Design and Development of a Unified framework for Secure Group Communications

Nagesh H.R[†], *K. Chandra Sekaran*[††]

[†]*Department of Computer Engineering, P.A. College of Engineering, Mangalore, Karnataka, INDIA*
[††]*Department of Computer Engineering, National Institute of Technology Karnataka, Surathkal, Karnataka, INDIA*

Summary
Many emerging network applications (e.g. teleconference, information services, distributed interactive simulation and collaborative network) are based on a group communications model. As a result, securing group communications, i.e., providing confidentiality, authenticity, and integrity of messages delivered between group members, will become networking issue. A secure group is a triple (U, K, R) where U denotes a set of users, K a set of keys held by the users, and R is a user-key relation. Secure groups are specified using key graphs. Three rekeying strategies: user-oriented, key-oriented, and group-oriented for securely distributing rekey messages after a join or leave are designed. Protocols for joining and leaving secure groups are designed and implemented. The rekeying strategies and join or leave protocols are implemented in a key server. This paper deals with design and development of a unified framework for secure group communications which implements join/leave protocols for all three rekeying strategies. The strategy is worked on client/server basis in a hierarchical fashion, structured as a tree with the server at the root and groups forming the nodes ending up in clients. The height of the tree has been kept as constant (=3). Because of this height, join and leave operations will become easier. Public key cryptosystem has been used for encryption, decryption of rekey messages, and original messages (key length=1024-bits). Behavior of the system for user-oriented, key-oriented, and group-oriented rekeying strategies is experimented and reported in this paper. Our implementation of secure group communications provides authentication, confidentiality and integrity of the messages delivered between the group members.

Key words:
Confidentiality, Group communications, Group key management, Key distribution, Multicast, Privacy, Rekeying, Security.

1. Introduction

Most network applications are based on the client-server paradigm and make use of unicast packet delivery. Many emerging applications on the other hand, are based upon a group communications model. In particular, they require packet delivery from one or more authorized sender(s) to a large number of authorized receivers. In the Internet, multicast has been used successfully to provide an efficient, best effort delivery service to large groups [2]. Securing group communications, i.e., providing confidentiality, authenticity, and integrity of messages delivered between members, will become a critical networking issue.

With a hierarchy of keys, there are many different ways to construct rekey messages and securely distribute them to users. Three rekeying strategies are used: user-oriented, key-oriented, and group-oriented. Join/leave protocols will be designed based upon these three rekeying strategies. Key-oriented and user-oriented rekeying, use multiple rekey messages, whereas group-oriented rekeying uses a single rekey message [8].

In [1] implementation of the three rekeying strategies and protocols for larger group size with more number of levels with DES 56-bit secret key algorithm has been done. In case of DES algorithm only one secret key is used for communication between the sender and receiver, which will be sent through network. This will reduce the security of the system. Hence public key algorithm (RSA) [6] is used in this paper. Public key algorithm uses two key pairs for communication. One is public key pair and another is private key pair. Only public key pair is sent through the network, which is known to everybody. Private key pair will not be sent in the network. This paper uses key length of 1024-bits. The increase in key length and the use of public key crypto system will increase the security of the system.

2. Secure Groups

This section deals with secure group, key graph, and different rekeying strategies: user-oriented, key-oriented, and group-oriented. It also deals with join/leave protocols for all three rekeying strategies.

A *secure group* is a triple (U, K, R) where:

- U is a finite and nonempty set of users,
- K is a finite and nonempty set of keys,
- R is a binary relation between U and K, $R \subset U \times K$,

called the *user-key* relation of the secure group. User *u* has key *k* if and only if (*u*, *k*) is in *R*. Each secure group has a trusted *key server* responsible for generating and securely distributing keys in *K* to users in the group. Specifically, the key server knows the user set *U* and the key set *K* and maintains the user-key relation R. Every user in *U* has a key in *K*, called its *public key*, which is sent by the user to server for pairwise confidential communication with the key server. There is a *group key* in *K*, shared by the key server and all users in *U*. The group key can be used by each user to send messages confidentially to other members of the group [10] [11].

2.1 Key Graphs

A key graph is a directed acyclic graph *G* with two types of nodes: *u*-nodes representing users and *k*-nodes representing keys. Each *u*-node has one or more outgoing edges but no incoming edge. Each *k*-node has one or more incoming edges. If a *k*-node has incoming edges only and no outgoing edge, then this k-node is called a root.

Given a key graph *G*, it specifies a secure group (*U*, *K*, *R*) as follows.

1) There is a one-to-one correspondence between *U* and the set of *u*-nodes in G.

2) There is a one-to-one correspondence between *K* and the set of *k*-nodes in *G*.

3) (*u*, *k*) is in *R* if and only if *G* has a directed path from the *u*-node that corresponds to *u* to the *k*-node that corresponds to *k*. As an example, the key graph in Fig. 1 specifies the following secure group:

$U = \{u_1, u_2, u_3, u_4\}$

$K = \{k_1, k_2, k_3, k_4, k_{12}, k_{234}, k_{1234}\}$

$R = \{(u_1, k_1), (u_1, k_{12}), (u_1, k_{1234}), (u_2, k_2), (u_2, k_{12}), (u_2, k_{234}), (u_2, k_{1234}), (u_3, k_3), (u_3, k_{234}), (u_3, k_{1234}), (u_4, k_4), (u_4, k_{234}), (u_4, k_{1234})\}$.

Associated with each secure group (*U*, *K*, *R*) are two functions, keyset() and userset(), defined as follows:

keyset(u) = $\{k \mid (u, k) \in R\}$

userset (k) = $\{u \mid (u, k) \in R\}$.

Intuitively, keyset(*u*) is the set of keys that are held by user *u* in *U*, and userset(*k*) is the set of users that hold key *k* in *K*. For examples, referring to the key graph in Fig. 1, we have keyset(u_4) = $\{k_4, k_{234}, k_{1234}\}$ and userset(k_{234}) = $\{u_2, u_3, u_4\}$.

Generalized definition of function keyset() to any subset *U* ' of *U*, and function userset() to any subset *K*' of *K*, in a straightforward manner, i.e., keyset(*U* ') is the set of keys each of which is held by at least one user in *U* ', and userset(*K* ') is the set of users each of which holds at least one key in *K* '.

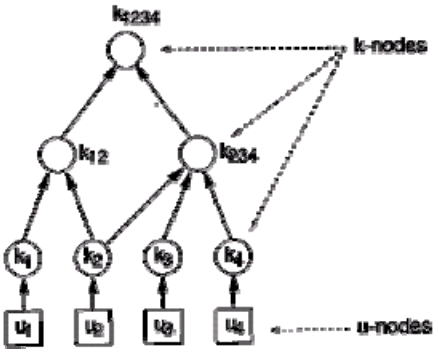

Fig. 1 A key Graph

When a user *u* leaves a secure group (*U*, *K*, *R*), every key that has been held by *u* and shared by other users in *U* should be changed. Let *k* be such a key. To replace *k*, the server randomly generates a new key k_{new} and sends it to every user in userset(*k*) except *u*. To do so securely, the server needs to find a subset *K'* of keys such that userset (*K* ') = userset(*k*) - {*u*} and use keys in *K* ' to encrypt k_{new} for distribution [10] [11].

2.2 Rekeying Strategies and Protocols

In this section, we illustrate three rekeying strategies: user-oriented, key-oriented, and group-oriented for a tree key graph and the protocols used when a new user joins/leaves the group.

A user *u* who wants to join (leave) a secure group sends a join (leave) request and the public key pair to the server, denoted by *s*. A join request initiates an authentication exchange between *u* and *s*. If user *u* is not authorized to join the group, server s sends a join-denied reply to *u*. If the join request is granted, then the user sends its public key pair to the server in response to authentication exchange [3, 4, 5]. The key pair generated by the user will be used as the individual key k_u, of *u*.

$s \Leftrightarrow u$: authenticate *u* and store public key pairs of *u*

to represent the authentication exchange between server *s* and user *u*.

After each join or leave, a new secure group is formed. Server *s* has to update the group's key graph by replacing the keys of some existing *k*-nodes, deleting some *k*-nodes (in the case of a leave), and adding some *k*-nodes (in the case of a join). It then securely sends rekey messages containing new group/subgroup keys to users of the new secure group.

2.3 Joining a Tree Key Graph

After granting a join request from *u*, server *s* creates a new *u*-node for user *u* and a new *k*-node for its individual key k_u. Server s finds an existing *k*-node (called the *joining point* for this join request) in the key tree and attaches *k*-node k_u to the joining point as its child.

To prevent the joining user from accessing past communications, all keys along the path from the joining point to the root node need to be changed. After generating new keys for these nodes, server s needs to securely distribute them to the existing users as well as the joining user. For example, as shown in Fig. 2, suppose u_9 is granted to join the upper secure group in the figure. The joining point is *k*-node k_{78} in the upper key graph, and the key of this *k*-node is changed to k_{789} in the lower key graph. Moreover, the group key at the root is changed from $k_{1\text{-}8}$ to $k_{1\text{-}9}$. Users u_1, . . ., u_6 only need the new group key $k_{1\text{-}9}$, while users u_7, u_8, and u_9 need the new group key $k_{1\text{-}9}$ as well as the new subgroup key k_{789}.

To securely distribute the new keys to the users, the server constructs and sends rekey messages to the users. A *rekey message* contains one or more encrypted new key(s), and a user needs to decrypt it with appropriate keys in order to get the new keys. Three different approaches are used to construct and send the rekey messages. They are user-oriented, key-oriented, and group-oriented rekeying.

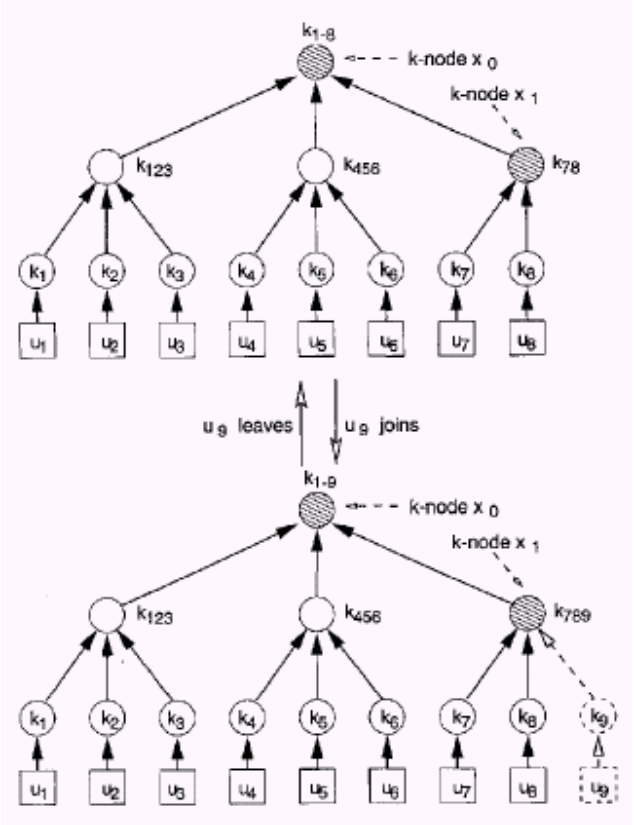

Fig. 2 A key tree graph before and after a join (leave).

2.3.1 User-Oriented Rekeying

Consider each user and the subset of new keys it needs. The idea of user-oriented rekeying is that for each user, the server constructs a rekey message that contains precisely the new keys needed by the user and encrypts them using a key held by the user. For example, as shown in Fig. 2, for user u_9 to join the upper secure group in the figure, server *s* needs to send the following three rekey messages:

$$s \longrightarrow \{u_1,\ldots,u_6\}: \{k_{1\text{-}9}\}k_{1\text{-}8}$$
$$s \longrightarrow \{u_7,u_8\} : \{k_{1\text{-}9}, k_{789}\}k_{78}$$
$$s \longrightarrow u_9 : \{k_{1\text{-}9}, k_{789}\}k_9.$$

Note that users $u_{1,\ldots,}u_6$ need to get the new group key $k_{1\text{-}9}$. There is no single key that is shared only by $u_{1,\ldots,}u_6$. However, key $k_{1\text{-}8}$ can be used to encrypt the new key $k_{1\text{-}9}$ for $u_{1,\ldots,}u_6$ without security breach since users u_7 and u_8 will also get this new group key from another rekey message.

User-oriented rekey messages can be constructed as follows. For each *k*-node *x* whose key has been changed, say, from *k* to *k*', the server constructs a rekey message by encrypting the new keys of *k*-node *x* and all its ancestors (up to the root) by the old key *k*. This rekey message is then sent to the subset of users that need precisely these new keys. Either *unicast* or *subgroup multicast* may be used. Moreover, one rekey message is sent to the joining user, which contains all of the new keys encrypted by the individual key of the joining user. This approach needs *h*-rekey messages [9] (h-height of the tree).

2.3.2 Key-Oriented Rekeying

In this approach, each new key is encrypted individually (except keys for the joining user). For each *k*-node *x* whose key has been changed, say, from *k* to *k*', the server constructs two rekey messages. First, the server encrypts the new key *k*' with the old key *k* and sends it to userset(*k*), which is the set of users that share *k*. All of the original users that need the new key *k*' can get it from this rekey message. The other rekey message contains the new key *k*' encrypted by the individual key of the joining user and is sent to the joining user.

As described above, a user may have to get multiple rekey messages in order to get all the new keys it needs. For example, as shown in Fig. 2, for user u_9 to join the upper secure group in the figure, server *s* needs to send the following four rekey,

$$s \longrightarrow \{u_1, \ldots ,u_8\} : \{k_{1\text{-}9}\}k_{1\text{-}8}$$
$$s \longrightarrow u_9 : \{k_{1\text{-}9}\}k_9$$
$$s \longrightarrow \{u_7,u_8\} : \{k_{789}\}k_{78}$$
$$s \longrightarrow u_9 : \{k_{789}\}k_9$$

This approach requires 2(h-1) rekey messages.

2.3.3 Group-Oriented Rekeying

In key-oriented rekeying, each new key is encrypted individually (except keys for the joining user). The server constructs multiple rekey messages, each tailored to the needs of a subgroup. Specifically, the users of a subgroup receive a rekey message containing precisely the new keys each needs. An alternative approach, called group-oriented, is for the server to construct a single rekey message containing all new keys. This rekey message is then multicasted to the entire group. Clearly, such a rekey message is relatively large and contains information not needed by individual users. The group-oriented approach has several advantages over key-oriented and user-oriented rekeying. First, multicast can be used instead of unicast or subgroup multicast. Second, with fewer rekey messages, the server's per rekey message overheads are reduced.

For example, as shown in Fig. 2, for user u_9 to join the upper secure group in the figure, server s needs to send the following two rekey messages; one is multicasted to the group and the other is unicasted to the joining user:

$$s \longrightarrow \{u_1,\ldots,u_8\}: \{k_{1\text{-}9}\}k_{1\text{-}8}\ , \{k_{789}\}k_{78}$$
$$s \longrightarrow u_9 \quad : \{k_{1\text{-}9}\ , k_{789}\}k_9$$

This approach reduces the number of rekey messages to one multicast message and one unicast message.

2.4 Leaving a Tree Key Graph

After granting a leave request from user *u*, server *s* updates the key graph by deleting the *u*-node for user *u* and the *k*-node for its individual key from the key graph. The parent of the *k*-node for its individual key is called the *leaving point*. To prevent the leaving user from accessing future communications, all keys along the path from the leaving point to the root node need to be changed. After generating new keys for these *k*-nodes, server *s* needs to securely distribute them to the remaining users. For example, as shown in Fig. 2, suppose u_9 is granted to leave the lower secure group in the figure. The leaving point is the *k*-node for k_{789} in the lower key graph, and the key of this *k*-node is changed to k_{78} in the upper key graph. Moreover, the group key is also changed from $k_{1\text{-}9}$ to $k_{1\text{-}8}$. Users $u_1, \ldots, u_6$ only need to know the new group key $k_{1\text{-}8}$. Users u_7 and u_8 need to know the new group key $k_{1\text{-}8}$ and the new subgroup key k_{78}. To securely distribute the new keys to users after a leave, same rekeying strategies are used.

2.4.1 User-Oriented Rekeying

In this approach, each user gets a rekey message in which all the new keys it needs are encrypted using a key it holds. For example, as shown in Fig. 2, for user u9 to leave the lower secure group in the figure, server s needs to send the following four rekey messages:

$$s \longrightarrow \{u_1,u_2,u_3\} \quad : \{k_{1\text{-}8}\}k_{123}$$
$$s \longrightarrow \{u_4,u_5,u_6\} \quad : \{k_{1\text{-}8}\}k_{456}$$
$$s \longrightarrow u_7 \quad : \{k_{1\text{-}8,}\ k_{78}\}k_7$$
$$s \longrightarrow u_8 \quad : \{k_{1\text{-}8}\ ,\ k_{78}\}k_8.$$

User-oriented rekey messages for a leave can be constructed as follows. For each *k*-node *x* whose key has been changed, say, from *k* to *k'*, and for each unchanged child *y* of *x*, the server constructs a rekey message by encrypting the new keys of *k*-node *x* and all its ancestors (up to the root) by the key *K* of *k*-node *y*. This rekey message is then multicasted to userset(*K*) [8].

2.4.2 Key-Oriented Rekeying

In this approach, each new key is encrypted individually. For example, as shown in Fig. 2, for user u_9 to leave the lower secure group, server s needs to send the following four rekey messages:

$$s \longrightarrow \{u_1,u_2,u_3\} \quad : \{k_{1\text{-}8}\}k_{123}$$
$$s \longrightarrow \{u_4,u_5,u_6\} \quad : \{k_{1\text{-}8}\}k_{456}$$
$$s \longrightarrow u_7 \quad : \{k_{1\text{-}8}\}k_{78,}\ \{k_{78}\}k_7$$
$$s \longrightarrow u_8 \quad : \{k_{1\text{-}8}\ \},k_{78,}\{k_{78}\}k_8.$$

2.4.3 Group-Oriented Rekeying

A single rekey message is constructed containing all new keys. For example, as shown in Fig. 2, for user u_9 to leave the lower secure group in the figure, server s needs to send the following rekey message:

let L_0 denote $\{k_{1\text{-}8}\}k_{123}$, $\{k_{1\text{-}8}\}k_{456}$, $\{k_{1\text{-}8}\}k_{78}$

let L_1 denote $\{k_{78}\}k_7$, $\{k_{78}\}k_8$

$$s \longrightarrow \{u_1,\ldots,\ u_8\}: L_0,L_1.$$

Note that for a leave, this single rekey message is about *d* times bigger than the rekey message for a join, where *d* is the average degree of a *k*-node. This approach uses only one rekey message, which is multicasted to the entire group [8].

3. UML State Diagram

A state diagram (also called a state chart diagram) shows the sequence of states that an object goes through during its life in response to outside stimuli and messages. The state is the set of values that describes an object at a specific point in time and is represented by state symbols and arrows connecting the state symbols represent the transitions. A state diagram represents the state of the method execution (that is, the state of the object executing the method), and the activities in the diagram represent the activities of the object that performs the method. The purpose of the state diagram is to understand the algorithm involved in performing a method. The Fig. 3 and Fig. 4 represent the state diagram of a server and a user (client) respectively (see after author biography).

The *Start* object initiates the server program and accepts the number of subgroups to be created from the

administrator. The *Start_server* object activates the server program and generates the public and private key pairs for the server, which will be stored in the Oracle database. It also creates the given number of subgroups with public and private key pairs and stores them in the database along with subgroupID. The *wait* object waits for the connection request (join/leave/message) from the user. If the request is join request, then the server authenticates the user with username and password by comparing it with the already stored database of different users. If the user is a valid user then the public key pairs of the user is stored in the server database. If the user is an invalid user then the server sends the message "invalid user name or password" to the user.

The *Add_to_subgroup* object randomly generates the subgroup number to which the new user is going to join. Then the user will be added to that randomly generated subgroup. The *key_updation* object regenerates the public and private key pairs for the server and the public and private key pairs for the subgroup to which the new user is joined. The *Rekeying* object generates the rekey message that contains the new keys needed by different users depending on the rekeying strategy, which was discussed in chapter 3. *Sending_rekey_message* object distributes rekey message to different users.

The *Receive_message* object receives the encrypted message with the users public key along with destination address. Then it checks whether the destination user is active or inactive. If the destination user is inactive then the message "Destination user in not logged in" will be sent to that user. If the destination user is active then it calls *Decrypt_and_encrypt* object, which will decrypt the message by using sender's public key and then encrypts it with destination user's public key and then it calls *Send_message* to the destination user.

The *Start* object will initiate the client program. The *Start_user* object generates the public and private key pairs for the user and stores them in the Oracle database. It calls *Connect* object, which will check whether server is reachable. If the server is not reachable, then the message "Server not started" will be received by the user. If the server is reachable then it will calls *login* object.

The *login* object accesses the public key pair stored in the database and accepts the username and password. It sends all these information to the server for authentication. Once the user gets logged in, then if the request is for sending message, then it will call *Message* object. The *Message* object accepts the message and the destination address, which encrypts the message with user's private key pair and stores the message in the database. Then the *Send_message* object will be invoked, which will access the encrypted message from the database and sends the message to the specified destination. If the *login* object calls *Disconnect* object, it will disconnects the user from the server.

4. Experiments and Performance Comparisons

We have designed and constructed a separate group key server, as well as a client, which implement join/leave protocols for all three rekeying strategies: user-oriented, key-oriented, and group-oriented. The experiments are carried out on Pentium-III, 128MB RAM, 700 MHz machine. The software's used for conducting the experiments are: Visual Basic 6.0, JDK 1.3 [7], Oracle8i client-server version, Windows NT workstations/ Windows 98 workstations. The server is initialized from number of groups to be created. When a user sends a join request it generates the random group number for that user and attaches the user to that group.

The system we have implemented provides following security services:

***Confidentiality*:** It is the protection of transmitted data from passive attacks, that is, it ensures that the information in a computer system and transmitted information are accessible for reading by authorized parties. One method to achieve this confidentiality is to encrypt data before transmitting it at the sending end and to decrypt the received message at the receiving end. We have implemented the RSA algorithm to preserve the confidentiality of the messages transmitted within the group. All messages transmitted within the group are encrypted using the group key that is shared by all users in the group. Also the message transferred between the clients and the server is encrypted using the user's individual key or session key. The confidentiality of the rekey messages is also preserved using the same technique. By this approach, only the authorized users will be able to decrypt the message, since they will possess the key required to decrypt the message.

Authenticity: The authentication service is concerned with assuring that a communication is authentic. In the case of a single message, the function of the authentication service is to assure the recipient that the message is from the source that it claims to be from. Our implementation supports basic authentication service of checking the user name and passwords. The server maintains a database of all users in the Oracle database, which consists of all users' user name and password. Every user who wishes to join the group provides a user name and password to the server, which verifies them against a database of user details. The confidentiality of this authentication messages is maintained using the RSA algorithm [6], which is a public key cryptosystem.

Integrity: It ensures that only authorized parties are able to modify the transmitted information. There is a very critical necessity that the clients are able to confirm that the rekey messages were actually sent by the server and not by any other unauthorized processes. To implement such checks, we use MD5 message digest algorithm in the following

way: when sending a rekey message the server computes the message digest of the rekey message and sends it to the respective client encrypted with the server's private key along with the actual rekey message. On the receiving end, the client recomputes the message digest on the received rekey message, decrypts the message digest that the server had sent using the server's public key and compares both of them. If both the message digests are equal, the client accepts the rekey message and the client's keys are updated. Otherwise the rekey message is discarded, as it is not a message from the authorized server. The Fig. 5 and Fig. 6 represent the implementation of these services at server and client side.

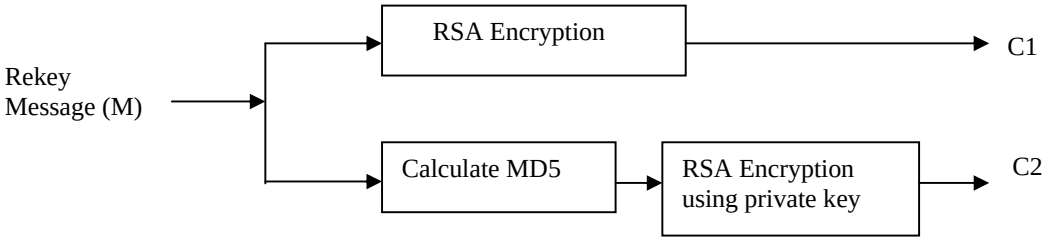

Fig. 5 Integrity check at server side

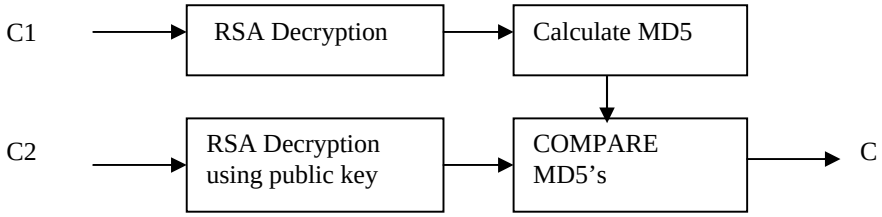

Fig. 6 Integrity check at client side

To evaluate the performance of different rekeying strategies, we have measured rekey message sizes (in bytes) and processing time (in seconds) used by the server per join/leave request. Specifically, the processing time per join/leave consists of the following components:

i) Time required for generating the new key pairs.
ii) Time required for encryption of new keys and constructing rekying messages.

We also measured size of rekey messages received by clients for join/leave. Table 1 presents the number and size of rekey messages with encryption sent by the server. Table 2 presents the size of rekey message received by different clients per join. Table 3 presents the size of rekey message received by different clients per leave. The rekey message size received by different users is more in group-oriented when compared to key-oriented and user-oriented. So the time required by the user to decrypt the rekey message for getting its new keys is more in group-oriented when compared to user-oriented and key-oriented. Table 4 presents the server processing time per request for 32 and 64 users. Processing time required by the server when a new user joins the group is more in user-oriented and key-oriented when compared to group-oriented. So if we use group-oriented rekeying users will get quicker response when a new user joins the group.

Table 1: Number and size of rekey messages sent by the server

Rekeying strategies	Rekey message size (bytes) Per join			No. of rekey messages Per join
	Others	Current	new	
User-oriented	7296	4256	608	3
Key-oriented	7296	4256	608	4
Group-oriented	18848	-----	608	2

Table 2: Size of rekey messages received by clients for join (32 clients)

	Join (message size in bytes)		
	New user	Other users in the new users group	Users of other subgroup
User-oriented	608	608	304
Key-oriented	608	608	304
Group-oriented	608	----	608

Table 3: Size of rekey messages received by clients for leave (32 clients)

Rekeying strategies	Leave (message size in bytes)	
	Users in the other group	Users in the current group
User-oriented	304	608
Key-oriented	304	608
Group-oriented	Other users	
	18848	

Table 4: Server processing time in seconds

No. of users	Rekeying strategies	Join (sec)	Leave (sec)
32	User-oriented	16	29
	Key-oriented	15.59	29
	Group-oriented	14.3	30
64	User-oriented	17	30
	Key-oriented	16.2	30
	Group-oriented	14.3	30

5. Conclusion

We have experimented three rekeying strategies: user-oriented, key-oriented, and group-oriented and the protocols when a user joins/leaves the group, where one user can join only one group at a time. We have measured number and size of rekey messages sent by the server, size of rekey messages received by clients for join/leave and server processing time. We have used height of the tree as three.

From the experimental results on the server side, group-oriented rekeying provides the best performance, with key-oriented rekeying in second place, and user-oriented rekeying in the third place. At the client side size of the rekey message received by the client is more in group-oriented, which requires more time for decryption and it is less in case of user-oriented and key-oriented. So client requires less time for decrypting the rekey message. The number of rekey messages generated by the server is less in case of group-oriented rekeying when compared to user-oriented and key-oriented.

The model we have developed can be easily extended to other applications such as teleconferencing, distributed interactive simulation, information services like instant news service or share market related application and any other application that involves secure communication between groups of users. We have used RSA algorithm of key length of 1024 bits. Time required to hack the key is more (300,000,000,000 MIPS years) as factoring method

is used to hack the key. Because of this key length, communication that takes place between the group members is more secure. Disadvantage of this system is that it takes more time when compared with DES. Speed compromise should be made to achieve more security. If a faster system such as Sun Ultra Sparc is used, we might get better speed. The other disadvantage of this system is that it uses only tree of height three. Hence as the number of users increases the degree of the node increases, in turn increasing the size of rekey messages. Since the height of the tree is three, joining and leaving operations will become easier. As a whole, this system can be used for a small organization for secure group communication.

References

[1] C.K. Wong, M. Gouda, and S.S. Lam, *"Secure Group Communications Using Key Graphs,"* IEEE/ACM Trans. Networking, February 2000.

[2] S.E.Deering, *"Multicast routing in internetworks and extended LANs,"* in Proc.ACM SIGCOMM'88, Aug. 1988, pp.55-64.

[3] A.O. Freier, P.Karlton, and P.C.Kocher, The SSL Protocol Version 3.0, 1996. Work in progress, Netscape Communication.

[4] J.G. Steiner, C. Neuman, and J. I. Schiller, *"Kerberos: An authentication service for open network systems,"* in Proc. USENIX Winter Conf., Feb. 1988, pp. 191-202.

[5] T.Y.C. Woo, R. Bindignavle, S. Su, and S.S Lam, *"SNP: An interface for secure network programming,"* in Proc. USENIX'94 Summer Technical Conf., Boston, M.A, June 1994.

[6] William Stallings, *"Cryptography and Network Security: Principles and Practice,"* Second Edition, Prentice Hall.

[7] http://vsys-www.infomatic.uni-hamburg.de /documents/docs /jdk1.3/api /java

[8] Nagesh H.R, K. Chandra Sekaran, K.M Hebbar *"Design, Implementation and Performance Analysis of Secure Group Communications"*, Proceedings of the 15th International Conference on Computer Communications, Bombay, India, Aug. 2003, Pp.880-890, International Council for Computer Communication, Washington, DC, USA

[9] Nagesh H.R, K. Chandra Sekaran, Niranjan N. Chiplunkar *"SURAKSHA: An Implementation of Secure Group Communications"*, Proceedings of the 12th International Conference on Advanced Computing and Communications, Ahamadabad, India, Dec. 2004, pp. 286-294.

[10] Nagesh H.R, K. Chandra Sekaran, Niranjan N. Chiplunkar *"Design, Implementation and Analysis of Secure Group Communications Using Key Graphs "*, Proceedings of the National Conference on Recent Trends in Networking Technology, Coimbature, India, Dec. 2003, pp. 180-190.

[11] Nagesh H.R, K. Chandra Sekaran *"An Efficient Key Agreement Protocol for Group Communications"*, Proceedings of the National Conference on Distributed Computing, India, March. 2004, pp. 188-193.

Acknowledgement

The authors wish to thank the anonymous reviewers for their constructive comments.

Biography:

Nagesh H.R received the B.E. and M.Tech. degrees, from Mangalore Univ. in 1996 and 2002, respectively. After working as a lecturer (from 1996), an assistant professor (from 2003) in the Dept. of Computer Science and Engineering, the NMAM Institute of Technology, and an associate professor (from 2005) , a Professor (from 2007) in the Dept. of Computer Science and Engineering , the P.A. College of Engineering., India. His research interest includes computer networks, cryptography, network security and distributed computing. He has published more than 20 publications in International and National proceedings.

K.Chandra Sekaran is a Professor of Computer Engineering at National Institute of Technology Karnataka, India. His research includes Computer Networks, Dependable Network / Distributed computing, Autonomic computing and Community Informatics. He has 20 years of teaching and research and one year Industry experience. He has published more than 86 publications in International and National proceedings and authored two books. He was the Organizing Chair of 14th International Conference ADCOM 2006, International Symposium on Ad Hoc and Ubiquitous Computing ISAHUC'06. He also served as a member of PC in various International conferences, reviewer in many Journals. He has supervised sponsored projects and IT consultant to some corporates in this region of India.

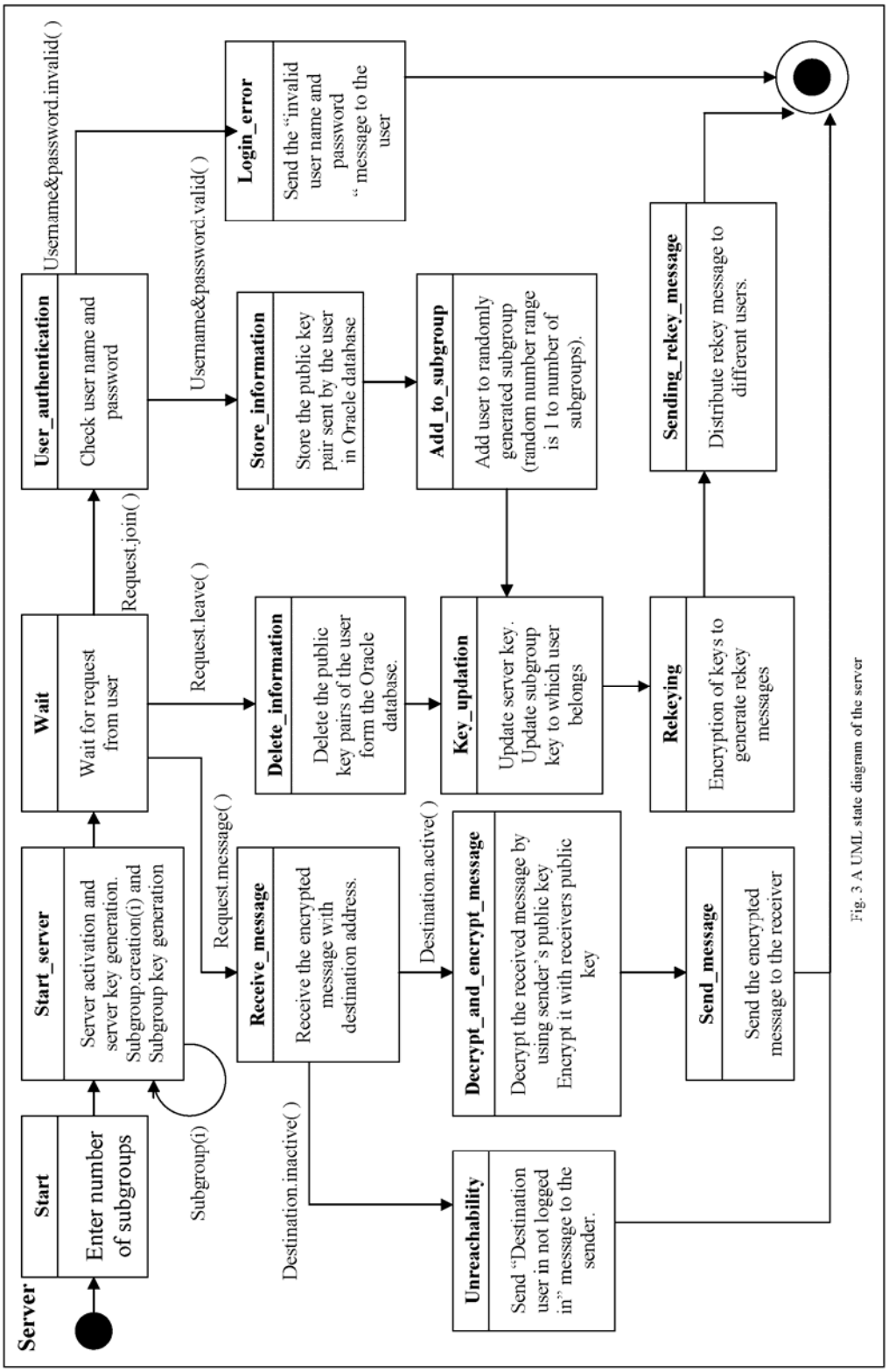

Fig. 3 A UML state diagram of the server

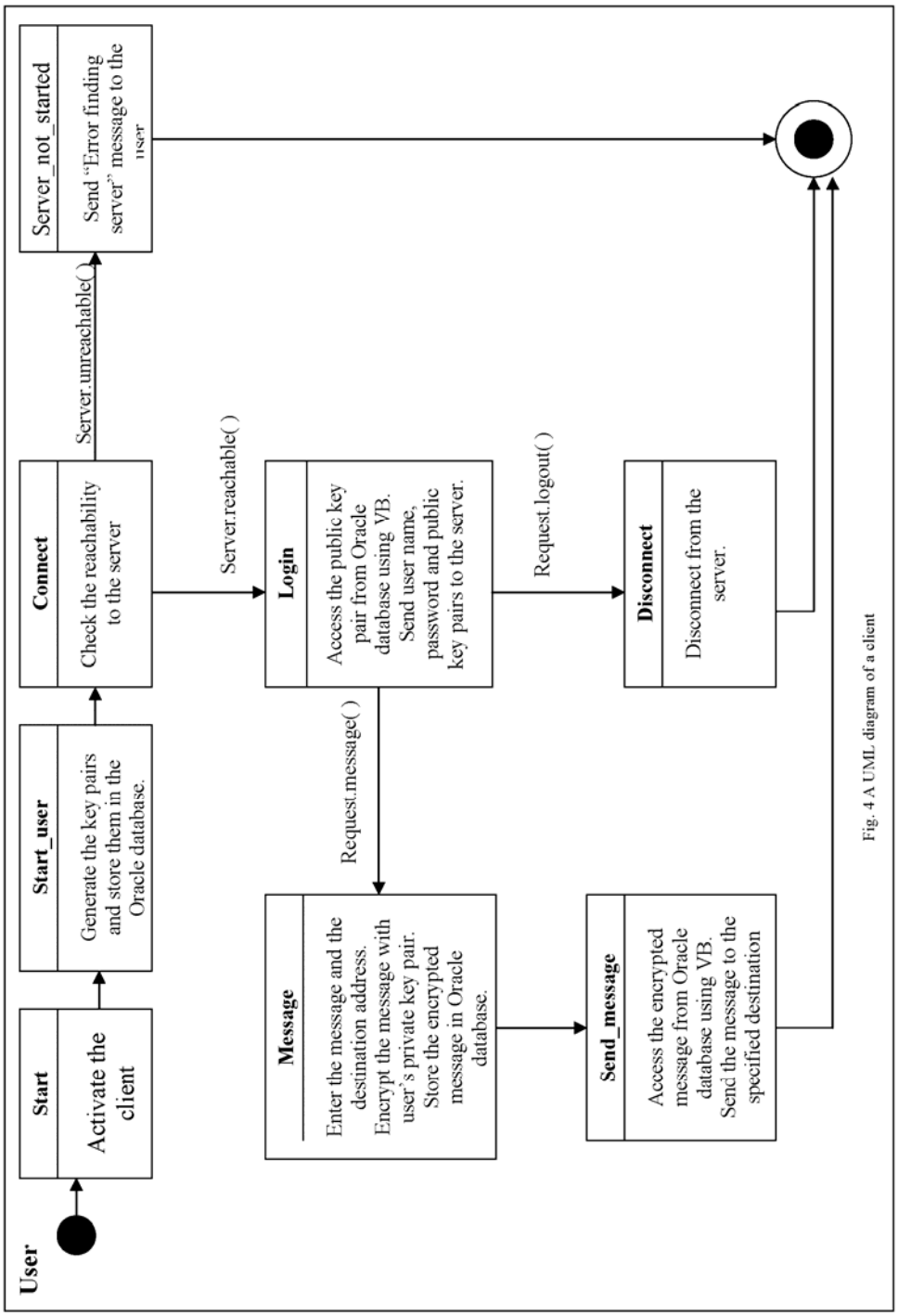

Fig. 4 A UML diagram of a client