큐 구조를 이용한 효율적인 그룹 동의 방식

홍성혁*

Efficient Group Key Agreement Protocol (EGKAP) using Queue Structure

Sunghyuck Hong*

요 약 그룹 통신은 인터넷을 통해 급속도로 발전 중이며, 그룹통신으로는 화상회의, 인스턴트 메시지, 데스크 탑 공유, 다양한 e-커머스등과 같은 응용프로그램들이 있다. 안전한 그룹통신을 위해서 그룹 키를 생성하여 통신 메시지를 암호화하여 교환함으로써 그룹통신의 기밀성을 보장할 수 있다. 본 연구에서는 그룹 키를 보다 안전하면서 효율적으로 생성하여 안전한 그룹통신에 기여하는데 그 목적이 있다.

주제어 : 키관리, 키동의, 안전한 그룹통신, 네트워크 보안, 안전한 그룹키

Abstract Group communication on the Internet is exploding in popularity. Video conferencing, 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.

Key Words : key management, key agreement, secure group communication, network security, secure group key

1. Introduction

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 [3]. To achieve a high level of security, the GK should be changed after any member joins or leaves so that former group members have no access to current communications and new members have no access to previous communications [2]. 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. To address this problem, a recent focus in key management is the efficient generation of GKs [1][8][9]. It is this need for efficient key generation that we address.

I describe a new approach to GK generation, the Efficient Group Key Agreement Protocol (EGKAP) with using Queue structure. EGKAP provides a queue-based divide and conquer algorithm that is more efficient than the Tree-based Group Diffie-Hellman (TGDH) protocol that is currently the most efficient group key generation protocol [9]. I describe the EGKAP protocol in detail below. Then, to demonstrate how the EGKAP provides an efficient way to determine high-performance members without additional computational overhead, I contrast its' efficiency with the TGDH protocol in several experimental tasks. Our

*백석대학교 정보통신학부 조교수
논문접수: 2012년 4월 30일, 1차 수정을 거쳐, 심사완료: 2012년 5월 11일

results show that the EGKAP is both feasible and computationally efficient.

2. Group Key Generation Protocols

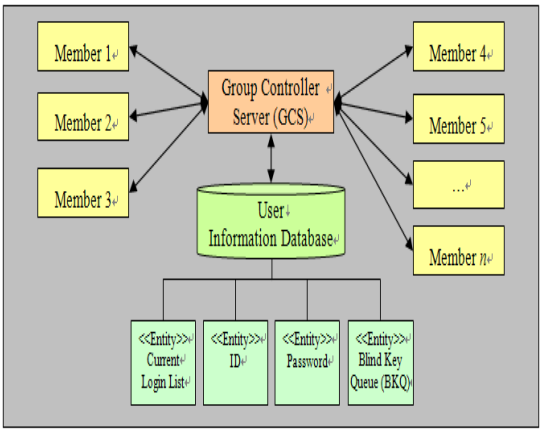
The TGDH (Tree-based Group Diffie-Hellman) protocol is currently the most efficient group key generation protocol [2]. This protocol reduces the complexity of generating a GK by using a tree structure during generation. The TGDH protocol has two shortcomings, however. One shortcoming is that TGDH does 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 group key generation. For maximum performance, a tree must be well-balanced. Because group members leave the group key 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 [2].

2.1 Group Diffie-Hellman Protocol

The EGKAP queue structure is used to determine high- and low-performance members, to prioritize the generation of GKs for high-performance users, and then to generate GKs in that order. In EGKAP, a Group Controller Server (GCS) is utilized, requiring all members to compute their public keys and store them in a queue structure on the GCS in the order of arrival. EGKAP, as its name indicates, uses a queue structure rather than a tree structure. Keys that are computed and transmitted quickly indicate a high-performance member; these keys are automatically stored at the front of the queue. This arrangement is advantageous because the first element in the queue (from the highest-performance member) will be the first one processed, thus avoiding delays in generating a GK. An additional benefit of the queue structure can be seen when comparing it with a tree structure. While queues have a simple linear data structure and can be changed when group members join or leave with minimal computational overheard, trees must maintain a balanced structure to maintain maximum performance. Thus, the biggest advantages of a queue structure over a tree structure are that a queue structure provides a useful way to determine high-performance members and that it avoids the additional overhead to maintain balance in a key tree.

The queue structure is a centralized control to generate a GK by using a GCS. Figure 1 shows a proposed EGKAP entity model. A GCS can access a member information database that contains a current login member list, IDs, Passwords, and a Blind Key Queue (BKQ). Upon login, the GCS validates the member's ID and password by accessing the member information database. After validating a member's identification, all members start to generate a GK by sending their blind key to the GCS. The GCS collects all blind keys and stores them in the BKQ in the order of their arrival. As noted earlier, 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). Performance is measured in terms of the time it takes a member's system to respond to the GCS with their BK.



[Figure 1] EGKAP Entity Model

Current group communication protocols use a self-signed certificate for member authentication, which has a well-known weakness in that members cannot ensure that the name on the public key is really a true member's name [5]. To compensate for this well-known weakness, our approach uses a GCS as a complete trusted party. Our threat model takes into account both passive and active outsiders (i.e. individuals who are not group members). Passive outsider attacks involve eavesdropping with the aim of discovering the GK(s); active outsider attacks involve injecting, deleting, delaying, and modifying protocol messages.

As group membership changes, the GCS determines who will continue to participate in generating a group key (but the GCS does not participate in generating the group key). Authorized members generate a GK that is distributed to all other members using an encrypted link using RSA. Since the GCS does not generate a key by itself to be sent to each member, the computational and communication costs of generating a new group key will not be incurred whenever membership changes.

2.2 How EGKAP Works

There are two overhead costs in group key management protocol: the first is communication cost, and the second is computation cost. Computation and communication costs are important because they impact the scalability of group key management [11]. Communication cost is relatively small and constant (often using a 3-round protocol [1]). Computation costs, in contrast, dominate the performance of GK management [11] 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 [7]. Therefore, EGKAP focuses on reducing computation costs.

The EGKAP protocol works in the following way. The GCS broadcasts a request to all members to generate a blind key. The GCS receives all blind keys and stores them into its' BKQ in the order of their arrival, with the higher-performance members' blind keys stored in the front of the BKQ and the lower-performance members' blind keys stored in the rear of the BKQ. The GCS requests members who are in the front half of the BKQ to compute their Diffie-Hellman key exchange with those blind keys and store them in the next level of the BKQ in order of arrival. Following the determination of each level, the GCS collects and stores all computed session keys in the BKQ. 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 BKQ automatically assigns each pair of keys. For example, the first spot's blind key will be computed with the last spot's blind key; the second spot's blind key will be computed with the second-to-last spot's blind key, and so on.

The blind 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 blind keys to the high-performance members, the GCS only allows the high-performance members who have blind 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 EGKAP protocol avoids the unnecessary delays involved in waiting for the completion of other members' GKs. Furthermore, EGKAP does not require the maintenance of a balanced key tree (as in TGDH).

2.3 Membership Operations EGKAP Supports

EGKAP supports the following two operations: join and leave

Whenever a new member joins a group communication, the GCS broadcasts a control message to other members to generate a new blind key and then store all the blind keys from members in the BKQ in the order of arrival. The GCS determines high-performance members who will generate a session key in the next level of the group key generation process by checking the location of a blind key in the BKQ. The high-performance members' blind keys are always automatically stored in the front of the BKQ. The major steps are shown in Table 2.

〈Table 1〉 Leave Protocol

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

〈Table 2〉 Join Protocol

Step 1:	When a new member joins, the GCS broadcasts a control message to all members.
Step 2:	Every member generates a blind key and sends it back to the GCS.
Step 3:	The GCS ● receives all blind 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' blind 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 GCS
Step 5:	The GCS ● repeats Step 3 until the final GK has been generated ● sends the final GK to all members when it has been generated

Leave Protocol - As a member leaves the group, the group key must be recomputed. When an old member leaves, the GCS broadcasts a message to other members to generate a blind key and then receives all blind keys into the BKQ. Major steps are shown in Table 1.

2.4 Testing EGKAP against Other Protocols

This section demonstrates how low-performance group members negatively affect the overall performance of the GK generation process and how the EGKAP protocol can address this problem. I show first, that the performance of group members' computer systems can affect the efficiency of group key generation. Then, we show that EGKAP is more efficient than alternate approaches, including TGDH and a recent variation of TGDH, known as the Enhanced Tree-Based Group Diffie-Hellman protocol (ETGDH). This performance analyses focus on the number of rounds, the total number of control messages, network overheads, and group key generation costs. The total cost is the sum of all

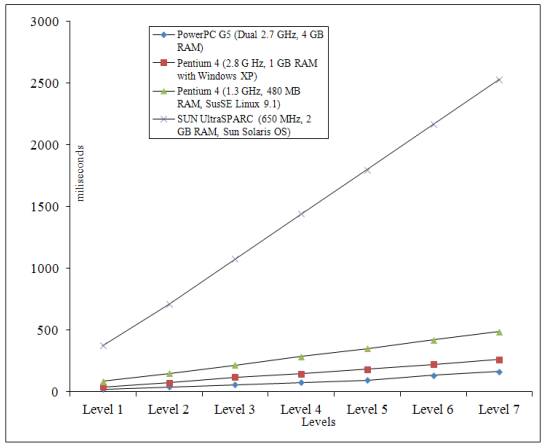
participants' costs(elapsed times) incurred by any participant in a given round or protocols on NS 2 which is simulator tool.

〈Table 3〉 Average Response Times in Each Group Key Generation Level (unit: msec)

OS	Unix	Linux	Windows XP	MAC OS
CPU Clock	650 MHz	1.3 GHz	2.8 GHz	5.2 G Hz
Level 1	375.3	83.4	39.1	21.4
Level 2	710.5	148.1	73.4	37.9
Level 3	1,0742	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

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 GCS 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 GCS 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 Figure 2 and Table 3). This indicates that GK generation processes can be improved by taking into account the performance of group members' computer systems. 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 EGKAP with TGDH and ETGDH, we compare and plot the overall performance of three protocols. At each level, the EGKAP protocol generates GKs in a shorter amount of time than either TGDH or ETGDH (see Figure 3).



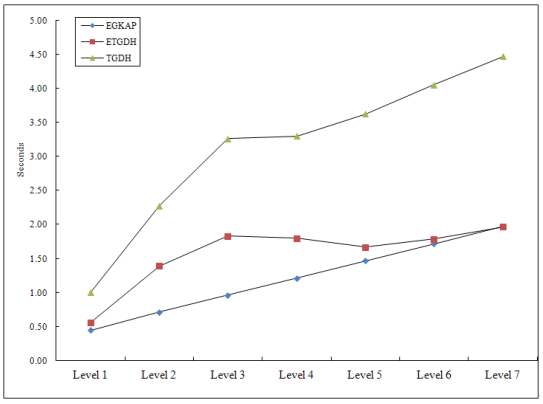
[Figure 2] Performance Analysis for Group Key Generation Process

Note that EGKAP determines high-performance members at each level to optimize GK generation (and guard against delays caused by network faults, system failures, or other issues during the GK generation process). Test results correspond 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. Thus, these results indicate that EGKAP presents a way to generate GKs more efficiently than a process that does not account for the performance of group members' systems.

3. Conclusion

A key is a building block to establish a secure group communication, and key must be generated efficiently and securely. Therefore, efficient and secure key generation protocol should be used for secure group communication. The EGKAP 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 continue to bring attention to the issue of efficient, secure group communication.



[Figure 3] Total Response Times for Generating a Group Key

Reference

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

[2] C. Wong, M. Gouda, and S. Lam, (2000), Secure group communications using key graphs, IEEE / ACM Transactions on Networking, 8(1).

[3] M. Fratto, IN PKI WE TRUST? (2001) - VeriSign topped a trio of outsourced PKI solutions for our fictional company. In real life, you have to do your homework and match your needs with the services' realities. Network Computing, 69.

[4] Y. Choie, E. Jeong and E. Lee,(2005), Efficient identity-based authenticated key agreement protocol from pairings, Applied Mathematics and Computation,162(1), 179-188.

[5] Y. Kim, A. Perrig, and G. Tsudik. (2001), Communication-efficient group key agreement, In 17th International Information Security Conference (IFIP SEC'01).

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

홍 성 혁

Sunghyuck Hong received the Ph.D. degree from Texas Tech University in August, 2007 major in Computer Science. Currently, he works at Division of Information and Communication in Baekseok University as an assistant professor. Before he joined Baekseok, he worked at International affairs in Texas Tech University as a senior programmer/analyst, and his jobs were development of ASP.NET web applications and maintenance of PC/Server. 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 Wireless Sensor Networks, Key Management, and Networks Security.

· E-Mail:shong@bu.ac.kr