Defence Science Journal, Vol. 57, No. 2, March 2007, pp. 279-286

SHORT COMMUNICATION

Implementing Secure Group Communications using Key Graphs

Ch. Aswani Kumar, R. Sri Lakshmi, and M. Preethi
Vellore Institute of Technology, Vellore–632 014

ABSTRACT

While the technical issues of securing unicast communications for client-server computing are fairly well-understood, the technical issues of securing group communications are not. The existing approach to improve the scalability is to decompose a large group of clients into many subgroups and employ a hierarchy of group security agents. In this paper, the secure group communications using key graphs and the implementation of a different hierarchical approach to improve the scalability and secure group communication using key graphs has been presented.

Keywords: Data encryption, group communication, key graphs, scalability of secure group communication

1. INTRODUCTION

Many emerging network applications are based on group communication models. In particular, these require packet delivery from one or more authorised senders to a large number of authorised receivers. In the internet, multicast has been used effectively to provide an efficient, delivery service to large groups. In the near future, providing confidentiality, authenticity, and integrity of messages delivered among group members, will become a critical networking issue[1]. Wong[1], *et al.*, presented the issues in secure group communications using key graphs.

Since every point-to-multipoint communication can be represented as a set of point-to-point communication, the current technology base for securing unicast communications can be extended in a straightforward manner to securing group communications. However such an extension is not scalable to large groups.

For group communications, the server distributes to each member a group key to be shared by all the members of the group[1]. For a group of n members, distributing the group key securely to all members requires n messages encrypted with individual keys[2]. Each such message may be sent separately *via* unicast. Alternatively, the *n* messages may be sent as a combined message to all group members *via* multicast. Either way, there is a communication cost proportional to the group size, *n*. In a point-to-point session, the cost of session establishment and key distribution are incurred just once, at the beginning of the session. But a group session may persist for a relatively longer time with members joining and leaving the session. Consequently, the group key should be changed frequently. To achieve a high level of security, the group key should be changed after every join and leave so that a former group member has no access to the current communications and a new member has no access to the previous communications[1].

Revised 18 November 2005

A trusted server creates a new group key after every join and leave. After a join, the new group key can be sent *via* unicast to the new member encrypted with its individual key and *via* multicast to existing group members encrypted with the previous group key. Thus, changing the group key securely after a join does not involve much work. After a leave, however, the previous group key can no longer be used and the new group key must be encrypted for each or the remaining group members using their individual keys. Changing the group key securely after a leave incurs computation and communication costs proportional to n, the same as initial group key distribution[1]. That is, large groups whose members join and leave frequently pose a scalability problem. In this paper, implementation of the secure group communications using key graphs has been presented.

2. SECURE GROUPS AND KEY GRAPHS

A secure group is a triple (U, K, R), where U is a finite and non-empty set of users; K is a finite and non-empty 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. A key graph is a directed acyclic graph G, with two types of nodes: U-nodes representing users and K-nodes representing keys[1,3]. 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. Figure 1 shows a key graph. Given a key graph G, it specifies a secure group (U, K, R) as follows[3].

- There is a one-to-one correspondence between U and the set of u-nodes in G.
- There is a one-to-one correspondence between K and the set of k-nodes in G.
- (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.

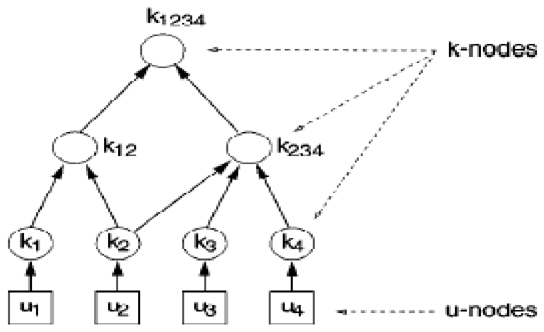


Figure 1. Key graph.

2.1 Re-keying Strategies and Protocols

A user who wants to join or leave a secure group sends a join or leave request to the key server, denoted by s. For a join request from user u, it is assumed that group access control is performed by server s using an access control list provided by the initiator of the secure group[1,3]. A join request initiates an authentication exchange between u and s. If user u is not authorised to join the group, server s sends a join-denied reply to u. If the join request is granted, it is assumed that the session key distributed as a result of the authentication exchange will be used as the individual key k_u of 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 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[1,4].

2.2 Encryption and Decryption Cost

An approximate measure of the computational costs of the server and users is the number of key encryptions and decryptions required by a join/leave request. Let n be the number of users in a secure group. For each join/leave request, the user that requests the join/leave is called the requesting user, and the other users in the group are nonrequesting users. For a key tree, d and h denote the degree and height of the tree, respectively. In this case, for a nonrequesting user u, the average cost of u

Table 1. Average cost per request

	Star	Tree	Complete
Cost of server	$n/2$	$(d+2)(h-1)/2$	2^n
Cost of a user	1	$d/(d-1)$	2^n

for a join or a leave is less than $d/(d-1)$, which is independent of the size of the tree[4,5]. Assuming that a request is equally likely to be a join or a leave, and the group size n is large, the average cost per request is given in Table 1 for the server and a user in the group. An approximate measure of the computational costs of the server and users is the number of key encryptions and decryptions required by a join/leave request[1,2]. From Table 1, it is obvious that complete key graphs should not be used. On the other hand, scalable group key management can be achieved using tree key graphs.

3. IMPLEMENTATION

The existing approach to improve scalability is to decompose a large group of clients into many subgroups and employ a hierarchy of group security agents. In this work, a different hierarchical approach was implanted to improve the scalability[1]. Instead of a hierarchy of group security agents, a hierarchy of key was employed. The process began by formalising the notion of a secure group as a triple (U, K, R), where U denotes a set of users, K denotes a set of keys, and $R \subset U \times K$ denotes a user-key relation, which specifies keys held by each user in U. In particular, each user is given a subset of keys, which includes the user's individual key, and a group key. It was then illustrated how organising the keys in K into a hierarchy and giving the users additional keys can improve scalability of group key management[5,6].

Let there be a trusted server responsible for group access control and key management. In particular, the server securely distributes keys to group members and maintains the user-key relation. To illustrate the approach, the following simple example of a secure group with nine members partitioned into three subgroups has been considered: $\{u_1, u_2, u_3\}$, $\{u_4, u_5, u_6\}$, and $\{u_7, u_8, u_9\}$. Each member is given three keys: Its individual key, a key for the entire group, and a key for its subgroup. Suppose that u_1 leaves the group; the remaining eight members form a new secure group and require a new group key; also, u_2 and u_3 form a new subgroup and require a new subgroup key. To send the new subgroup key securely to u_2 (u_3), the server encrypts it with the individual key of u_2 (u_3). Subsequently, the server can send the new group key securely to members of each subgroup by encrypting it with the subgroup key. Thus, by giving each user three keys instead of two, the server performs five encryptions instead of eight. As a more general example, suppose the number n of users is a power of d, and the keys in K are organised as the nodes of a full and balanced d-array tree. When a user leaves the secure group, to distribute new keys, the server needs to perform approximately $d\log d(n)$ encryptions (rather than n-1 encryptions). For a large group, say, 100 000, the savings can be substantial.

With a hierarchy of keys, there are different ways to construct rekey messages and securely distribute these to the users[7]. Three rekeying strategies: User-oriented, key-oriented, and group-oriented were investigated.

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 these using a key held by the user. In key-oriented 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. In group-oriented approach, the server constructs a single rekey message containing all the 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 join/leave protocols have been designed and specified based upon these rekeying strategies.

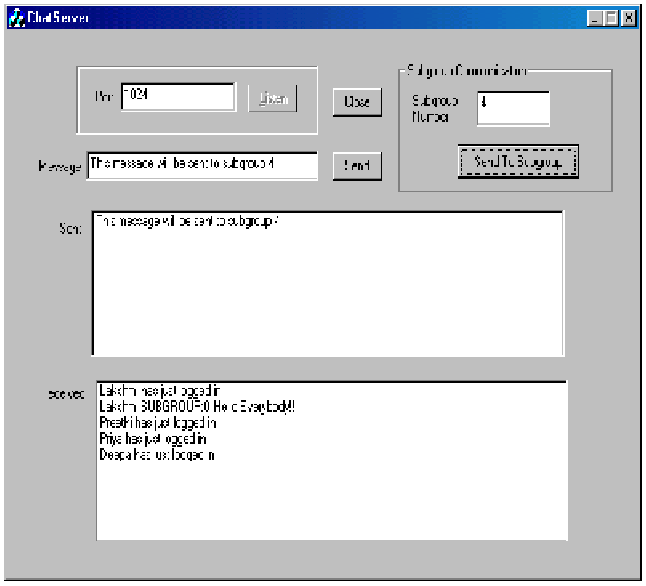
3.1 Key Server

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 individual key, which is shared only with the key server and is used 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. Each user if wants to send messages confidentially to other members of the group can use the group key[8, 9]. Figures 2 and 3 present implementation of a key server and client. The key server performs the following actions.

(a) Creates a new *u*_node for the new user. It also creates a new *k*_node to represent the user's individual key *ku*. It then finds an existing *k*_node in the key tree, generates *k*_nodes from that level to the current level and attaches *k*_node *ku* to the last node in that path as its child. All the keys from the joining point to the root are changed.

(b) Upgrades the key tree by deleting the *u*-node for user *u* and the *k*-node for its individual key from the key tree. If all the users in the subgroups to which the deleted user belongs, have already left the group, then the *k*_nodes denoting those subgroups are also deleted. The parent of the recently deleted *k*_node is the leaving point. All the keys from the leaving point to the root are changed. Figure 4 presents the key server changing the keys after each client joins and leaves. Figure 5 shows the messages sent by the key server to a client.

(c) Places the number of recently deleted *u*_node in a queue so that when a new user joins the group, this location can be allotted. Also allocates nodes (both *u*_node and *k*_nodes) at that position to the new user, when a new user joins the group and if there are any holes in the key tree, the keys from the joining point to the root are changed.



Port: 1024

Subgroup Number: 4

Message: This message will be sent to subgroup 4

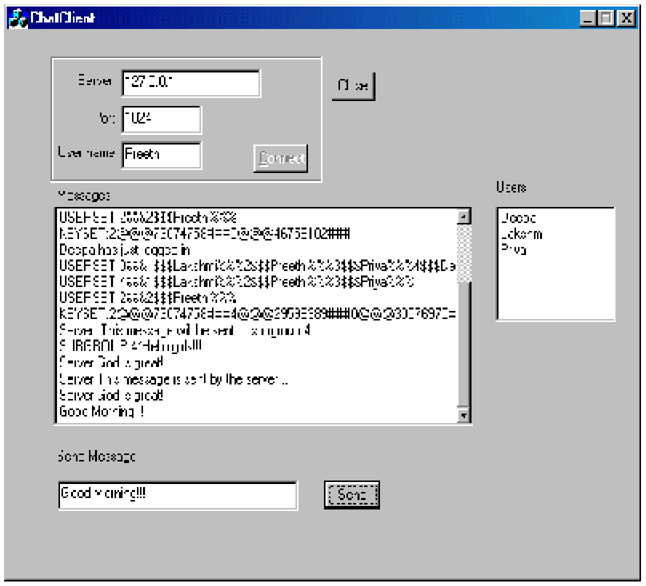
Sent:
```
This message will be sent to subgroup 4
```

Received:
```
Lakshmi has just logged in
Lakshmi SUBGROUP:0 Hello Everybody!!
Preethi has just logged in
Priya has just logged in
Deepa has just logged in
```

Figure 2. Key server.

(d) Constructs user set of all the keys in the key tree, the key set of all the users holding a particular key, a message containing the individual key of a particular user.

(e) Generates two seeds for random number generation using the current time of the machine in which the server application is running. It selects one random number and then manipulates it as a 64-bit number. Also it divides the message into 64-bit blocks.



Server: 127.0.0.1

Port: 1024

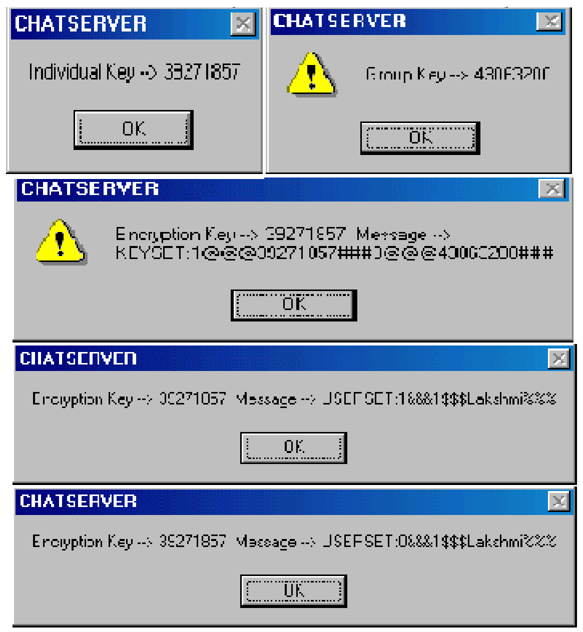
User name: Preeth

Messages:
```
USERSET 2%%&2$$$Preeth%%%
KEYSET:2@@73C74758==3@@@4675E102####
Deepa has just logged in
USERSET 3%%& $$$Lakshmi%%%2$$$Preeth%%%3$$$Priya%%%4$$$De
USERSET 4%%& $$$Lakshmi%%%2$$$Preeth%%%3$$$Priya%%%
USERSET 2%%&2$$$Preeth%%%
KEYSET:2@@73C74758==4@@@2959E89####0@@2300769[illegible]
Server: This message will be sent to subgroup 4
SUBGROUP:4 Hello guys!!!
Server: God is great!
Server: This message is sent by the server...
Server: God is great!
Good Morning!
```

Users:
```
Deepa
Lakshmi
Priya
```

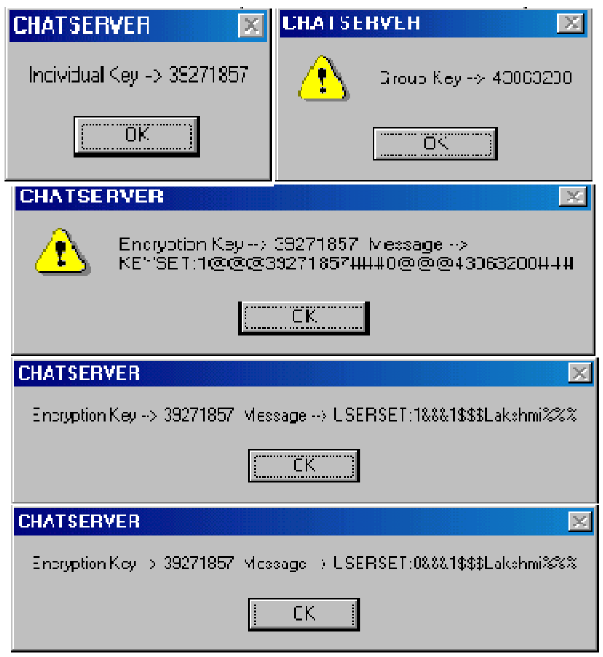
Send Message: Good Morning!!!

Figure 3. Client window.



```
Individual Key --> 39271857
Group Key --> 43063200
Encryption Key --> 39271857  Message -->
KEYSET:1@@@39271857###0@@@43063200###
Encryption Key --> 39271857  Message --> USERSET:1&&&1$$$Lakshmi%%%
Encryption Key --> 39271857  Message --> USERSET:0&&&1$$$Lakshmi%%%
```

Figure 4. Key server changes the keys after each client joins or leaves.



```
Individual Key --> 39271857
Group Key --> 43063200
Encryption Key --> 39271857  Message -->
KEYSET:1@@@39271857###0@@@43063200###
Encryption Key --> 39271857  Message --> USERSET:1&&&1$$$Lakshmi%%%
Encryption Key --> 39271857  Message --> USERSET:0&&&1$$$Lakshmi%%%
```

Figure 5. Messages sent by the key server to a client.

(f) It receives incoming connections. Creates a new socket for new user in the group and accepts the user in the group. Receives all the messages sent by the users. If the user is new, then it interprets the "NEW" message, adds the user information to the end of a linked list, adds the user name to a list of names maintained by the server, and sends the KEYSET and USERSET messages to all the current users in the group. If the message sent by a client is for the entire group, it creates copies of the message and sends to all the users in the group. If the message is only for a particular subgroup, it sends the message only to users of that particular subgroup.

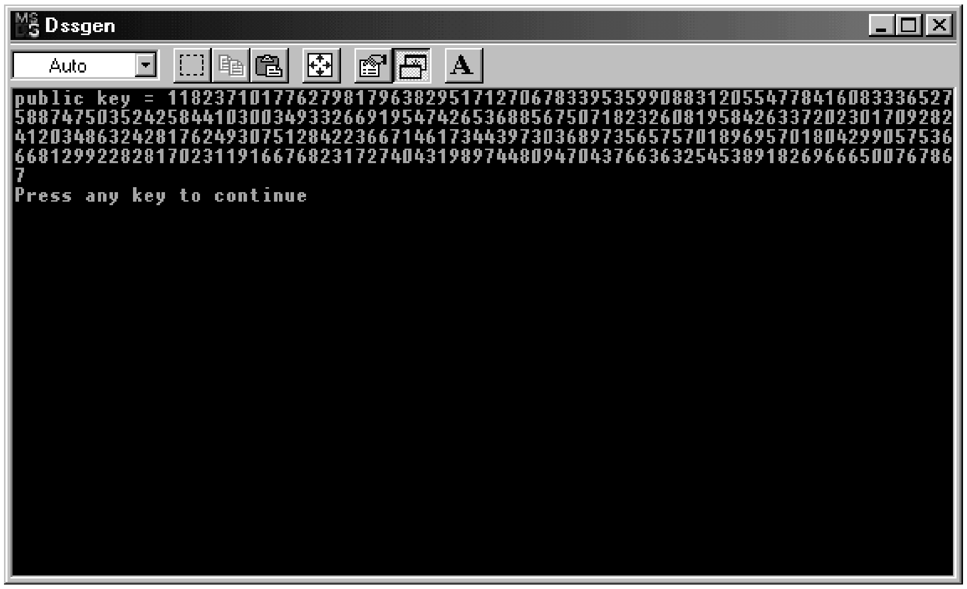
(g) Sends messages from the server to all the clients in the group by encrypting, using the group key. When a user joins or leaves the group, it sends the current list of users to all the members in the group. Sends messages to users of a particular subgroup by encrypting it using the subgroup key.

(h) Deletes the socket held by the deleted user, removes the user's information from the linked list, and also deletes the name from the list maintained by the server. It then sends the KEYSET and USERSET messages to the remaining users in the group.

3.2 Network Communication

The network communication was implemented using socket program. The client receives messages from the server and all other users in the group 10, 11. Also the client sends messages to the server and all other users in the group by encrypting using the group key. If the message is to be sent to a particular subgroup the group number is obtained and it is encrypted using that particular subgroup, key and then it sends the messages to users in that particular subgroup. Sockets will be either in listen mode or connect mode[12].

In our prototype implementation, rekey messages have additional fields, such as, subgroup labels for new keys, server digital signature, message integrity check, etc. Using common values generation, a prime q, another much larger prime $p = 2nq+1$, (where n is random) and a generator g have been generated. Each individual user randomly generates its own private key $x < q$ and makes available a public key $y = gx \bmod p$. Figure 6 presents generation



```
public key = 1182371017762798179638295171270678339535990883120554778416083336527
58874750352425844103003493326691954742653688567507182326081958426337202301709282
41203486324281762493075128422366714617344397303689735657570189695701804299057536
66812992228281702311916676823172740431989744809470437663632545389182696665007678 6
7
Press any key to continue
```

Figure 6. Generation of public and private keys for DSA algorithm.

of public and private keys for DSA algorithm. The security of the system depends on the sizes of *p* and *q* (at least 512-bit and 160-bit respectively). On the client side, the message digest of the received message is computed[13-15]. From this digest, the verification function is computed using the public key. If the user is verified, then the identity of the message is established.

The system has been implemented using Microsoft Visual C++, MIRACL (multiprocessing integer and real arithmetic C/C++ Library) and on the platform Windows 2000. The visual aspect of Visual C++ is used for designing the user interface of a program. MIRACL is a big number library, which implements all of the primitives necessary to design big number cryptography into the real-world application. It is primarily a tool for cryptographic system implementers.

4. CONCLUSIONS

To address the scalability problem of group key management, key trees or graphs were used. On the server side, group-oriented rekeying provides the best performance. On the client side, user-oriented rekeying provides the best performance, with key-oriented rekeying in the second place, and group-oriented rekeying in the third place. To simplify protocol design, key distribution by a central server is used in the work. However, centralised approaches may not be suitable for many applications, where participants wish to generate group keys by themselves so that they can be sure of the freshness and randomness of the group keys.

REFERENCES

1. Wong, C.K.; Gouda, M. & Lam, S.S. Secure group communications using key graphs. *IEEE/ ACM Trans. Networking*, Feb. 2000, **8**, 16-30.

2. Wong C.K. & Lam, S.S. Digital signatures for flows and multicasts. *In* Proceedings of the IEEE ICNP'98, October 1998. Revised version in *IEEE/ACM Trans. Networking*, August 1999, **7**, 502-13.

3. Bird, R.; Gopal, I.; Herzberg, A.; Janson, P.; Kutten, S.; Molva, R. & Yung, M. The kryptoknight family of lightweight protocols for authentication and key distribution. *IEEE/ACM Trans. Networking*, February 1995, **3**, 31-41.

4. Mittra, S. Iolus: A framework for scalable secure multicasting. *In* Proceedings of the ACM SIGCOMM'97, 1997, pp. 277-88.

5. Cormen, T.H.; Leiserson, C.E. & Rivest, R.L. Introduction to algorithms. Cambridge, MA, MIT Press, 1989.

6. Harney H. & Muckenhirn, C. Group key management protocol (GKMP) architecture. RFC 2094, July 1997.

7. Yang, Wen-Her & Shich, Shiuh-Pyng. Secure key agreement for group communications. Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan.

8. Wallner, D.; Harder, E. & Agee, R. Key management for multicast: Issues and architectures. RFC 2627, June 1999.

9. Poovendran, R. Key management for secure multicast communications. University of Maryland, 1999, (PhD Thesis).

10. Ezzell, Ben. Windows 2000 programming with Visual C++. Publications, 2000.

11. Chapman, Davis. Visual C++ in 21 days. Techmedia Publications, 1998.

12. Tanenbaum, Andrew S. Computer networks. Prentice Hall of India Pvt Ltd, 2001.

13. Rhee, Man Young. Cryptography and secure communications. McGraw Hill Publications, 1994.

14. Stallings, William. Cryptography and network security: Principles and Practice. Prentice Hall Inc, 2000.

15. Davis, Carlton R. IPSec securing VPNs. RSA Press, Tata McGraw-Hill Edition, 2001.

Contributors

Mr Aswani Kumar is a Lecturer in the Dept of Information Technology, Vellore Institute of Technology, Deemed University, Vellore. He received Masters in Computer Science from Nagarjuna University and presently pursuing his PhD. He has published 10 research papers in various international and national journals and conferences. His research interests include Information retrieval, and computational intelligence.

Ms R. Sri Lakshmi received BTech (Information Technology) from the Madras University. Presently, she is working as Assistant System Engineer, at the Tata Consultancy Services, Chennai. Her research interests include: Computer networks, network security and cryptography.

Ms M. Preethi received BTech (Information Technology) from the Madras University. Presently, she is doing MS (Computer Science) from the California State University, Los Angles, USA. Her research interests include: Security, and object-oriented aided designing.