Group Mutual Exclusion based Secure Distributed Protocol

Jaehyrk Park Sukkyu Gang Kwangjo Kim

International Research center for Information Security (IRIS),
Information and Communications University. (ICU),
58-4 Hwa-am Dong, Yuseong Gu, Daejeon, Korea 305-732

{eye2174, redorb, kkj}@icu.ac.kr

Abstract A quorum system is a collection of sets(quorums) every two of which have a nonempty intersection. Quorum systems have been used for a number of applications in the area of distributed systems. In this paper, we present a method of controlling the access to a secure database based on group quorum systems. Our protocol is based on mutual exclusion algorithm along with fault-tolerance property. Also, the security of algorithm is based on secret sharing scheme. Through presenting secure protocol keeping group mutual exclusion, we deal with the problems associated with integrating cryptography and distributed algorithm.

1 Introduction

Integration of security and distributed computing area has been dealt with by a few researchers. Distributed algorithm does not deal with the security itself. So, we suggest secure distributed algorithm based on group mutual exclusion. Let's consider the following scenario [3]. The music service company MusicBank has a large digitized music files. MusicBank customers buy the e-ticket to access a set of music file. The protocol should run using a widespread collection of access servers, which may be completely separate from the actual data servers. Note that servers may be unavailable due to crashes or communication failures, so the protocol needs to overcome this problem and allows the high availability of the service. In a distributed network, many customers have e-ticket of data they want to listen but they can not access music file at the same time since shared one can be used one by one. This mechanism is called mutual exclusion. Mutual exclusion guarantees an exclusive access to a common music file among a set of competing customers. So, it is worth considering both mutual exclusion for consistency of competing processes and the security of each process at the same time.

1.1 Our Approaches

We specify secure distributed algorithm by using both group mutual exclusion algorithm and secret sharing scheme(SSS). Even though we add SSS to distributed algorithm, communication complexity is not changed. However, our algorithm not only guarantees the requirements of mutual exclusion algorithm but also customers' privacy and authentication.

1.2 Organization

The rest of the paper is organized as follows: In Section 2, we introduce related works in brief. Section 3 presents our model and assumptions as preliminaries for the rest of our paper. Section 4 describes our proposed algorithm based on Quorum system in detail. The evaluation of the proposed algorithm is discussed in Section 5. We finalize this paper with conclusion in Section 6.

2 Related Work

In this Section, we provide some related work about distributed algorithm. The general architecture is omitted here. For the interested readers, refer to [1], [2], and [4].

2.1 Quorum based Algorithms for Group Mutual Exclusion

Group mutual exclusion is formulated by Joung in 2001 [1]. It allows a resource to be shared by processes of the same group, but requires processes of different groups to use the resource in a mutually exclusive style. A process requests a session that is a resource area which users want to use before entering its critical section(CS). Processes are allowed to be in the CS simultaneously provided they have requested the same session. An example of group mutual exclusion, described by Joung [1], is a CD juke box shared by multiple processes. Any number of processes can simultaneously access the currently loaded CD, but processes wishing to access different CD other than the currently loaded one must wait. In this case, the sessions are the CDs in the juke box.
Joung introduced the concept of m-group quorum system. Let $P = \{1, 2, ..., n\}$ be a set of nodes, which belong to m groups. Each node does not belong to groups, but each request belongs to groups.

Definition 1.(m-group quorum system) $\omega = (C_1, C_2, ..., C_m)$ *over* P *consists of* m *sets, where each* $C_i \subseteq 2^P$ *is a set of subsets of* P *satisfying the following conditions:*

Non-emptiness $\forall\ 1 \leq\ i \leq\ m, Q_i \neq \phi$

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

C_i is called a cartel, and each $Q \in C_i$ is a quorum. ω can be used to solve group mutual exclusion as follow: each process i of group j, when attempting to enter CS, must acquire permission from every member in a quorum $Q \in C_j$, Upon exiting CS, process i returns the permission to the members of the quorum. Suppose a quorum member gives permission to only one process at a time. Then, by the intersection property, no two processes of different groups can be in CS simultaneously. The minimality property is used rather to enhance efficiency.

2.2 Secret sharing Scheme using Quorum system

Naor and Wool suggested control method to access to a secure database via quorum systems [3]. SSS realizing the access structures of quorum systems is essential for their method. Their scheme used Quorum system in order to make their protocol to be fault-tolerant. The difference between their scheme and the proposed scheme is that we deal with both mutual exclusion of competing processes and security of algorithm.

Definition 2. (Secret sharing scheme) *Let* S *be a finite set of secrets. We say that a secret-sharing scheme* Π *realizes an* m*-group quorum system* ω *if* Π *is a mapping* $\Pi\ :\ S \times R\ \rightarrow\ S_1 \times S_2 \times \cdots \times S_n$ *, from the cross product of secrets and random strings to a set of shares such that the following two requirements hold:*

1 *The secret can be reconstructed by any subset in* ω *. That is, for every secret* $s \in S$ *and set* $A \in \omega$ $(A = i_1, \cdots, i_{|A|})$ *there exists a function* $h_A : S_{i_1} \times \cdots \times S_{|A|} \rightarrow S$ *such that for every random string* r*, if* $\Pi(s, r) = s_1, ..., s_n$ *then* $h_A(\{S_i\}_{i \in A})\ =\ s$.
2 *Every subset not in* ω *can reveal any partial information about the secret. Formally, for any subset* $Z\ \nsubseteq\ \omega$*, for every two secrets* $a, b \in S$*, and for every possible collection of shares* $\{s_i\}_{i \in Z}$*:* $P(\{s_i\}_{i \in Z}|a)\ = P(\{s_i\}_{i \in Z}|b)$*, where the probability is taken over the random string* r.

3 Our Model

3.1 Attack

Before we explain our model, we must consider possible attacks. In group mutual exclusion algorithm, processes who want same data can access to file, even though process is 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. *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.

3.2 Assumptions and Notations

Our scheme can be regarded as an improvement of the scheme [2]. 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(U) who want to access data. Since we use SSS, reconstruction function, h_A, for secret information and polynomial function, $\prod_i(k, r)$ 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.

4 Our Proposed Algorithm

We show how our algorithm can be proceeded here. Because of space constraints, we omit some procedure needed for mutual exclusion. The detailed algorithm will be provided in the full paper.

4.1 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.

4.2 Commitment Phase

The algorithm is shown in Figure 1.

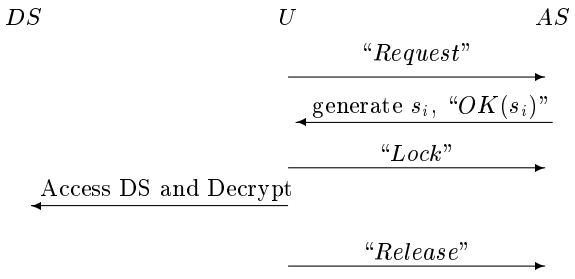

Figure 1: Sketch of our Algorithm

We describe the whole phase in brief. U who wants to access data requests to AS. AS check whether U is authorized person or not through looking up customers' list directory. After that, AS generate k, r and calculate s_i

using SSS. AS send "$OK(s_i)$" or "$Enter(g, s_i)$" message to U. After collecting all s_i from each AS in a set of quorum, U can get key SK. U send "$LOCK$" message to each AS to inform that U will access data. U can decrypt an encrypted data and show data content. Also, U who wants to access the same data can access data.

4.3 Proposed Algorithm

Our algorithm consists of two parts. AccessServer is for the behavior of AS that as a quorum member. RequestUser is for the behavior of a user that acts as a group member.

RequestUser Procedure

□var $Rstatus = wait$: status of request;
$Q = \phi$: set of process;
K : set of process;
G : set of group;
p : request process;
X : data part that user want;
T : set of partial secret value;
q : server process;
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 "$Request(G, p, X)$" 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 $q \in Q$;
$SK = h_A(\{S_i\}_{i \in A})$;
$Rstatus := in$;
$y(x) = D(x) \otimes SK$;
$Rstatus := out$;
send "$Release$" to all $q \in Q$;
$K := \phi$;
end /* end of $K = Q$ */
end /* end of $Rstatus = wait$ */
end /* end of arrival "$OK(s_i)$" */

1. When User p whose group set is $G(p)$ wants to get data, he sends "$Request(G, p, X)$" to all server processes in one quorum Q.

2. There are three cases to get data.(1)When U receives "$OK(s_i)$"from every process in Q, U arbitrary selects one group and get secret key using reconstruction function, $SK = h_A(\{S_i\}_{i \in A})$. (2)$U$ sends "$Lock(g)$" to every process in Q and enter CS, With the key he can decrypt data, $y(x) = D(x) \otimes SK$. (3)When U receives "$Enter(g, s_i)$" from some process in Q, U sets g as group selection and enters CS.

3. When exiting from CS, U sends "$Release$" to every process in Q.

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 "$Request(G, p, X)$" from p
begin
/*check authorization*/
if $p \in$ Registration List then begin
Insert "$Request(G, p, X)$" 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 k with encryption */
/* random function $Key_\alpha(X)$ */
$k = Key_\alpha(X)$;
/* generate pseudo random string */
/* with a private seed rr */
$r = R_{rr}(X \otimes p)$;
/* compute s_i using the SSS*/
$s_i = \prod_i(k, r)$;
send "$OK(s_i)$ to p ;

$sentok := p$;
$Que[i].status := waitlock$;
$status := waitlock$;
end /*end of $status = vacant$ */
end /*end of $p \in$ Registration List */
end /*end of arrival "$Request(G, p, X)$" */

1. When q receives "$Request(G, p, X)$" from U, q inserts it to the queue Que. Using identity of U and X, it checks the authorization.

2. If the request is from an authorized user , AS doesn't give the permission to other process, and request process belongs to the same group, the server generates $k = Key_\alpha(X)$ and a pseudo-random string $r = R_{rr}(X \otimes p)$. Server i then computes its share of the key, $s_i = \prod_i(k, r)$ using the SSS, and send "$OK(s_i)$" to U.

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

4. When q receives "$Release$", q stops further sending of "$Enter(g, s_i)$". And if there is no process to which "$Enter(g, s_i)$" is sent, q replies "$Finished$".

The meaning of variables used in two Algorithm are as follows. As for each RequestUser process, $Rstatus$ stores the status of the request. $RStatus = wait$ means it is waiting for "$OK(s_i)$" or "$Enter(g, s_i)$". In means that it is in the CS, out means that it has exited from CS. The quorum currently used is stored in Q. The set of processes from which "$OK(s_i)$" 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 AccessServer 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 "$OK(s_i)$" is sent and waiting for "$Lock$" from the process. $enter$ when the process is entering CS. $releasing$ when the process is releasing. $waitrcancel$ when "$Cancel$" is sent and waiting for the reply. Each AccessServer process sends "$OK(s_i)$" to at most one request at any time. The RequestUser process to which "$OK(s_i)$" is sent is stored in variable $sentok$. $status$ 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. Also, variable $status$ stores the current status of the process.

5 Evaluation

5.1 Security and Mutual Exclusion

[1] and [2] proposed distributed algorithm satisfying requirement of group mutual exclusion. However, these papers focus on the consistency of competing processes which are the issue of distributed computing area. [3] also uses quorum system for fault-tolerance of each server. However, they consider security of distributed protocol, not avoidance of process conflict. We presented an algorithm keeping both the property of group mutual exclusion and secure algorithm satisfying confidentiality and authentication.

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 Even though some parties happen fault, other correct server can give service to user.

Table 1: Comparison of Algorithms

	[1]	[2]	[3]	Ours
Privacy	X	X	O	O
Authentication	X	X	O	O
Unforgeability	X	X	O	O
Availability	O	O	O	O
Consistency	O	O	O	O
Mutual exclusion	O	O	X	O
Starvation-freeness	O	O	X	O
Deadlock-freeness	O	O	X	O

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.

Group mutual exclusion Using group quorum system, customers who request different data cannot access data, but customer that have requested the same data can.

Starvation-freeness The customers who want to access different data should be able to access data eventually.

Deadlock-freeness The customers who want to access different data do not be waiting for other data permanently.

5.2 Performance

Table 2 shows the performance comparison of main computation and communication overhead of the various algorithms. The main computation complexity of [1] and [2] is $O(T)$. Our algorithm can be regarded as an improvement of the scheme [2]. In proposed algorithm, computation of each partial secret information is pre-computed during registration phase. So, computation complexity of proposed algorithm is the same as [1] and [2].

The communication complexity of the proposed algorithm is shown. Let $|Q|$ be the size of the smallest quorum in a coterie. In case of the worst case complexity, the total number of messages per request is $9|Q|$. The worst case number of messages is larger than $2c+1$ in [1]. Additional cryptographic technique does not affect the total number of messages. So, the communication complexity of our algorithm is $9|Q|$.

Table 2: Performance Comparision

	Computation	Communication
[1]	$O(T)$	$2c+1$
[2]	$O(T)$	$9\|Q\|$
Ours	$O(T)$	$9\|Q\|$

6 Conclusions

We suggested a secure distributed algorithm that guarantees not only group mutual exclusion avoiding conflict of each process but also security requirements even though performance complexity is almost same. In the future, we will do research more secure and practical distributed algorithm not just distributed algorithm that focuses on consistency of each process.

References

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

[2] Y.Manabe and J.Park, A Quorum based group mutual exclusion algorithm without unnecessary blocking, submitted to SRDS 2003.

[3] 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.

[4] V.Hadzlilacos, A Note on Group Mutual Exclusion, *In Proc. 20th PODC*, pp 189-206, 2000.

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

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