A Thesis for the Degree of Master

A Study on Secure Group Mutual Exclusion Algorithm

Jaehyrk Park

School of Engineering

Information and Communications University

2004

A Study on Secure Group Mutual Exclusion Algorithm

A Study on Secure Group Mutual Exclusion Algorithm

Advisor : Professor Kwangjo Kim

by

Jaehyrk Park
School of Engineering
Information and Communications University

A thesis submitted to the faculty of Information and Communications University in partial fulfillment of the requirements for the degree of Master of Science in the School of Engineering

Daejeon, Korea
Dec. 29. 2003
Approved by

(signed)

Professor Kwangjo Kim
Major Advisor

A Study on Secure Group Mutual Exclusion Algorithm

Jaehyrk Park

We certify that this work has passed the scholastic standards required by the Information and Communications University as a thesis for the degree of Master

Dec. 29. 2003

Approved:

Chairman of the Committee
Kwangjo Kim, Professor
School of Engineering

Committee Member
Jae Choon Cha, Assistant Professor
School of Engineering

Committee Member
C.Pandu Rangan, Invited Professor
School of Engineering

M.S. Jaehyrk Park
20022061

A Study on Secure Group Mutual Exclusion Algorithm

School of Engineering, 2004, 47p.
Major Advisor : Prof. Kwangjo Kim.
Text in English

Abstract

A distributed system can be viewed as a set of processes that share many types of resources, such as processors, memory cells, buses, and printers. Emerging network technologies require efficient distributed processing ability. A distributed algorithm in distributed system deals with how to make the computers connected together work well. Main focus of distributed algorithm is mutual exclusion problem that means many shared resources must be accessed in a mutually exclusive manner.

This thesis gives a new quorum-based distributed group mutual exclusion algorithm. In the group mutual exclusion problem, multiple processes can enter critical section at the same time if they belong to the same group. The former quorum-based group mutual exclusion algorithm has a case when two processes cannot enter critical section at the same time even if they can do so. We call the above situation as unnecessary blocking. We give a new algorithm which prevents unnecessary blocking. Also, in this thesis, we present a method to control the access to a secure database based on group mutual exclusion algorithm using quorum systems. The security of algorithm is based on the secret sharing scheme. The main goal of our algorithm is to integrate security with distributed algorithm.

Contents

List of Tables

List of Figures

List of Abbreviations

Algo. Algorithm

CA Certification Authority

DB Database

PKI Public Key Infrastructure

SSS Secret Sharing Scheme

SDR System of distinct representatives

List of Notations

AS Access Server

CS Critical Section

DS Data Server

G set of group

K set of process

p One requesting process

q One access server process

Q One quorum that is a set of processes

SK Secret Key

T set of partial secret value

U set of access server processes, $U = \{q_1, q_2, ..., q_n\}$

V set of requesting processes, $V = \{p_1, p_2, ..., p_n\}$

X data part that user want

I. Introduction

1.1 Distributed Algorithm and Security

A distributed system consists of a collection of computers, called sites, that are geographically distributed and connected by a communication network. Examples of distributed systems include distributed inventory control systems, banking systems, airline reservation systems, and campus-wide file systems. In other words, a distributed system can be viewed as a set of processes that share many types of resources, such as processors, memory cells, buses, and printers. However, under the distributed system environment, there is no shared memory for each process. Therefore, in order to achieve cooperative tasks, processes must communicate with other processes via communication links.

Distributed algorithm in distributed system deals with how to make the computers connected together work well. Designing distributed algorithms to control distributed systems is by no means easy since there is no way to capture the global state of the system. So, computers must send and receive messages to other computers to get enough information to do their tasks. However, emerging network technology requires efficient distributed processing ability, which is related to grid computing or distributed algorithm area.

The main focus on distributed algorithm is mutual exclusion problem that means many shared resources must be accessed in a mutually exclusive manner. In other words, mutual exclusion is crucial for the design of distributed system. Many problems involving replicated data, distributed shared memory, and others require that a resource be allocated to a single process at a time. However, solutions to these problems

are often vulnerable to site and communication failures. Intersecting quorums can be used to provide fault-tolerant solutions to mutual exclusion problems. A lot of researches on distributed algorithm with fault-tolerance can be found in the literature.

Both fault-tolerance and security field try to stop bad effects from spreading out throughout the system and corrupting the system. These bad effects are obviously different in the two cases: fault tolerance is looking at just trying to contain crashes, whereas security is looking towards trying to stop bad people from doing bad things. However, in a distributed system, several replicas may be maintained at different sites to improve fault tolerance. Maintaining replicas may also affect the integrity and secrecy of the data. Thus, it is natural to consider security issues together with fault tolerance.

1.2 Our Contributions

In this thesis, we present a quorum based efficient distributed algorithm. We also propose a secure distributed algorithm through adapting cryptographic primitive to our new algorithm. We summarize our contributions as follows:

Quorum-based distributed algorithm : We give a new quorum-based distributed group mutual exclusion algorithm. In the group mutual exclusion problem, multiple processes can enter critical section at the same time if they belong to the same group. The former quorum-based group mutual exclusion algorithm has a case when two processes cannot enter critical section at the same time even if they can do so. We call the above situation as unnecessary blocking. We give a new algorithm which prevents unnecessary blocking and prove its correctness.

Secure Quorum-based distributed algorithm : Usually, researches on distributed algorithm don't deal with security aspects except for fault-tolerance since their goal is consistency with competing processes in distributed system and low complexity. Approaches on designing secure distributed algorithm can be valuable. We specify a secure distributed algorithm by using both group mutual exclusion algorithm and secret sharing scheme. Moreover, even though we add cryptographic technique to distributed algorithm, communication complexity is not changed. Our algorithm not only guarantees the requirements of mutual exclusion algorithm but also users' privacy and authentication.

1.3 Organization

The organization of this thesis is as follows:

- Chapter II: Basic concepts that are fundamentally used for the construction of our distributed algorithm are briefly introduced. The goal is to unify these concepts into rationale of our algorithm designing.
- Chapter III: Problem in former group mutual exclusion algorithm is introduced. We propose new algorithm without unnecessary blocking problem and show correctness proof. Then, secure distributed algorithm is presented by using SSS.
- Chapter IV: To analyze our algorithm, security and performance comparisons are presented.
- Chapter V: A summary of the results and open problems are presented.

II. Preliminaries

In this chapter, we introduce basic concepts which would be employed in our algorithm and were presented in previous related works.

2.1 Background

We first present the fundamental concept for understanding our new distributed algorithm. In order to make efficient distributed algorithm, Mutual exclusion and group mutual exclusion problem should be solved. In this chapter, we describe major concept and techniques frequently used in designing secure distributed algorithm.

2.1.1 Distributed Systems

A set of computers connected by a set of communication links is called a distributed system. We characterize distributed systems by the absence of shared memory. In a distributed system, processes on a computer do their tasks with other processes on remote computers. To achieve co-operative tasks, processes must communicate with other processes via communication links since there is no shared memory. The following motivates distributed systems.

High Performance Since the system consists of several computers, independent tasks can be processed in parallel. Load balancing is easy.

Distribution of users When users of the system are geometrically distributed, it is natural to process tasks distributedly.

Extensiveness In general, addition of computers and communication links can be done easily with small modification of the current system. Replacement of computers and communication links is also easy. This property naturally comes from the fact that distributed systems are loosely coupled.

Fault-tolerance A centralized system cannot provide services when the central machine stops by failure. Distributed systems may provide services if there are several alive components.

Distributed systems have many advantages compared with centralized systems. However, designing distributed algorithms to control distributed systems is by no means easy because of the following reasons: Computers must send/receive messages to other computers to get enough information to do their tasks. Messages are delivered with delay and therefore in principle there is no way to capture the global state of the system. In addition, there is no process which controls the entire distributed system. Therefore, to achieve fault-tolerance, algorithms must consider failures such as process stops and message loss.

2.1.2 Mutual Exclusion Problem

Mutual exclusion is one of the fundamental problems in distributed systems. The mutual exclusion problem first arised when the concept of concurrent processes were introduced in operating systems. When more than one process shares memory cells, undesirable situations may happen: Suppose that two processes P_1 and P_2 that share a variable, say x, wish to increment x by one. To increase the value of x, a process loads the value of x into a register in CPU, increase the value of the register by one, and then stores it back into x. If P_2 starts executing

the above procedure after P_1 finishes its execution, the result is correct, *i.e.*, the value of x is incremented by two. However, what if their executions are interleaved? Consider, for example, the following interleaved execution sequence. P_1 loads x, P_1 increments the register, P_2 loads x, P_2 increments the register, P_2 stores the register into x, and then P_1 stores the register into x. x is incremented by only one. To guarantee such an undesirable situation does not happen, the concept of critical section is introduced. A program text can be partitioned into two kinds of sections : sections in which there are no accesses to shared resources (*e.g.*, shared variables) and sections in which shared resources are accessed. The latter sections are called critical sections or critical regions. When some resource (for example, a file, a communication channel, a printer) is shared among processes, no two processes are allowed enter the critical section(CS) to use it at the same time. To enter a CS, a process must assure that there is no process which is being in CS in the distributed system. Many algorithms have been proposed to solve the distributed mutual exclusion problem. Also, such algorithms follow one of these three types.

Permission based principle A process P wishing to enter a CS requests some other processes' permission. If the permission is given from the process P is asking, P can then enter the CS.

Token based principle There is an object called a token in a distributed system and it travels among processes. A process can enter a CS while it is holding the token. The mutual exclusion is guaranteed because there is only one token in the system and there are no two processes having a token at the same time.

Quorum based principle A process can enter a CS when it receives

permission from a set of processes. The set of processes is called a quorum.

In this thesis, we use quorum based principle to solve mutual exclusion problem.

2.1.3 Quorum System

Quorum system is a basic tool providing a uniform and reliable way to achieve coordination in a distributed system. They are useful for distributed and replicated databases, name servers, mutual exclusion, and distributed access control and signatures. Quorum systems have important intersection property. Set systems with the intersection property are known as *quorum systems*, and the sets in such a system are called quorums.

A protocol template based on quorum systems works as follows. In order to perform some action (update the database, say), the user selects a quorum and accesses all its elements. The intersection property then guarantees that the user will have a consistent view of the current state of the system.

For example, if all the members of a certain quorum give the user permission to enter the critical section, then any other user trying to enter the critical section before the first user has exited (and released the permission-granting quorum from its lock) will be refused permission by at least one member of any quorum it chooses to access.

2.1.4 Group Mutual Exclusion Problem

Recently, the concept of group mutual exclusion [39] has been proposed. There are multiple groups of processes. The processes in the same group can enter CS at the same time. This problem corresponds to the follow-

ing situation. There is a CD jukebox and each process wants to read some data on the CDs. If CD A is loaded, multiple processes which want to read data on the CD A can access it at the same time. These processes are in the same group. On the other hand, the processes which want to read data on CD B cannot do so when A is loaded. These processes forms a different group. In addition, the same data might be copied on multiple CDs. Thus, some processes might be a member of multiple groups.

2.1.5 Secret Sharing

Secret sharing was originally suggested for threshold access structures by Shamir and Blakley [30, 1]. The idea of secret sharing is to start with a secret, and divides it into pieces called shares which are distributed among users such that the pooled shares of specific subsets of users allow reconstruction of the original secret. It was extended to arbitrary access structures in [10]. In other words, The idea of a threshold scheme may be broadened to a generalized secret sharing scheme as follows: Given a set P of uses, Λ (the access structure) to be a set of subsets, called the authorized subsets of P. Shares are computed and distributed such that the pooling of shares corresponding to any authorized subset $A \in \Lambda$ allows recovery of the secret S, but the pooling of shares corresponding to any unauthorized subset $B \subseteq P$,$B \notin \Lambda$ does not. In this thesis, we can apply any secret sharing to build our new secure distributed algorithm. However, we use secret sharing scheme realizing the access structures of quorum systems by presented [24]. Definition of their scheme is as follows:

Definition 2.1.1 (Secret sharing) *Let $U = \{1, ..., n\}$ and let S be a finite set of secrets. A secret-sharing scheme is a mapping $\Pi : S \times R \rightarrow$*

$S_1 \times S_2 \times \cdots \times S_n$, *where* R *is a set of random strings, and for each* $i \in U, S_i$ *is a set of secret shares.* Π *is said to realize an access structure* Λ *if it satisfies the following conditions:*

1. *The secret can be reconstructed by any subset in* Λ*. That is, associated with every set* $A \in \Lambda$ $(A = \{i_1, ..., i_{|A|}\})$ *there is a function* $h_A : S_{i_1} \times \cdots \times S_{i_{|A|}} \rightarrow S$ *such that for every* $(s, r) \in S \times R$*,if* $\Pi(s, r) = \{s_1, ..., s_n\}$ *then* $h_A(s_{i_1}, ..., s_{i_{|A|}}) = s$*.*

2. *No subset, unless it is a member of* Λ*, can reveal any partial information about the secret (in the information theoretic sense). Formally, for any subset* $Z \nsubseteq \Lambda$*, for every two secrets* $a, b \in S$*, and for every possible collection of shares* $\{s_i\}$ *where* $i \in Z$

$$P(\{s_i\}_{i \in Z}|a) = P(\{s_i\}_{i \in Z}|b),$$

where the probability is taken over the random string r*.*

2.2 Related works

2.2.1 Distributed Algorithm

Mutual exclusion is one of the fundamental problems in distributed systems. The first distributed mutual exclusion algorithm is proposed by Lamport [16]. To guarantee mutual exclusion, no deadlock, and no starvation, distributed mutual exclusion algorithms must have some arbitration mechanism. To this end, he proposed a logical clock.

After that, many distributed mutual exclusion algorithms have been proposed [34, 33]. Quorum-based algorithm [18] is one of the solutions with fault-tolerance. Some extensions, such as there are multiple units of the same resource [13, 22] and the set of resources each process can

access differs from process to process [14, 36], have been discussed. Recently, group mutual exclusion [39] has been proposed. There are multiple groups of processes. The processes in the same group can enter CS at the same time. For group mutual exclusion, shared memory system algorithm [7, 15], a token-based algorithm [4], and a quorum-based algorithm [40] have been proposed. We found problem of [40] and such problem leads a poor efficiency. This thesis proposes a new algorithm which avoids this problem.

2.2.2 Quorum-based Secure Protocol

A quorum system is a collection of sets (quorums) every two of which have a nonempty intersection. Quorum systems have been used in the study of distributed control and management problems such as mutual exclusion [18, 28], data replication protocols [5, 11], name servers [19] and selective dissemination of information [41].

Herlihy and Tygar [8] have suggested a scheme to protect quorum based replicated databases. However, their scheme has several drawbacks: access is controlled over the whole database and not at record granularity, there is no way to revoke a user's access once it is obtained, and the scheme uses a k-of-n threshold access structure, which implies a large communication overhead and high load since the threshold must be $k > n/2$ for the structure to be a quorum system.

Reiter and Birman [29] considered a database protection scheme against servers being corrupted. In their scheme it is the responsibility of the users to verify that the data sent by the servers is genuine. They rely on a k-of-n threshold scheme heavily, and do not separate between the data servers and the access servers.

Beaver and Wool [3] have suggested a secure multi-party protocol which respects any arbitrary quorum system. Also, they showed several

multi-party protocols which are based on specific quorum systems.

This proposed schemes use quorum system under secure protocol based on cryptographic primitives. Our algorithm adapt security to quorum based distributed algorithm.

III. Proposed Scheme

3.1 Group Mutual Exclusion Algorithm without Unnecessary Blocking

The distributed system consists of processes and channels. The processes are asynchronous and fail by fail-stop model. The process failure can be detected. The communication between processes are done by message passing through FIFO(First-In, First-Out), asynchronous, and reliable(no message loss occurs) channels.

This thesis assumes that the processes are divided into requesting processes and access server processes for the simplicity of discussion. The access server processes manages mutual exclusion and the requesting processes just make requests to enter CS. In actual systems, one process can act as a access server process and a requesting process at the same time. This thesis' discussion can also be applied to such systems. Let us denote $U = \{q_1, q_2, ..., q_n\}$ be the set of access server processes and $V = \{p_1, p_2, ..., p_m\}$ be the set of requesting processes.

Ç $= \{g_1, g_2, ..., g_l\}$ is the set of groups, $g_i \subseteq V$ and $g_i \neq \phi$ for every $i(1 \leq i \leq l)$. The processes in g_i can use the shared resource at the same time. Each process belongs to at least one group in Ç. The set of groups p_i belongs to is called as p_i's group set and denoted as $G(p_i)$. That is, $G(p_i) = \{g \in \text{Ç} | p_i \in g\}$. Note that $G(p_i) \neq \phi$.

According to the example of CD jukebox, the following is assumed about group selection, which is not explicitly stated but written in the algorithm in [40]. When process p_i enters CS, it selects one group $g \in G(p_i)$, which corresponds to the selection of CD. This group is called

as p_i's group selection and denoted as $gs(p_i)$. Though this assumption is not explicitly stated, the definition of group mutual exclusion is as follows:

Definition 3.1.1 *If two conditions are satisfied, we call it Group Mutual Exclusion.*

mutual exclusion Processes p_i and p_j cannot be in CS at the same time if $gs(p_i) \neq gs(p_j)$.

starvation freedom Every process which wants to enter CS must be eventually able to do so.

3.1.1 Problem in Former Algorithm

This section shows the outline of algorithm in [40] and its unnecessary blocking. M-group quorum system $C = \{C_1, .., C_m\}$ is defined as follows. C_i is set of quorums, where each quorum $Q \in C_i (Q \subseteq U, Q \neq \phi)$ satisfies the following two properties.

Intersection Property $\forall\, 1 \leq\ i, j \leq\ m, i \neq\ j, \forall\, Q_1 \in\ C_i, \forall\, Q_2 \in\ C_j \mapsto\ Q_1 \bigcap\ Q_2\ \neq\ \phi$

Minimality $\forall 1 \leq i \leq m, \forall\, Q_1, Q_2 \in\ C_i, Q_1 \neq\ Q_2 \mapsto Q_1 \nsubseteq Q_2$

Each access server process has one vote to send "Ok". It can send "Ok" to requesting processes in at most one group at the same time. The intersection property means that for any two requesting processes in different groups, the quorums intersects, thus these two processes cannot enter CS at the same time, because of the above access server processes' rule of sending "Ok". The outline of their mutual exclusion algorithm is as follows:

When requesting process p wants to enter CS

1. p selects one group g_i from $G(p)$, selects one quorum Q from C_i, and sends "$Request(g_i)$" to every member of Q.

2. When p receives "Ok" from every member in Q, p enters CS.

When access server process q receives "$Request(g_i)$" from p
q sends "Ok" to p if

1. q sends no "Ok" to any other processes, or

2. q has sent "Ok" to a request p' such that $gs(p') = g_i$.

The actual algorithm is more complicated to avoid starvation and achieve efficiency. The above algorithm has problem which leads unnecessary blocking. Consider the following example. p_1,whose group set $G(p_1) = \{g_1\}$ sends request and receives "Ok" from every member in $Q \in C_1$. p_2, whose group set $G(p_2) = \{g_1, g_2\}$, then appears. The algorithm requires p_2 to select one group from $G(p_2)$ before making request. Suppose that p_2 selects g_2 as $gs(p_2)$. Then, p_2 cannot enter CS because $gs(p_2) \neq g_1$. This blocking is unnecessary because p_2 could enter CS if p_2 would set g_1 as $gs(p_2)$.

This unnecessary blocking comes from the condition that p_2 must set $gs(p_2)$ when there is no information about the other requests. If p_2 can set $gs(p_2)$ after current status is obtained (for example, some process whose group selection is g_1 is currently entering CS), p_2 can set a better group as $gs(p_2)$ and this type of unnecessary blocking is avoided.

In the next section, we show the outline of our algorithm to avoid this unnecessary blocking.

3.1.2 Avoiding unnecessary blocking

In order to avoid bad group selection, each requesting process must be able to set its group selection after it receives some replies from access

server processes. We introduce two-phase mutual exclusion algorithm. It was firstly used in [14] to solve the generalized mutual exclusion problem and then improved in [35]. In the generalized mutual exclusion problem, there are multiple shared resources and each process may have different accessible resources. In the two-phase algorithm, each requesting process makes its decision after it receives "Ok" from every process in a quorum. It then informs its decision to the processes in the quorum. In the generalized mutual exclusion, the decision is which resource it uses. In the group mutual exclusion, the decision is which group it selects as $gs(p)$.

Firstly, requesting process p sends "$Request$" to the processes in q quorum Q before it sets $gs(p)$. Each process q in Q, which received the request, replies "Ok" or the information that q has sent "Ok" to another process. Using the replies, p enters CS if (1) every process in Q replies "Ok" or (2) some process in Q reports that some process p' in entering CS and $gs(p') \in G(p)$.

In case (1), p can set any group in $G(p)$ as $gs(p)$, since there is no other processes which blocks p. p informs $gs(p)$ to the processes in Q. In case (2), p selects $gs(p) = gs(p')$ and enters CS. By the rule (2), unnecessary blocking is avoided.

3.1.3 New Group Mutual Exclusion Algorithm

This subsection shows our new group mutual exclusion algorithm. We provide overall algorithm in Appendix.

Algorithm for requesting process

The outline of procedure for requesting process is as follows:

1. When p whose group set is $G(p)$ wants to enter CS, p selects a quorum $Q \in C$ and sends "$Request(G)$" to every process in Q.

2. There are two cases to enter CS.

(a) When p receives "Ok" from every process in Q, p arbitrary selects one group $g \in G(p)$ as $gs(p)$, sends "$Lock(g)$" to every process in Q, and enters CS.

(b) When p receives "$Enter(g)$" from some process in Q, p sets g as $gs(p)$ enters CS.

3. When exiting from CS, execute the following:

(a) (entered CS by "Ok") p sends "$Release$" to every process in Q. When p receives "$Enter(g)$" from every process in Q, it sends "$Over$" to every process in Q

(b) (entered CS by "$Enter$") p sends "$NoNeed$" to the process "$Enter$" is arrived.

The exiting procedure when entered by "Ok" is a little complicated. When "$Release$" is arrived at a access server process, the process must not send "Ok" to a waiting request immediately. Let us consider the following example. p_1, whose group set $G(p_1) = \{g_1\}$, uses $Q_1 = \{q_1, q_2\}$ and sends "$Request$". q_1 and q_2 send "Ok" to p_1 and p_1 enters CS. After that, p_2 sends "$Request$" to $Q_2 = \{q_2, q_3\}$ and $G(p_2) = \{g_2\}(g_2 \neq g_1)$. q_3 sends "Ok" to p_2. However, since q_2 has sent "Ok" to p_1, p_2 receives no reply from q_2. Then, p_3, whose group set $G(p_3) = \{g_1\}$, sends "$Request$" to $Q_3 = \{q_1, q_3\}$. q_1 replies "$Enter(g_1)$" to p_3, since it has sent "Ok" to a request whose group selection is g_1. Thus, p_3 can enter CS. Now, suppose that p_1 exits from CS. p_1 sends "$Release$" to q_1 and q_2. If q_2 sends "Ok" to p_2 immediately, p_2 enters CS, although p_3 is currently entering CS. Thus, group mutual exclusion is not achieved. Therefore, each process must not send "Ok" to a waiting request until exiting of every process which entered CS by receiving "$Enter$".

Two-phase release procedure is used to achieve it. When "$Release$" is arrived, each process sends no more "$Enter$" to any other requests,

waits for exiting of every process to which "*Enter*" is sent, and then replies "*Finished*". When the requesting process p receives "*Finished*" from every process in Q, it means all request which entered CS by receiving "*Enter*" has exited. Then p sends "*Over*" to every member of Q. When "*Over*" is arrived, each access server process sends "*Ok*" to the highest priority waiting request.

Algorithm for access server process

The outline of procedure for access server process is thus as follows: Variable *status* stores the current status of the process. $Status = vacant$ means there is no request, *waitlock* means that it has sent "*Ok*" to some process but "*Lock*" is not arrived, and *locked* means "*Lock*" is received. Variable *group* stores current group when some process is entering CS.

In order to avoid starvation, each request has Lamport's logical clock [16]. A request with smaller logical clock has a higher priority. Thus the oldest request will eventually be the highest priority and it can enter CS. The procedure to update the logical clock and assign the logical clock to each request is omitted in this procedure for simplicity.

The following is outline of the procedure for access server processes.

1. When q receives $Request(G)$ from p, q inserts it to the queue Que.

(a) If $status = vacant$, q sends "*Ok*" to p.

(b) If $status = locked$ and $group \in G$, q sends "$Enter(group)$" to p.

2. When q receives "Lock(g)" from p, q sets $group = g$ and $status = locked$. q then sends "$Enter(g)$" to every waiting request in Que whose group set G satisfies $g \in G$.

3. When q receives "Release", q stops further sending of "$Enter$" (by changing $status$). And if there is no process to which "$Enter$" is sent, q replies "$Finished$".

4. When q receives "$NoNeed$" from p, q sends "$Finished$" to the process "$Release$" is arrived, if there is currently no process q has sent "$Enter$" or "Ok".

5. When q receives "$Over$", q sets $status = vacant$ and tries to send "Ok" to highest priority request in Que.

In order to avoid deadlock, an additional mechanism is necessary. Assume that the priority of p_2 is higher than that of p_1. At q_1, "$Request$" arrives in the order of p_1, p_2 and at q_2, arrives in the order of p_2,p_1. In this a case, the "Ok" sent from q_1 to p_1 must be cancelled to avoid deadlock. The cancel procedure is just the same as the one for simple mutual exclusion in [32].

1. When process q receives "$Request(G)$" from p_2, if q has sent "Ok" to p_1 but "$Lock$" has not been arrived, and p_2's priority is higher than that of p_1, then q sends "$Cancel$" to p_1.

2. When p_1 receives "$Cancel$" from q, if it has not entered CS, it replies "$Cancelled$" to q (and waits for next arrival of "Ok").

3. When q receives "$Cancelled$", q sends "Ok" to the highest priority request in Que.

The meaning of variables used in two Algorithm are as follows: As for each requesting process, $Rstatus$ stores the status of the request. $RStatus = wait$ means it is waiting for "Ok" or "$Enter$". In means that it is in the CS, out means that it has exited from CS. The quorum currently using is stored in Q. The set of processes from which "Ok" has

been arrived (when making a request) or "$Finished$" has been arrived (when releasing) is stored in K. Thus, if $K = Q$, the requesting process can enter CS (when making a request) or can send "$Over$" (when releasing).

Next, the meaning of variables for each access server process are described. Que is the priority queue of requests. Each entry $Que[i]$ has entry "$Que[i].pr$" (the requesting process), $Que[i].G$(the set of groups), and $Que[i].status$ (status of the request). $Que[i].status = wait$ when it is blocked by a higher priority request. $waitlock$ when "Ok" is sent and waiting for "$Lock$" from the process. $enter$ when the process is entering CS. $releasing$ when the process is releasing. $waitcancel$ when "$Cancel$" is sent and waiting for the reply.

Each access server process sends "Ok" to at most one request at any time. The requesting process to which "Ok" is sent is stored in variable $sentok$. Variable $status$ stores $Que[i].status$ of the request of $sentok$. When there is no such request, $status = vacant$. Variable $using$ is the set of processes currently entering CS.

Note that when p exits from CS and makes another request, p might receive replies to the older request. p can ignore such old replies easily if the logical clock of each request is attached to every reply message. The procedure to ignore such replies is omitted in Algo.1 in Appendix for simplicity.

3.1.4 Correctness of the Algorithm

This subsection shows the correctness of the algorithm. Firstly, it is shown that group mutual exclusion is achieved.

Theorem 3.1.1 *p_1 and p_2 never enter CS at the same time by our Algo.1 if $gs(p_1) \neq gs(p_2)$.*

Proof: Suppose that the above situation occurs. Let $g_1 = gs(p_1)$, $g_2 = gs(p_2)$, and $Q_1(Q_2)$ be the quorum $p_1(p_2)$ uses. p_1(and p_2) enter CS by (1) receiving "Ok" from every member of $Q_1(Q_2)$ or (2) receiving "$Enter$" from some process in $Q_1(Q_2)$. In case (2), there is another process p_1'(and p_2') whose group selection is g_1(and g_2) and p_1'(and p_2') enters CS before p_1(and p_2). p_1'(and p_2') receives "Ok" from every member of some quorum, say Q_1'(and Q_2'). Though p_1'(and p_2') might have exited from CS before p_1(and p_2) exits from CS, the processes in Q_1'(and Q_2') cannot send "Ok" to any other process until p_1 (and p_2) exits from CS and sends "$NoNeed$".

In case(1), let $p_1' = p_1$ (p_2'=p_2) and Q_1'= Q_1(Q_2'= Q_2).

$p_1' \neq p_2'$ holds in any cases since $gs(p_1') \neq gs(p_2')$.

Now, every process in $Q_1'(Q_2')$ has sent "Ok" to $p_1'(p_2')$ at the same time. Since $Q_1' \cap Q_2' \neq \phi$ and each process sends "Ok" to at most one process at the same time, this situation cannot occur. □

Theorem 3.1.2 *No starvation occurs by the algorithm in Algo.1*

Proof: Assume that starvation occurs. Let p_1 be the highest priority request which cannot enter CS forever. Let Q_1 be the quorum p_1 selects. From the assumption, p_1 is the highest priority request that is not entering CS from some time t. Let t' be the time when "Request" from p_1 arrives at every member of Q_1. Let us consider the system state after time $T = max(t, t')$. Each process $q \in Q_1$ must try to send "Ok" to p_1 because p_1 is the highest priority. If q has not sent "Ok" to any process, obviously it sends "Ok" to p_1. If q has sent "Ok" to another process, say p_2, q sends "$Cancel$" p_2. If p_2 has not entered CS, it replies "$Cancelled$" and thus, q will be able to send "Ok" to p_1. If p_2 has entered CS before arrival of "$Cancel$", p_2 eventually exits from CS. After that, q does not send "$Enter$" to any other requests. In addition,

every process which entered CS by receiving "$Enter$" from q (before exiting of p_2) also eventually exits from CS. Thus, q eventually sends "$Finished$" to p_2 and thus, p_2 eventually sends "$Over$" to q. Therefore, q will be able to send "Ok" to p_1 and no starvation occurs. □

3.2 Toward Secure Group Mutual Exclusion Algorithm

Through the previous researches, we have designed efficient group mutual exclusion algorithm without unnecessary blocking. As already explained, researches on distributed algorithm don't focus on security aspects because their goal is to avoid conflict between each process and to keep efficiency at the same time. Through studying on both security and distributed algorithm area, we have identified the relationship between them. Two areas both deal with fault-tolerance of each party.

Usually, many distributed systems manage some forms of data, such as files or databases. The performance and fault-tolerance of such systems may be enhanced if the repositories for the data are physically distributed. Nevertheless, distribution makes security more difficult, since it may be difficult to ensure that each repository is physically secure. Approach on designing secure distributed algorithm based on group mutual exclusion algorithm can be valuable. We specify secure distributed algorithm by using both group mutual exclusion algorithm and secret sharing scheme.

3.2.1 Requirements

We formulate the requirements in order to satisfy secure distributed algorithm. Until now, there are no clear requirements defined for secure

distributed algorithm since our challenge is performed for the first time. We classify security and general requirement as follows:

General Requirement

- **Group Mutual Exclusion** At any given time, no two processes of different groups are in CS simultaneously.
- **Deadlock freeness** Non-existence of blocking states, i.e.states without successor, should be guaranteed.
- **Starvation-freeness** A process wishing to enter CS will eventually succeed.
- **Consistency** At least one correct process should know whether other request is legitimate or not.

Security Requirement

- **Privacy** Only authorized user can see secret information through keeping information secret.
- **Authentication** Identity of each user should be checked.
- **Unforgeability** Only authorized user can issue valid secret key.
- **Availability** Fault-tolerance. Even though some fault happen, service for users should be kept continuously.

3.2.2 Our Model and Assumptions

We can consider the following general scenario for secure access control to a database. The photography company has a large digitized pictures database of various parts of the local area. This database is updated periodically, as new photographs are added. Company customers buy

the license to access a set of photographs, say of some geographic area. When the license expires, the customer is not allowed access any more. Furthermore, the company would like to be able to quickly revoke the customers privileges at any time due to unauthorized transfer of information. The company needs a distributed protocol to enforce this permission policy. The protocol should run using a widespread collection of access servers, which may be completely separate from the actual data servers. A basic part of this protocol is just maintaining a consistent view of the permission status of every customer, which is a classical question concerning replicated data. Note that servers may be unavailable due to crashes or communication failures, so the protocol needs to overcome this and allow high availability of the service. The information in the database is highly sensitive so it must be protected. The protection should be against cheating users, rather than against dishonest access server personnel. In this example, through using secret sharing scheme, we can make safe protocol for access control. In distributed network, many users have permission of data they want to listen but they can not access data at the same time since shared data can be used one by one. Mutual exclusion guarantees an exclusive access to a common resource among a set of competing processes. So, it is worth to considering both mutual exclusion for consistency of competing processes and security of each process at the same time.

Attack.

Before we explain our model, we must consider all the possible attacks. In group mutual exclusion algorithm, processes who want same data can access to file, even though process is a malicious actor. In other words, he can eavesdrop or modify secret data easily without any interruption mechanism if he just wants to access the same data. The aim of the distributed algorithm is to improve efficiency of algorithm

with guaranteeing deadlock-freeness and starvation-freeness. However, for the consideration of practical use, the proper security of algorithm should be guaranteed.

We define potential attackers in our model in order to help practical use of distributed algorithm. Potential attackers can be classified two groups : general attacker who gets data without any action due to failure of processes and cryptographic attacker who eavesdrops and modifies data through corrupted server. Before proceeding any further, we need to clarify the scope of fault-tolerance both in distributed algorithm and in security area.

Definition 3.2.1 *Fault-tolerance in distributed algorithm means that even though crashes or communication failures happen, service can be kept safely since influence of faults can be localized. Fault-tolerance of security aspect means that attacker can not eavesdrop and modify data using corrupted server.*

When communication failures happen, a legitimate user can get continuous service from servers of correct one quorum at least using intersection property of quorum system against general attacker. Also, cryptographic attacker can not get partial secret value from corrupted server. Our algorithm is focused on providing strong fault-tolerance against both general attacker and cryptographic attacker.

Assumptions

Our scheme can be regarded as an improvement of the our proposed scheme. We assume two elements. First, we assume that there are no coordination between the servers in distributed system. Each server replies to a request based only on information it holds locally. Second, each user has a secure and authenticated channel of communication

with the servers. So, Alice cannot masquerade as Bob and obtain access permission by this.

We make use of three components in our whole algorithm. There are access servers(AS) who grant access, data server(DS) that maintains database and users who want to access data. Since we use SSS, reconstruction function, h_A, for secret information and polynomial function, $\prod_i(SK, r)$ are necessary. Q is denoted as requested quorum set. $D(x)$ is denoted as decrypted function. We will stands SK is a secret key for accessing DB for the user. We will use X as requested data item from user.

3.2.3 Modified Algorithm

Registration Phase

To get the permission for accessing data, registration phase is necessary. Customers should buy the e-ticket to access a set of music data. So, customers go to registration office and buy the e-ticket. The office for access server gives the e-ticket to customer. After registration, each access server has customers' list which store their ID, e-ticket period, and permission of data X. When customers request to access data, access server checks their list and give authorization to them.

Commitment Phase

The algorithm is shown in Figure 3.1.We describe the whole phase in brief. $User$ who wants to access data requests to AS. AS check whether $User$ is authorized person or not through looking up customers' list directory. After that, AS generate SK, r and calculate s_i using SSS. AS send "$Ok(s_i)$" or "$Enter(g, s_i)$" message to $User$. After collecting all s_i from each AS in a set of quorum, $User$ can get key SK. $User$ send "$Lock$" message to each AS to inform that $User$ will access data.

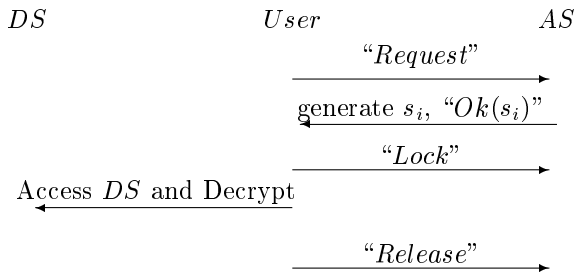

Figure 3.1: Sketch of our Algorithm

$User$ can decrypt an encrypted data and show data content. Also, $User$ who wants to access the same data can access data.

Proposed Algorithm

Our algorithm consists of two parts. AccessServer is for the behavior of AS that as a quorum member. RequestingProcess is for the behavior of a user that acts as a group member. We modified our algorithm in order to adapt secret sharing schemes for access structure to our group mutual exclusion algorithm.

Modified RequestingProcess Procedure

□**var** $Rstatus = wait$: status of request;

p : request process; q : server process;

$Q = \phi$: set of process within one quorum;

K : set of process; /* reply is received */

G : set of group; /* current group set */

X : data part that user wants;

T : set of partial secret value;

SK : secret key value;

□When p (group set is G) wants to enter CS

begin

$Rstatus := wait$;

Select arbitrary Q from coterie;

$K := \phi$; $T := \phi$;

send "$Request(G, X)$" to all $q \in Q$;

end; /*end of request initiation*/

□At arrival of "$Ok(s_i)$" from q

begin

if $Rstatus = wait$ **then begin**

$K := K \cup \{q\}$; $T := T \cup \{s_i\}$ /*collect s_i from Q */

if $K = Q$ **then begin**

select arbitrary $g \in G$;

send "$Lock(g)$" to all $q \in Q$;

$SK = h_A(\{s_i\}_{i \in A})$;

$Rstatus := in$;

$y(x) = D(x) \oplus SK$;

$Rstatus := out$;

send "$Release$" to all $q \in Q$;

$K := \phi$;

end /* end of $K = Q$ */

end /* end of $Rstatus = wait$ */

end /* end of arrival "$Ok(s_i)$" */

The outline of modified requesting processes as important procedure part is as follows:

1. When user p whose group set is $G(p)$ wants to enter CS, p selects a quorum $Q \in C$ and sends "$Request(G, X)$" to every process in Q.

2. There are two cases to enter CS.

(a) When user p receives "$Ok(s_i)$" from every process in Q, p arbitrary selects one group $g \in G(p)$ as $gs(p)$ and get secret key using reconstruction function, $SK = h_A(\{s_i\}_{i \in A})$. After that, he sends "$Lock(g)$" to every process in Q, and enters CS with the key he can decrypt data, $y(x) = D(x) \oplus SK$.

(b) When p receives "$Enter(g, s_i)$" from some process in Q, he waits until he receives "$Ok(s_i)$" from other processes in Q. After that, p sets g as $gs(p)$. Then get secret key using reconstruction function, $SK = h_A(\{s_i\}_{i \in A})$. After that, he sends "$Lock(g)$" to every process in Q, and enters CS with the key he can decrypt data, $y(x) = D(x) \oplus SK$.

Modified AccessServer Procedure

□**var** $status = vacant$:status;
$group$: current group;
$Que = null$:priority queue of requests;
$waiting = null$:process;
$sentok = null$:process;
$using = null$:set of processes;

□At arrival of "$Request(G, X)$" from p
begin
/*check authorization*/
if $(p, X) \in$ Registration List **then begin**
Insert "$Request(G, X)$" to Que;
/*assume $Que[i]$ be the position*/
$Que[i].status := wait$;
$Que[i].pr := p$; $Que[i].G := G$;

if $status = vacant$ **then begin**

/* generate SK with encryption random function $Key_\alpha(X)$ */

$SK = Key_\alpha(X)$;

/* generate pseudo random string with a private seed γ */

$r = R_\gamma(X \oplus p)$;

/* compute s_i using the SSS*/

$s_i = \prod_i(SK, r)$;

send "$Ok(s_i)$" to p ;

$sentok := p$;

$Que[i].status := waitlock$;

$status := waitlock$;

end /*end of $status = vacant$ */

end /*end of $(p, X) \in$ Registration List */

end /*end of arrival "$Request(G, X)$" */

The outline of modified access server processes as important procedure part is as follows:

1. When q receives "$Request(G, X)$" from p, q inserts it to the queue Que. Using identity of p and X, it checks the authorization. If the request is from an authorized user

(a) If $status = vacant$, AS doesn't give the permission to other processes, and the server q generates $SK = Key_\alpha(X)$ and a pseudo-random string $r = R_\gamma(X \oplus p)$. Server q_i then computes its share of the key, $s_i = \prod_i(SK, r)$ using the SSS, and send "$Ok(s_i)$" to p.

(b) If $status = locked$ and $group \in G$, q sends "$Enter(group, s_i)$" to p.

2. When q receives "$Lock(g)$"from p, q sends "$Enter(g, s_i)$" to every waiting request in Que whose group set G satisfies $g \in G$.

3.2.4 Comparison

Our Algo.1 and Joung [40] proposed distributed algorithm satisfying requirement of group mutual exclusion. However, two algorithms focus on the consistency with competing processes which are the issue of distributed computing area. Protocol by presented Naor and Wool [24] also uses quorum system for fault-tolerance of each server. However, they consider only the security of the distributed protocol. We presented an algorithm keeping both the property of group mutual exclusion and secure algorithm satisfying confidentiality and authentication.

Security Comparison

- **Group mutual exclusion** By theorem 3.1.1, group mutual exclusion is satisfied. In other words, customers who request different data cannot access data, but customers that have requested the same data can. Therefore, at any given time, no two customers of different groups can be in CS at the same time.

- **Deadlock-freeness** By theorem 3.1.1, each customer who wants to access different data does not be waiting for other data permanently.

- **Starvation-freeness** By theorem 3.1.2,the customers who want access different data are able to access data eventually.

- **Consistency** The intersection property of a quorum system ensures that in any set which can collectively grant the permission to the customer, at least one server is informed that the request is not legitimate. So, the consistency is guaranteed by the fact that obtaining replies from less than a quorum of servers does not leak information to the user.

Table 3.1: Comparison of Algorithms

	[40]	[24]	Alg1.	Alg2.
Mutual exclusion	O	X	O	O
Deadlock-freeness	O	X	O	O
Starvation-freeness	O	X	O	O
Consistency	O	O	O	O
Privacy	X	O	X	O
Authentication	X	O	X	O
Unforgeability	X	O	X	O
Availability	O	O	O	O

- **Privacy** Through SSS and intersection property, even though cryptographic attacker gets the permission from the corrupted server, he cannot get partial information.
- **Authentication** After customer grants by a quorum of servers using authorization check, he can access data.
- **Unforgeability** Using SSS, customer can obtain partial information from k of the n servers. That means any malicious actor can not forge a complete secret information by corrupted servers fewer than k servers.
- **Availability** Due to fault-tolerance property of our algorithm, service for users can be kept continuously.

Performance Comparison

Table 3.2 shows the performance comparison of main communication overhead of algorithms. Let $|Q|$ be the size of the smallest quorum in a coterie.

Table 3.2: Communication Comparison

	Communication
[40]	$8\|Q\|$
Our Algo1.	$9\|Q\|$
Our Algo2.	$9\|Q\|$

The no-exclusion case is that there is only one request at any time. The case is considered to the best case in the discussion of simple mutual exclusion.

1. Process p sends "$Request(G, X)$" to every member of Q.
2. p receives "$Ok(s_i)$" from every member of Q.
3. p sends "$Lock(g)$" to every member of Q and enters CS.
4. p exits from CS and sends "$Release$" to every member of Q.
5. p receives "$Finished$" from every member of Q.
6. p sends "$Over$" to every member of Q.

In case of, the total number of messages is $6|Q|$.

Next, consider the worst case, when the highest priority request arrives later.

1. p sends "$Request(G, X)$" to every member of Q.
2. Each process $q_i \in Q$ has sent "$Ok(s_i)$" to another process p_i, whose priority is lower than that of p. q_i sends "$Cancel$" to p_i.
3. p_i sends "$Cancelled$" to q_i.
4. q_i receives "$Cancelled$" and sends "$Ok(s_i)$" to p.

5. p receives "Ok" from every member of Q. Thus, it sends "$Lock$" to every member of Q and enters CS.

6. p exits from CS and sends "$Release$" to every member of Q.

7. p receives "$Finished$" from every member of Q.

8. p sends "$Over$" to every member of Q.

9. q_i receives "$Over$" and sends "Ok" again to the process to which "$Cancel$" is sent. (The messages for p_i to enter and exit from CS is counted as the messages for p_i)

The total number of messages per request is $9|Q|$. The worst case number of messages is larger than $8|Q|$ in [40]. Additional cryptographic technique does not affect the total number of messages. So, the communication complexity of our algorithm is $9|Q|$.

IV. Conclusion

Throughout this thesis, we have studied on secure group mutual exclusion algorithm design in the distributed system environment. For the concrete design, we reviewed previous related works and pointed out their problems. And then we have suggested the improved algorithm without unnecessary blocking. Also, we have proposed a secure algorithm based on our proposed distributed algorithm.

Firstly, we have presented new quorum based distributed group mutual exclusion algorithm. The Joung's quorum-based algorithm has a case when two processes cannot enter critical section at the same time even though they can do so. We proposed a new algorithm which prevents unnecessary blocking and show its correctness proof.

Secondly, we have proposed a secure quorum-based distributed algorithm. Generally, researches on distributed algorithm don't deal with security aspects. However, challenge on designing secure algorithm can be valuable. So, we proposed secure distributed algorithm by using both group mutual exclusion algorithm and secret sharing scheme. Also, our algorithm guarantees the requirements of mutual exclusion algorithm but also users' privacy and authentication.

As further works, real estimation from implementation is meaningful to consolidate our efficiency. Also, it is necessary to prove that our quorum system is SDR or not. In the context of designing decentralized economic mechanisms such as distributed algorithm, it turned out to be important to know when one can construct an SDR for a collection of sets that cover the parameter space characterizing a finite number of economic agents. We already introduced m group quorum system that satisfied non-dominated, non empty, and intersection property of

each element. Such property would be used for proof of SDR. And it is valuable to do research more secure practical distributed algorithm not just pure distributed algorithm without security.

Appendix: Algo1. Overall Program

Program RequestingProcess(p:process)

□**var** $Rstatus = wait$: status of request;

$Q = \phi$: set of process; /* quorum */;

K : set of process; /* reply is received */

G : set of group; /* current group set */

□When p (group set is G) wants to enter CS

begin

$Rstatus$:= wait;

Select arbitrary Q from coterie;

$K := \phi$;

send "$Request(G)$" to all $q \in Q$;

end; /*end of request initiation*/

□At arrival of "Ok" from q

begin

if $Rstatus = wait$ **then begin**

$K := K \cup \{q\}$;

if $K = Q$ **then begin**

select arbitrary $g \in G$;

send "$Lock(g)$" to all $q \in Q$;

$Rstatus := in$;

.../* in the CS */

$Rstatus := out$;

send "$Release$" to all $q \in Q$;

$K := \phi$; /* waits for "$Finished$" */

end /* end of $K = Q$ */

end /* end of $Rstatus = wait$ */

end /* end of arrival "Ok" */

□At arrival of "$Enter(g)$" from q

begin

if $Rstatus = wait$ **then begin**

send "$NoNeed$" to all $r \in Q - \{q\}$;

$Rstatus := in$;

.../* in the CS */

$Rstatus := out$;

send "$NoNeed$" to q;

end /* end of $Rstatus = wait$ */

end /* end of arrival "$Enter$" */

□At arrival of "$Cancel$" from q

begin

if $Rstatus = wait$ **then begin**

$K := K - \{q\}$;

send "$Cancelled$" to q;

end /* end of $Rstatus = wait$ */

end /* end of arrival "$Cancel$" */

□At arrival of "$Finished$" from q

begin

$K := K \cup \{q\}$;

if $K = Q$ **then** send "$Over$" to all $q \in Q$;

end /* end of arrival "$Finished$" */

Program AccessServer Procedure(q:process)

□**var** $status = vacant$:status;

$group$: group; /* current group */

$Que = null$:priority queue of requests;

$waiting = null$:process; /* waits "$Lock$" */

$sentok = null$:process; /* "Ok" is sent */

$using = null$:set of processes;

□At arrival of "$Request(G)$" from p

begin

Insert the request to Que;

/*assume $Que[i]$ be the position*/

$Que[i].status := wait$;

$Que[i].pr := p$; $Que[i].G := G$;

if $status = vacant$ **then begin**

send "Ok" to p ;

$sentok := p$;

$Que[i].status := waitlock$;

$status := waitlock$;

end /*end of $status = vacant$ */

else if $status = locked$ **then begin**

if $group \in G$ and $Que[1].status = enter$

/* $Que[1]$:highest priority request */

then begin

send "$Enter(group)$" to p;

$using := using + \{p\}$;

$Que[i].status := enter$;

end

end /* end of locked */

else if $status = waitlock$ then begin

if $Que[i]$ is highest priority **then begin**

send "$Cancel$" to process $sentok$;

/* assume $Que[k].pr = sentok$ */

$Que[k].status := waitcancel$;

$status := waitcancel$;

end

end /* end of waitlock */

end /*end of "$Request$" arrival */

□At arrival of "$Lock(g)$" from $p(p = Que[i].pr)$

begin

$using := \{p\}$;

$Que[i].status := enter$;

$status := locked$;

$group := g$;

if $Que[1].status = enter$ or $g \in Que[1].G$

then begin / * $Que[1]$ can enter CS */

for every request $Que[k](k \neq i)$ such that

$g \in Que[k].G$ do begin

send "$Enter(g)$" to $Que[k].pr$;

$Que[k].status := enter$;

$using := using + \{Que[k].pr\}$;

end; /* end of do */

end;

end; /*end of arrival "$Lock$" */

□At arrival of "$Release$" from $p(p = Que[i].pr)$

begin

$status := releasing$;

remove entry $Que[i]$;

$using := using - \{p\}$;

if $using = \phi$ **then** send "$Finished$" to p;

end;

□At arrival of "$NoNeed$" from $p(p = Que[i].pr)$

begin

remove entry $Que[i]$;

if $p = sentok$ **then** NewChance

else if $p \in using$ **then begin**

$using := using - \{p\}$;

if $using = \phi$ **then** send "$Finished$" to $sentok$;

end

end;

□At arrival of "$Cancelled$" from $p(p = Que[i].pr)$

begin

$Que[i].status := wait$;

SendOk;

end;

□At arrival of "$Over$" from $p(p = Que[i].pr)$

SendOk;

□**procedure** SendOk; /* permission released. */

begin

if Que is not empty **then begin**

/* Que[1] :highest priority request */

Send "Ok" to $Que[1].pr$;

$sentok := Que[1].pr$;

$Que[1].status := waitlock$;

$status := waitlock$;

end /* end of Que is not empty */

else $status := vacant$;

end;

안전한 그룹 상호배제 알고리즘에 관한 연구

박재혁

컴퓨터 기술의 발달과 인터넷 보급의 확산은 전통적인 중앙집중적 네트워크 관리환경에서 분산 컴퓨팅 환경으로 변화를 이끌고 있다. 분산 시스템은 프로세서나 메모리 셀, 버스, 프린터등과 같은 많은 타입의 리소스를 공유하는 프로세스 집합들로 구성된다. 분산시스템의 분산알고리즘은 많은 분산된 컴퓨터들이 어떻게 하면 협력적으로 작업을 수행할 수 있는지를 목표로 하고 있다. 현존하는 많은 분산알고리즘은 리소스를 공유하는 프로세스간의 서로 충돌이 이루어지는 상호배제문제를 해결하는데 초점을 맞추고 있다. 특히 최근에는 그룹단위의 상호배제 문제의 연구가 이루어지고 있다. 그룹 상호배제는 하나의 리소스를 같은 그룹내의 모든 프로세스에 의해서 공유되도록 할 수 있는 상호배제의 일반화이다. 하지만, 다른 그룹의 프로세스들은 상호 배타적인 방법으로 하나의 리소스를 사용하도록 요청된다. 즉, 다른 그룹의 프로세스들은 이미 임계영역에 있는 프로세스가 그 리소스에 대한 사용이 끝난 후 임계영역에 들어갈 수 있다. 그룹상호배제를 통해 효율성을 증가시킬 수 있는 장점이 있다.

분산된 환경에서 프로세스간의 충돌을 막고 효율적으로 컴퓨팅이 가능하도록 하는 것이 중요하나, 기본적인 인터넷의 개방성과 분산된 환경은 전송되는 정보에 대한 안전성에 큰 취약성을 노출한다. 따라서, 분산알고리즘의 설계 시 상호배제를 통한 프로세스간의 효율성과 함께 안전성을 보장하는 것은 가치 있는 일이다.

이러한 목적을 달성하기 위한 방법론으로 본 논문에서는, 우선 기존의 제안된 그룹상호배제 알고리즘의 문제점을 소개하고, 그런 문제점을 해결하는 새로운 알고리즘을 제안한다. 두번째로, 제안된 알고

리즘을 기반으로 하여 보안성을 고려한 안전한 쿼럼 기반의 분산알고리즘을 제안한다. 제안되는 알고리즘은 분산알고리즘을 기반으로 하기때문에 기본적인 요구사항을 만족하며, 또한 보안요구사항을 동시에 만족한다.

References

1. G.R.Blakley, Safeguarding cryptographic keys. *Proc. AFIPS, NCC*, pp.313-317, 1979.

2. J.Benaloh and J.Leichter, Generalized secret sharing and monotone functions. *In Advances in Cryptology-CRYPTO'88*, LNCS403, pp.27-36, Springer-Verlag, 1988.

3. D.Beaver and A.Wool, Quorum based secure multi-party computation, *Advance in Cryptology-EUROCRYPT'98*, LNCS 1403,pp.375-390, Springer-Verlag, 1998.

4. S.Cantareli, A.K. Datta, F.Perit, V.Villain, Token Based Group Mutual Exclusion for Asynchronous Rings, *Proc. of 21st ICDCS*, pp.691-694, 2001.

5. S.B.Davidson, H.Garcia-Molina, and D.Skeen, Consistency in partitioned networks. *ACM Computing Surveys*, Vol.17, No.3, pp.341-370, 1985.

6. S.Fujita, M.Yamashita, and T.Ae, Distributed k-Mutual Exclusion Problem and k-Coteries, *Proc.Symp. Algorithms*, pp.22-31,1991.

7. V.Hadzlilacos, A Note on Group Mutual Exclusion,*Proc. 20th PODC*, pp.100-106, 2001.

8. M.P.Herlihy and J.D.Tygar, How to make replicated data secure, *CRYPTO'87*, LNCS298, pp.379-391, Springer-Verlag, 1988.

9. S.Ichiro and K.Tadao, A Distributed Mutual Exclusion Algorithm, *ACM*, Vol.3, No.4, pp.344-349, 1985.

10. M.Ito, A.Saito, and T.Nishizeki, Secret Sharing schemes realizing general access structure. *Proc. IEEE Global Telecommunication Conf.(Globecom'87)*, pp.99-102, 1987.

11. S.Jajodia and D.Mutchler, Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Trans. Database Sys.*, Vol.15, No.2, pp.230-280, 1990.

12. H.Kakugawa, S.Fujita, and M.Yamashita, Availability of k-Coteries,*IEEE Trans.Computerss*, Vol.42, No.5, pp.553-558, 1993.

13. H.Kakugawa, S.Fujita, and M.Yamashita, A Distributed k-Mutual Exclusion Algorithm using k-Coterie, *Information Processing Letters*, 1994.

14. H.Kakugawa, M.Yamashita, Local Coteries and a Distributed Resource Allocation Algorithm, *Information Processing Society of Japan*, Vol.37, No.8, 1996.

15. P.Keane and M.Moir, A Simple Local-Spin Group Mutual Exclusion Algorithm. *IEEE Trans.Parallel and Distributed Systems*, Vol.12,No.7, pp.673-685, 2001.

16. L.Lamport, Time, clocks, and the ordering of events in a distributed system, *Communication of ACM*, Vol.21, No.7, pp.558-565, 1978.

17. M.Maekawa, A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems, *ACM Trans.on Computer Systems*, Vol.3, No.2, pp.145-159, 1985.

18. H.Molina and D.Barbara, How to Assign Votes in a Distributed Systems, *Journal of the ACM*, Vol.32, No.4, pp.841-860, 1985.

19. S.J.Mullender and P.M.B.Vitanyi, Distributed matchmaking. *Algorithmica*, No.3, pp.367-391, 1988.

20. Y.Manabe and S.Aoyagi, A distributed k-mutual exclusion algorithm using k-coterie, *COMP*, 1993.

21. Y.Manabe, R.Baldoni, M.Raynal, S.Aoyagi, k-Arbiter : A safe and general scheme for h-out-of-k mutual exclusion, *Theoretical Computer Science*, pp97-112, 1998.

22. Y.Manabe, N.Tajima,(h, k)-arbiters for h-out of-k mutual exclusion problem, *Proc.of 19th ICDCS*,pp.216-223, 1999.

23. M.L.Neilsen, Properties of nondominated K-coteries, *The J. of Systems and Software*, pp.91-96, 1997.

24. M.Naor and A.Wool, Access Control and Signatures via Quorum Secret Sharing,*In Proc. 3rd ACM Conf. Comp. and Comm. Security*, pp.157-168, 1998.

25. Geraint Price, Broadening the Scope of Fault Tolerance within Secure Services, *Security Protocols Workshop2000*, LNCS 2133, pp.155-169, 2000.

26. J.Park, K.Kim, and Y.Manabe, Group Mutual Exclusion using Group Choice,*CISC02*, 2002.

27. G.Ricart and A.K.Agrawala, An optimal algorithm for mutual exclusion in computer networks, *Communication of ACM*, pp.9-17, 1981.

28. M.Raynal, Algorithms for Mutual Exclusion. MIT press, 1986.

29. M.K.Reiter and K.P.Birman, How to securely replicate services, *ACM Trans. Prog. Lang. Sys.*, Vol.16, No.3, pp.986-1009, 1994.

30. A.Shamir, How to share a secret. *Communications of the ACM*, Vol.22, No.11, pp.612-613, 1979.

31. I.Suzuki and T.Kasami, A Distributed Mutual Exclusion Algorithm, *ACM Trans.Computer Systems*, Vol.3, No.4, pp.344-349, 1985.

32. B.A.Sanders, The Information Structure of Distributed Mutual Exclusion Algorithms, *ACM TOCS*, Vol.4, No.4, pp.284-299, 1987.

33. R.K.Srimani, S.R.Das,Distributed Mutual Exclusion Algorithms, *IEEE Computer Society Press*, 1992.

34. M.Singhal,A taxonomy of distributed mutual exclusion, *Journal of Parallel and Distributed Computing*,Vol.18, No.1, pp.94-101, 1993.

35. S.-C.Sung, Y.Manabe, Coterie for Generalized Mutual Exclusion Problem, *Trans.IEICE* Vol.E82-D, No.5,pp.968-972, 1999.

36. K.Vidyasankar, A Highly Concurrent Group Mutual l-exclusion Algorithm, *Proc.of 21th PODC*,2002.

37. Y.-J.Joung, Asychronous group mutual exclusion (extended abstract). *In Proc. 17th PODC*,pp.51-60, 1998.

38. Y.-J.Joung, The congenial talking philosophers problem in computer networks(extended abstract). *In Proc, 13th DISC*, LNCS 1693, pp.195-209, 1999.

39. Y.-J.Joung, Asynchronous Group Mutual Exclusion, *Distributed Computing*, Vol.13, No.4, pp.189-206, 2000.

40. Y.-J.Joung, Quorum-based Algorithms for Group Mutual Exclusion, *Proc. of DISC*, LNCS 2180, pp.16-32, 2001.

41. T.W.Yan and H.Garcia-Molina, Distributed selective dissemination of information, *In Proc. 3rd Inter. Conf. Par. Dist. Info. Sys.*, pp.89-98, 1994.

42. L.Zhou, F.Schneider and R.Van Renesse, COCA: A Secure Distributed Online Certification Authority, *ACM Trans. Computer Systems*, Vol.20, No.4, pp.329-368, 2002.

Acknowledgement

First, I would like to express my sincere gratitude to Prof. Kwangjo Kim, my academic advisor, for his constant direction and support. He always has shown his consistent affection and encouragement for me to carry out my research and life in ICU. Special thanks also goes to Prof. Jae Choon Cha and Prof. C.Pandu Rangan for their generosity and agreeing to serve as committee members of my thesis.

I also would like to thanks to all members of cryptology and information security laboratory: Jeongkyu Yang, Kyusuk Han, and Seokkyu Kang, Vo Duc Lim from Vietnam, Yan Xie, Xiaofeng Chen, Kui Ren, Jiqiang Iv,and Ping Wang from China, for giving me lots of interests and good advices during the course of my study.

In addition, I appreciate to the graduates, Wooseok Ham, Jongseong Kim, Hyunrok Lee, and Hyungki Choi, for their everlasting guidance in life and study of ICU.

Most of all, I should mention my father and mother for their endless concerns and devotional affection. I cannot forget their trust and encouragement on me. My sister and her husband also have given me warmhearted concerns. I hope God bless my family and to be happy.

Finally, I will always remember the life of ICU. It filled up my poor knowledge and made me a grown-up person.

Curriculum Vitae

Name : Jaehyrk Park

Date of Birth : Jul. 29. 1975

Sex : Male

Nationality : Korean

Education

1994.3–2002.2	Computer Science Inha University (B.A.)
2002.2–2004.2	Cryptology and Information Security, Engineering Information and Communications University (M.S.)

Career

2003.8–	Graduate Research Assistant Ubiquitous System Security Technique Next Information Technology Zone(NITZ)
2003.1–2003.12	Graduate Research Assistant Research on Link Security Algorithm and Standardization Electronics and Telecommunication Research Institute(ETRI)

2002.4–2002.12 Graduate Research Assistant
Research on Easy Security Technology
Electronics and Telecommunications Research Institute(ETRI)

2002.2–2002.7 Graduate Research Assistant
Development of Electronic Voting System for World-Cup 2002
Information Research center for Information Security, ICU

2002.7–2002.8 Apprentice Researcher
Communication Science Laboratories(CSL), NTT, Japan

2002.2–2004.2 Graduate Research Assistant
Cultivation of Top Level IT Security Manpower
The Ministry of Information and Communications(MIC)

Academic Experience

2002.4– KIISC student member

Publications

(1) 2003.11 Jaehyrk Park, Seokkyu Kang, and Kwangjo Kim, Group Mutual Exclusion based Secure Distributed Protocol, *The 1st Computer Security Symposium 2003*, Kokura, Japan.

(2) 2003.7 Jaehyrk Park and Kwangjo Kim, 그룹상호배제 기반의 안전한 프로토콜, 2003년도 한국정보보호학회 학술대회, pp283-288, 배재대학교, 한국

(3) 2003.8 박재혁, 김광조, 유태환, 한경수, EPON의 보안요구사항과 QoSS의 적용, 2003년도 한국정보보호학회 충청지부 학술대회, pp.251-259, 충북대학교, 한국

(4) 2002.11 Jaehyrk Park, Yoshifumi Manabe, and Kwangjo Kim, Mutual Exclusion algorithm using Group Choice, 2002년도 한국정보보호학회 학술대회, pp.53-56, 항공대학교, 한국

(5) 2002.11 함우석, 박재혁, 이송원, 김종승, 최수길, 김광조, 김숙연, 남택용, QoSS의 연구 동향과 적용, 2002년도 한국정보보호학회 학술대회, pp.352-355, 항공대학교, 한국