the scalability of GK management [10]. Communication cost is relatively small and constant (often using a 3-round protocol [9]). Computation costs, in contrast, dominate the performance of GK management [10] because those costs depend directly on group size and because GK generation requires logarithmically-many exponentiations. All members' keys must have a contribution to calculate a GK that ensures GK secrecy [4]. Therefore, QGDH focuses on reducing computation costs by assessing the computing power of group members' systems.

In general, the computing power of a member's system depends on CPU capacity, main memory size, and the type of operating system. It is, however, impossible to retrieve another machine's system information and evaluate all of its individual factors by normal network operations due to security reasons. The only way to evaluate each machine's performance is to measure the total response time.

To determine high performance members efficiently, a *group controller* is introduced in QGDH. The group controller is the last member to join the group. The group controller also is responsible for controlling the overall group key generation process by assigning members to the proper positions in the key generation whenever changes occur in the group membership. The role of the group controller in a dynamic peer group communication can potentially improve the group key generation process. He does not have any privilege. The *group controller* requires all members to compute their public keys and store them into a queue structure on the *group controller* in the order of arrival. The queue is a First-In-First-Out (FIFO) queue where the high performance members' keys are always stored into the front positions in the queue. The benefit of using a FIFO queue is that QGDH provides an efficient way to determine high performance members' machines without the additional overhead of actively comparing each member's response time [8].

3.2. How QGDH Works

The QGDH protocol works in the following way. The group controller broadcasts a request to all members to generate a blinded key. The group controller receives all blinded keys and stores them into its' BKQ in the order of their arrival, with the higher-performance members' blinded keys stored in the front of the BKQ and the lower-performance members' blinded keys stored in the rear of the BKQ.

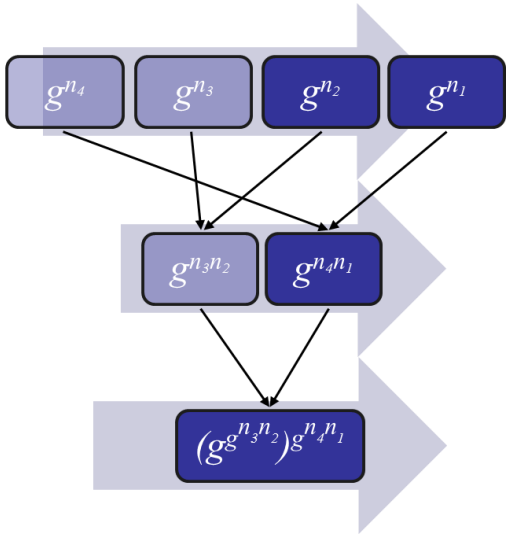

Fig. 4. Blind Key Queue Structure

The group controller then requests members who are in the front half of the BKQ (the higher-performance members) to compute their Diffie-Hellman key exchange with those blinded keys and store them in the next level of the BKQ in order of arrival. Following the determination of each level, the group controller collects and stores all computed session keys in the BKQ in **Fig. 4**. The highest-performance member's key is always stored into the first spot in each level, the second highest performance member's key is stored into the second spot, and so on. The keys in darker areas in **Fig. 4** are determined as faster member's keys. The BKQ automatically assigns each pair of keys. For example, the first spot's blinded key will be computed with the last spot's blinded key (g^{n4n1}); the second spot's blinded key will be computed with the second-to-last spot's blinded key (g^{n3n2}).

If group size is odd, then one of the member's keys will not be able to calculate with other keys. In such a situation, the orphan key will be shifted to the front areas in the next level of the BKQ and will wait until the other session keys are completed and stored in the BKQ. After all session keys are stored in the BKQ, the group controller will assign new session key pairs based on their locations in the current BKQ. The key in the front of the BKQ is computed with the key in the rear of the BKQ, and the orphan key will be assigned with the key that is located at the rear of the BKQ at this time. Whenever an orphan key is present, it will be shifted to the front area of the next level of the BKQ and wait until a corresponding key is stored in the rear of the BKQ. After continuing this process, only two session keys will remain and the final GK will be computed.

The blinded keys in the rear half of the BKQ are regarded as the low-performance members' keys and these are not used to compute intermediate keys. After the low-performance members have provided their blinded keys to the high-performance members, the group

controller only allows the high-performance members who have blinded keys to continue to participate in the computation of the GK. Thus, only high-performance members are selected to participate in the GK generation process and the QGDH protocol avoids the unnecessary delays involved in waiting for the completion of other members' GKs. Furthermore, QGDH does not require the maintenance of a balanced key tree (as in TGDH).

There are two additional benefits of QGDH. First, since all members participate in generating a GK, that GK will be cryptographically stronger and more reliable than one generated in a protocol that uses only a single member's key. Second, all messages in our protocol are digitally signed by the sender's private key using Public Key Infrastructure (PKI). Certificates include the version, serial number, the algorithm ID, the name of the issuer, the name of the sender, (or, equivalently, the signer), the certificate signature algorithm (RSA), and a time stamp. Thus, a more trusted certificate and a more secure GK are additional advantages the QGDH protocol.

3.3. Membership Operations QGDH Supports

QGDH supports the following four operations: join, leave, partition, and merge.

Join Operation - Whenever a new member joins a group communication, the group controller broadcasts a control message to other members to generate a new blinded key and then store all the blinded keys from members in the BKQ in the order of arrival. The group controller determines high-performance members who will generate a session key in the next level of the GK generation process by checking the location of a blinded key in the BKQ. The high-performance members' blinded keys are always automatically stored in the front of the BKQ. The major steps in the join operation are shown in **Table 3**.

Table 3. Join Operation

Step	Description
Step 1:	When a new member joins, the group controller broadcasts a control message to all members.
Step 2:	Every member generates a blinded key and sends it back to the group controller.
Step 3:	The group controller • receives all blinded keys and stores them into the BKQ in the order of arrival • selects a next-level key pair for GK generation by matching high-performance members' and low-performance members' blinded keys • broadcasts to only members who are located at the front of the BKQ
Step 4:	Approved members compute a key pair and send it back to the group controller
Step 5:	The group controller • repeats Step 3 until the final GK has been generated • sends the final GK to all members when it has been generated

Leave Operation - As a member leaves the group, the GK must be recomputed. When an old member leaves, the group controller broadcasts a message to other members to generate a

blinded key and then receives all blinded keys into the BKQ. Major steps are shown in **Table 4**.

Table 4. Leave Operation

Step	Description
Step 1:	When an old member leaves, the group controller broadcasts a control message to other members to generate a blinded key for group key secrecy.
Step 2:	Every member generates a blinded key and sends it back to the group controller
Step 3:	The group controller • receives all blinded keys and stores them into the BKQ in order of arrival • selects a next-level key pair for GK generation by matching high-performance members' and low-performance members' blinded keys • broadcasts only to members who are located at the front of the BKQ
Step 4:	Approved members compute a key pair and send it back to the group controller
Step 5:	The group controller • repeats Step 3 until the final GK has been generated • sends the final GK to all members when it has been generated

Partition Operation - In contrast to the ETGDH, there is no subgroup controller in QGDH. A partitioned group is defined in QGDH as members that are disconnected from the group controller. The partitioned group will not be able to communicate with other partitioned group members because it does not have any subgroup controllers to reconnect them. The group controller uses polling to check members' networking status. If the group controller detects a member who is unreachable, the partition operation will be implemented until the disconnected member's status is resolved. In such a case, a GK must be regenerated, and the rest of the procedures are the same as join and leave operations. The major steps are shown in **Table 5**.

Table 5. Partition Operation

Step	Description
Step 1:	When a group controller detects unreachable members by using polling, the group controller broadcasts a control message to other members to generate a blinded key for group key secrecy.
Step 2:	Every member generates a blinded key and sends it back to the group controller
Step 3:	The group controller • receives all blinded keys and stores them into the BKQ in order of arrival • selects a key pair at the next level for GK generation by matching high-performance members' and low-performance members' blinded keys • broadcasts only to members who are located at the front of the BKQ
Step 4:	Approved members compute a key pair and send it back to the group controller
Step 5:	The group controller • repeats Step 3 until the final GK has been generated • sends the final GK to all members when it has been generated

Merge Operation - In QGDH neither a subgroup controller nor a partitioned group exists. Only unreachable members exist. Therefore, the merge operation is not necessary in QGDH. The merge operation is the same as the join operation, and the unreachable members must

rejoin the group to communicate with other members. The major steps are shown in **Table 6** (identical to **Table 3** which shows the Join Operation).

Table 6. Merge Operation

Step	Description
Step 1:	When a new member joins, the group controller broadcasts a control message to all members.
Step 2:	Every member generates a blinded key and sends it back to the group controller.
Step 3:	The group controller • receives all blinded keys and stores them into the BKQ in the order of arrival • selects a next-level key pair for GK generation by matching high-performance members' and low-performance members' blinded keys • broadcasts to only members who are located at the front of the BKQ
Step 4:	Approved members compute a key pair and send it back to the group controller
Step 5:	The group controller • repeats Step 3 until the final GK has been generated • sends the final GK to all members when it has been generated

Summary - the QGDH protocol provides an efficient method to determine high performance members. It does so by storing the blinded keys' in the BKQ in the order they are received.

4. Experiments and Performance Comparisons

This section demonstrates how low-performance group members negatively affect the overall performance of the GK generation process and how the QGDH protocol can address this problem. We show first, that the performance of group members' computer systems can affect the efficiency of GK generation. Then, we show that QGDH is more efficient than alternate approaches, including TGDH and ETGDH.

4.1. Comparing GK Protocols on Different Systems

To investigate the performance of group members' computer systems, four different machines with different operating systems (Windows XP, Linux, Unix, and Macintosh) were selected from a given network. The Windows XP machine is a Pentium 4 (2.8 GHz, 1 GB of RAM), the Linux (SuSE Linux 9.1) is a Pentium 4 (1.3 GHz, 480 MB of RAM), the Unix (Sun Solaris 9) is a SUN UltraSPARC (650 MHz, 2 GB RAM), and the Macintosh is a PowerPC G5 (Dual 2.7 GHz, 4 GB RAM). Recall that is not necessary for the group controller to actually detect the system resources in terms of processing power and RAM. Performance is measured solely in terms of the time it takes the system to respond to the group controller with the BK. The experimental results clearly show that the low-performance members' systems are slower at generating GKs than are the high-performance members' systems (see **Fig. 5** and **Table 7**). This indicates that GK generation processes can be improved by taking into account the performance of group members' computer systems.

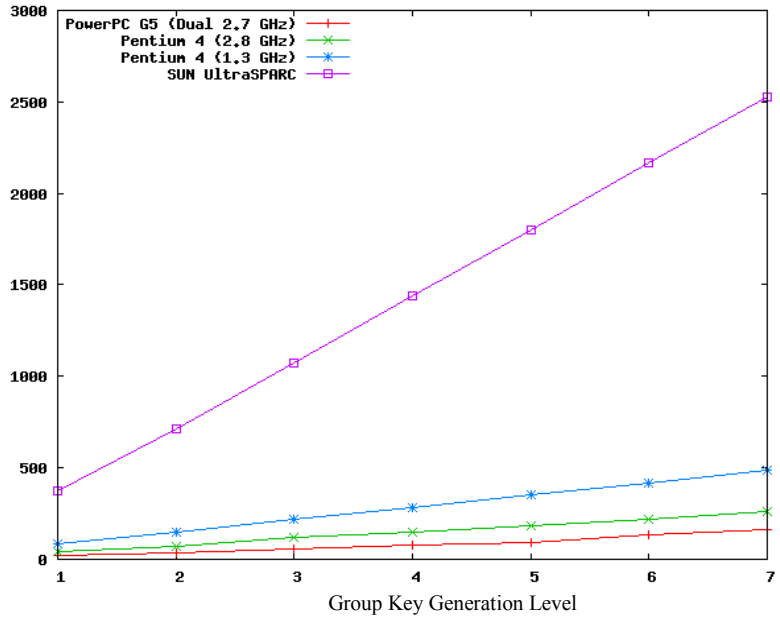

Fig. 5. Performance Analysis for Group Key Generation Process

Table 7. Average response times in each group key generation level (msec)

Operating Systems	Unix	Linux	Windows XP	Mac OS
CPU Clock Pulse	650 M Hz	1.3 G Hz	2.8 G Hz	Dual 2.7 GHz*
Level 1	375.3	83.4	39.1	21.4
Level 2	710.5	148.1	73.4	37.9
Level 3	1,074.2	215.7	117.1	57.4
Level 4	1,441.9	284.0	145.3	74.3
Level 5	1,797.7	349.9	181.2	93.3
Level 6	2,166.8	418.1	220.3	134.4
Level 7	2,527.5	487.0	262.5	163.7
Average	1,442.0	283.7	148.4	83.2

* According to SPEC CPU 2000, the performance of the dual 2.7 GHz CPU is 1.9 times faster than a single 3.0 GHz CPU. Thus, the computing power of the dual 2.7 GHz CPU is theoretically equal to the computing power of a single 5.2G Hz CPU.

To contrast QGDH with TGDH, ETGDH, and BD, we compare and plot the overall performance of the four protocols. Test results show to the average response times of generating the GK when a new member joins a group (including computation and communication overhead). The experiment consisted of the same four machines mentioned in the previous paragraph. At each level of GK generation (shown in **Table 7**), the QGDH protocol generates GKs in a shorter amount of time than either TGDH, ETGDH or BD (see **Fig. 6**). Recall that QGDH determines high-performance members at each level to optimize GK generation (and guard against delays caused by network faults, system failures, or other problems during the GK generation process). Thus, these results indicate that QGDH presents a way to generate GKs more efficiently than a process that does not account for the performance of group members' systems (TGDH). These results also indicate that QGDH

presents a way to generate GKs more efficiently than tree-based protocols such as ETGDH. Due to a high latency on generating BD's key as $O(n^2)$ [12], BD records the wrost performance among other protocols, and the difference between QGDH and ETGDH is not generated.

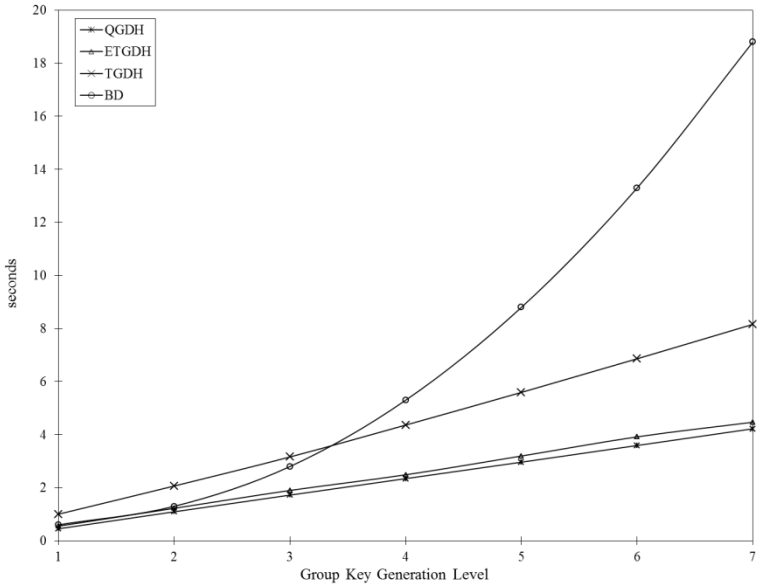

Fig. 6. Total Response Times for Generating a Group Key

4.2. Evaluating Membership Operations of Different GK Protocols

To verify that the proposed protocols are efficient, measurements of the time to complete the GK generation process were taken. The response times are measured by the times elapsed between invocation and completion. The overall performance of each machine is determined by the total response time (T_R) in terms of computing power and load (T_C), tree maintenance or queuing time (T_m), and network latency (T_N). T_C refers to the computing power which is a constant factor in terms of the capacity of the CPU, clock, main memory size, hard disk size, etc. T_m is the overhead involved in maintaining the key tree or the queuing structure. T_N is the time spent exchanging messages.

$$T_R = T_C + T_m + T_N \tag{4}$$

Communication and computation costs for each membership operation (join, leave, merge, and partition) are analyzed in terms of T_R. Other variables to be considered include the number of rounds and the communication costs. QGDH was compared to TGDH and ETGDH. The number of current groups, merging groups, leaving members, and partitions are denoted by n, m, k ($m \geq k$) and p, respectively. The height of the key tree constructed by the TGDH protocol is h. **Table 8** shows the communication and computation costs of these four operations [1].

TABLE 8. Communication and computation costs summary

Protocol		Communication		Computation	Properties
		Rounds	Messages	Exponentiations	
QGDH	Join	2	2n - 2	$3(\log_2 n) / 2$	QGDH has developed the disadvantage of ETGDH by using queue structure, which the key tree must be balanced. Otherwise, the overall performance wouldn't be $O(\log n)$
	Leave	1	2n - 2	$3(\log_2 n) / 2$	
	Partition	1	2n - 2	$3(\log_2 n)$	
	Merge	2	2n - 2	$3(\log_2 n) / 2$	
ETGDH	Join	2	2n - 2	3h / 2	ETGDH has been developed to overcome the TGDH's disadvantage which the overall performance depends on the slowest member's performance. ETGDH allows higher performance members who will be able to join in key generating process, so it can avoid unnecessary delay.
	Leave	1	2n - 2	3h / 2	
	Partition	1	2n - 2	3h	
	Merge	2	2n - 2	3h / 2	
BD	Join	2	2n + 2	3	The main idea in BD is to distribute the computation among members, such that each member performs only three exponentiations. This is achieved by using two communication rounds, each of them consisting of n broadcasts. BD requires $O(n^2)$ modular multiplications, and BD's group key is $K=g^{x_1x_2+x_2x_3+\ldots x_{n-1}x_n} \bmod p$ *(p is a exponential base)*
	Leave	2	2n - 2	3	
	Partition	2	2n + 2m	3	
	Merge	2	2n – 2p	3	
TGDH	Join	2	3	3h / 2	TGDH used a tree structure to reduce complexity of group key. However, every member needs to join in key generating process, so the overall performance depends on the slowest member's performance.
	Leave	1	1	3h / 2	
	Partition	min(log2 p, h)	2h	3h	
	Merge	$\log_2 k + 1$	2k	3h / 2	

BD [16] has n - 1 modular exponentiations with a small exponent. Unfortunately, n - 1 such exponentiations can be expensive when n is large. For example, BD requires $O(n^2)$ 1024-bit modular multiplications, if modular exponentiation is implemented with the square-and-multiply algorithm. BD is the most expensive protocol in terms of communication in leave. The cost order among others is determined strictly by the computation cost, since they all have the same communication cost (one round consisting of one message).

The rounds in **Table 8** correspond to how many broadcasts are used to send and receive blinded keys among each member. In ETGDH and QGDH as a new member joins, this new member must send his or her blinded key to all members (round 1). After sending this key, the new member needs to obtain all other members' blinded keys. Thus, all current members send their blinded keys back to the new member in parallel (round 2). Therefore, two rounds are required in the join and merge events. There is only one round in the leave event because a leaving member simply reports his or her leaving to all other members by using one broadcast.

5. Conclusion

The QGDH protocol, a GK generation protocol that uses a queue structure to determine high-performance group members and prioritize the generation of their GKs, offers a way to overcome the limitations of the TGDH protocol, a protocol that assumes that all members have equal computing power. In the QGDH model, the problem of a low-performance member possibly slowing down the GK generation process is mitigated. The QGDH protocol is more efficient than both the TGDH and ETGDH protocols across a number of computer systems and for groups of various sizes. Therefore, the QGDH protocol represents an improvement in

efficiency over existing approaches to secret key cryptography. This and other advances have the potential to reduce computational overhead and address the challenge of maximizing efficiency in secure group communication.

References

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

[2] Bresson, E., Chevassut, O. "Provably authenticated group Diffie-Hellman key exchange", In *Proc. of the 8th ACM CCS'01*, 2001. Article (CrossRef Link)

[3] Tripathi, S.; Biswas, G.P.; , "Design of efficient ternary-tree based group key agreement protocol for dynamic groups," *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International* , vol., no., pp.1-6, 5-10 Jan. 2009. Article (CrossRef Link)

[4] Choie, Y., Jeong, E., and Lee, E. "Efficient identity-based authenticated key agreement protocol from pairings," Applied Mathematics and Computation, vol. 162, no. 1, pp. 179-188, 2005. Article (CrossRef Link).

[5] Kaufman, C., Perlman, R, Speciner, M., "Network security: private communication in a public world, second edition," *Prentice Hall Press Upper Saddle River, 2002*. Article (CrossRef Link)

[6] Diffie W. and Hellman, M. E. "New directions in cryptography," IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644-654, 1976. Article (CrossRef Link).

[7] Fekete, A., Lynch, N., and Shvartsman, A. "Specifying and using a partitionable group communication service," *In ACM PODC '97*, 1997. Article (CrossRef Link)

[8] Hong, S. and Lopez-Benitez, N. "Enhanced Group Key Generation Algorithm," 10th IEEE/IFIP Network Operations and Management Symposium, 1-4, 2006. Article (CrossRef Link).

[9] Katz, J. and Yung, M. "Scalable Protocols for Authenticated Group Key Exchange," Journal of Cryptology, vol. 20, no. 1, pp. 85-113, 2006. Article (CrossRef Link).

[10] Kim, Y. Group key agreement: theory and practice, *Ph.D. dissertation*, 2002. Article (CrossRef Link)

[11] Kim, Y.; Perrig, A.; Tsudik, G.; , "Group key agreement efficient in communication," *Computers, IEEE Transactions on* , vol.53, no.7, pp. 905- 921, July 2004. Article (CrossRef Link)

[12] Kim, Y., Perrig, A., and Tsudik, G. "Tree-based group key agreement," ACM Transaction on Information and System Security, 2004. Article (CrossRef Link).

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

[14] Steiner, M., Tsudik, G., and Waidner, M. "Key agreement in dynamic peer groups," IEEE Transactions on Parallel and Distributed Systems, vol. 11, no. 8, pp. 769-780, 2000. Article (CrossRef Link).

[15] Wong, C., Gouda, M., and Lam S. Secure group communications using key graphs, IEEE / ACM Transactions on Networking, vol. 8, no. 1, pp. 16-30, 2000. Article (CrossRef Link).

[16] Mike Burmester and Yvo Desmedt, "A secure and efficient conference key distribution system", Advances in Cryptology – EUROCRYPT '94, number 950 in Lecture Notes in Computer Science. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1995. Article (CrossRef Link)

Sunghyuck Hong received the Ph.D. degree from Texas Tech University in August, 2007 major in Computer Science. Currently, he works at Baekseok University as an assistant professor. He is a member of editorial board in the Journal of Korean Society for Internet Information (KSII) Transactions on Internet and Information Systems. His current research interests include Secure Cloud Computing, Secure Wireless Sensor Networks, Key Management, and Networks Security

Sungjin Lee received the Ph.D. degree from Korea Advanced Institute of Science and Technology (KAIST) in August, 2000 major in information and communication. He is an advisor of supreme public prosecutor's Office digital investigation and advisory board of the National Institute of Scientific Investigation. Currently, he works at Baekseok University as an associate professor. His research area is digital forensic.