An Efficient Group Rekeying Method Using Enhanced One Way Function Tree Protocol

B.Parvatha Varthini [†] and S.Valli [††],

[†]*Department of Computer Applications, St.Joseph's College of Engineering,Chennai – 119, India.*
[††]*Department of Computer Science, Anna UniversityChennai – 25, India.*

Summary

There are several group key management solutions to implement group rekeying in secure multicast communication. These solutions suffer from 1 affects all phenomena if they use centralized architecture or from data translation latency if they use decentalised architecture. Further, these group rekeying protocols add communication as well as computational overhead in every group member and in the group key controller (GKC) of the group. This proposed work develops an efficient group rekeying solution that follows an adaptive group rekeying architecture. This adaptive group rekeying architecture provides a good tradeoff between the one affects all phenomena and the data translation latency. This method uses an enhanced one way function tree (EOFT) algorithm based on the Chinese remainder theorem for group rekeying. EOFT has minimum computational overhead and communicational overhead particularly during the join of the members in the group.

Key words:

data translation latency, enhanced one way function tree, group key controller, rekeying the group key
.

Introduction

Multicast effectively delivers the messages from one source to many destinations instead of sending n unicast messages to n destinations. Multicast reduces the load and network resource consumption. Due to the explosive growth of the Internet, multicast applications need a lot of attention in recent years. Many Internet applications like software update, pay channel TV, video conferencing and multiplayer games use multicast. Multicast security is more complex than unicast security. The content of the multicast traffic is to be sent confidentially among the group members. Another important multicast security requirement is group authentication. Encrypting the messages using a cryptographic secret key (the group key) provides security in multicast communication. The GKC generates and distributes the group key to the group members.

The dynamic nature of the group introduces a great challenge for multicast communication. The group key is to be changed (rekeying) when a new member joins the group or an existing member leaves the group. If the group is highly dynamic, the rekeying process is to be triggered often and it increases the complexity in computation and communication. The group rekeying procedure should ensure forward secrecy (the group members leaving the group cannot derive the future group keys) and backward secrecy (a new member joining the group cannot discover the previous keys). The existing group rekeying schemes maintain forward secrecy and backward secrecy. But they fail to meet both the scalability issue of the centralized group rekeying architecture and the data translation latency of the decentalised group rekeying architecture.

This proposed work provides an adaptive group rekeying solution that provides a good trade off between the scalability issue and the data translation latency. This work uses EOFT algorithm for group rekeying because EOFT has a minimum number of rekeying update messages compared to the other existing group rekeying protocols.

2. Existing Work

There are many group key management solutions to implement the group rekeying in secure multicast communication.

2.1 Centralized Architecture

A single GKC manages all the group members in the architecture developed by Wallnner [1]. The GKC generates and encrypts the group key separately for every

Manuscript received December 5, 2006.
Manuscript revised December 25, 2006.

member of the group. When a member joins/leaves the group, the GKC rekeys the group key. The GKC encrypts and sends this new key to every member separately. This architecture is inefficient when the group size is too large. It suffers from one affects n phenomena.

Hierarchical Key Graph Algorithm is another rekeying algorithm that does the group rekeying effectively. Here the group is viewed as a logical tree structure to reduce the overhead incurred at the group members during the join/leave operation. The GKC creates a rooted balanced tree that has as many leaf nodes as there are members. Each leaf node of the key tree is associated with a member of the group. Each internal node represents a logical subgroup. The root node represents the group key. The existing key graph algorithms are Logical Key Hierarchy Graph (LKH), one way Function tree (OFT).

Wong [11] suggests LKH algorithm for rekeying. It is an efficient rekeying method for large groups. It uses a hierarchical key tree. The GKC shares a separate unique secret key with each one of the members through unicast. GKC creates all internal node keys but sends only a subset of them to each member. Each member receives the keys of all the nodes in the path from that member to the root. The number of keys that each member holds is equal to log n, where n is the size of the group. When a member joins the group, the GKC needs to change every internal node key in the new member's path to the root. When a member leaves the group, the GKC changes only the keys that the leaving member knows.

Waldovogel [9] proposed another key management scheme OFT. Wei-chi Ku [10] suggests an improved version of OFT for key establishments in large dynamic groups. OFT uses a logical key tree for rekeying with better leave rekeying compared to LKH. The GKC shares a separate unique secret key with each one of the members through unicast. The GKC uses a one-way function 'g' to compute a blinded key corresponding to each key in the key tree. Each member receives the blinded keys of the sibling of the internal nodes in the path from that member to the root of the tree. Each internal node computes its key by applying a mixing function 'f' to the blinded keys of its children. When a member joins, the GKC sends $\log_2 n$ blinded keys to the new member. When a member leaves, the GKC needs to send as many blinded keys as the length of the path from the rekeyed node to the root.

2.2 Decentralised Architectures

IKAM, Iolus and Hydra are some of the decentalised group communication architectures. IKAM architecture according to Hardjono [3] divides the group into one domain and many areas. The domain consists of a number of administratively manageable areas. A group member resides within any one of these areas. Two important IKAM entities are the domain key distributor (DKD) and the area key distributor (AKD). DKD is responsible for the key management at the domain level. AKD is responsible for the key management at the area level. Each host in IKAM has three keys namely Traffic Encryption Key, Area Group Key and Member Private Key shared with its AKD. The maintenance of these different keys adds new complexity.

Iolus proposed by Mittra [5] splits a large group into smaller subgroups. The top level group is managed by a group security controller (GSC) who is responsible for the security of the entire group. Each subgroup is managed independently by a GSA (Group Security Agent).The membership change in any subgroup is located locally. The corresponding subgroup controller GSA does rekeying only in that subgroup when a member leaves or joins a subgroup. It does not affect the rest of the group. Hence it solves the 1 affects n issue. But it introduces data translation latency since every subgroup agent encrypts the incoming data stream with the key of the outgoing data stream whenever the data is sent between the subgroups.

Hydra architecture of Rafaeli [7] consists of two hierarchical levels. Group members are in the bottom level and separated into subgroups. The top level is composed of Hydra servers who are subgroup managers. The multicast group known as the Hydra group agrees on a common group key. This hydra group is virtually divided into hydra server groups to distribute the agreed key to their respective subgroup members in the bottom level. However, hydra needs a long time to relay the whole group for every membership change.

The decentralised group rekeying architecture solves this scalability issue by organizing the group as different subgroups with their own Traffic Encryption Key. It reduces the number of rekey update messages when a member joins or leaves. But it includes the data translation latency due to additional encryption and decryption of the messages when they are sent between subgroups.

2.3 Adaptive Architecture

Yacine Challal[12] introduces an adaptive solution SKAM which structures group members into clusters

according to the application requirements. Each cluster is composed of a set of subgroups that share the same TEK. Each subgroup is managed by a SKAM agent. In clusters, the root agent is active and all internal agents are passive. Messages are decrypted and reencrypted only at the cluster's root. The reformation of clusters is activated periodically. The time interval is selected arbitrarily without considering any standard parameters.

This proposed adaptive hybrid architecture calculates the rekeying cost during each join/leave operation but not periodically. Hence it removes the time interval selection anomaly found in SKAM. SKAM uses 1 root- N leaves rekeying protocol which is inefficient in large groups. This work uses EOFT [6], an enhanced one way function tree protocol that developed by the authors. EOFT has good performance over other group rekeying protocols.

3. Enhanced OFT

3.1 Overview of EOFT

This proposed work uses EOFT algorithm for group rekeying. EOFT [6] enhances the efficiency of OFT using the Chinese reminder theorem. It follows a hierarchical key graph for rekeying. This algorithm uses CRT tuple of smaller remainders instead of keys of larger bits. Any computation on keys is carried out in parallel on their corresponding CRT remainders. This algorithm reduces the computational overhead of a leave/join event compared to OFT. The GKC shares a pseudo function P, a random number M, an Arithmetic function A along with a separate unique secret key with each member through unicast. A pseudo function P can be a hash/mac function or simple transformation of bits depending on the security level required. The GKC generates a random number $M=[m_i]$ for finding mod m_i reminder tuples corresponding to each key. Each member receives the CRT tuples of pseudo keys of the sibling of the internal nodes in the path from that member to the root of the tree. Each internal node's key is computed by applying an arithmetic function R on its children's pseudo key tuples. The arithmetic function R can be +/-/*/ a repeated combination of odd number of these three operations. But existing works use only hash for P and XOR for R.

3.2 EOFT Member-Join Rekeying

When a member joins the group, the GKC creates a new node in the tree, so that the tree is balanced. Splitting the nearest leaf node from the root does the balancing of the tree. After accommodating the new member in the tree, the GKC shares a unique secret key, new R and M with the new member through unicast. The random number M is derived from the previous number by omitting any one of its factors. The new arithmetic operation R is a new combination of odd number of operators +, *, and -. The GKC just informs the new R and M through a multicast message encrypted with the old group key to the existing group members. Unlike in OFT and LKH, the GKC does not send any new internal node keys to the existing members in the join event. Hence EOFT reduces the join event communication overhead. The GKC multicasts a single rekeying message to the members during the join event.

3.3 EOFT Member-Leave Rekeying

When a member leaves the group, the GKC reconstructs the tree as a balanced tree. If the departing member's sibling is a leaf node, it gets associated with its parent node or it assumes the parent's position on the tree. The GKC needs to rekey to maintain forward secrecy. Unlike in LKH, the GKC does not change all the pseudo keys that the departing member knows. Since the GKC does not use pseudo keys to encrypt any message, it does not change all the pseudo keys supplied to the departing member. For the sibling leaf node of the departing member, the GKC sends a new key encrypted with its old key. The GKC rekeys all the keys from rekeyed node's position to the root. The GKC sends $\log_2 n + 1$ messages during leave where n is the size of the group.

The group key evaluation process of EOFT is illustrated in Fig 1. The algorithm required for each member to compute the group key is given in the following algorithm.

3.4 Algorithm for Computing the Group Key using EOFT

The steps involved in computing the group key , by a group member A are given below.

Step 1: The group member A receives its secret key K_a through unicast registration from the GKC.

Step2: The GKC sends Pseudo function P, Arithmetic function R and a random number M to all the members A, B, C, D, E, F, G, and H as in Fig 1. Also it sends the pseudo key remainder tuples of the siblings of the internal nodes along the path from the member to the root. They are $[K_b']$, $[K_{cd}']$, $[K_{eh}']$ for the member A.

Step 3: The group member A evaluates its pseudo key $[K_a']$ as $P(K_a)$

Step 4: The group member A evaluates $[K_{ab}]$ as $R([K_a'], [K_b'])$ and $[K_{ab}']$ as PK_{ab})

Step 5: The group member A evaluates $[K_{ad}]$ as $R([K_{ab}'], [K_{cd}'])$ and $[K_{ad}']$ as $P(K_{ad})$

Step 6: The group member A evaluates $[K_{ah}]$ as $R([K_{ad}'], [K_{eh}'])$ which is the group key

Thus all the group members compute the group key using the above algorithm.

3.5 Performance of EOFT

We compare the performance of the EOFT with the other group key management schemes quantitatively. The measure of communication complexity depends on the number and size of the messages transmitted for key updates. EOFT reduces join rekeying communication complexity to simply one multicast message compared to all other group rekeying algorithms. The leave rekeying communication complexity is EOFT in just half that of LKH.

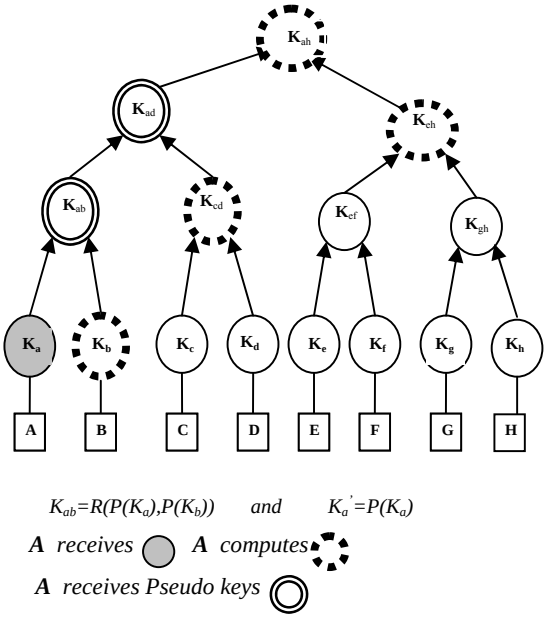

Fig. 1 EOFT Group Key Computation

4.The Proposed Adaptive Hybrid architecture

In this proposed architecture, when a group is initiated, it behaves as a single group controlled by a single GKC. When a member wishes to join the group initially, it sends the request to the single GKC that is available. The GKC decides whether to continue as one subgroup or divide into two subgroups so that the rekeying overhead is less. Suppose the splitting cost is less, the GKC splits the group into two subgroups as parent and child. The current GKC manages the parent group. The half of the members who joined recently leaves the current group and joins the new subgroup. This proposed architecture activates one of the members of new subgroup as the GKC of the childgroup. Suppose the splitting cost is more, then the group continues to be in the same architecture and does rekeying. When a member leaves the group initially, rekeying is achieved with the existing single GKC architecture.

In the case of more than one subgroup, a member who wishes to join the group broadcasts a join request. The nearest subgroup's GKC will reply at the earliest and proceed with the joining operation. When a member joins the subgroup S_j, the subgroup's GKC calculates the rekeying cost in the group Sj as well as the rekeying cost to be incurred due to splitting. In the case of less splitting cost, Sj is divided into two subgroups S_j and

S_k such that S_j is the parent of S_k. The half of the members who joined S_j recently leaves the current group and joins the new subgroup S_k. This proposed architecture activates one of the members of new subgroup S_k as the GKC of the subgroup S_k.

When a member leaves the group, the corresponding subgroup's GKC calculates the cost induced by rekeying in that subgroup and data translation latency for communication with its parent. It also calculates the cost of rekeying in the group formed by merging with its immediate parent. When a member leaves from the subgroup S_k, its GKC calculates the rekeying cost within that subgroup S_k, and compares with the cost of merging S_k with its parent S_j. If the merging cost is less, the subgroup S_k is merged with the subgroup S_j and then a mutually accepted group key is computed.

4.1 Stochastic Model Of Multicast Sessions

This work creates sample multicast sessions to calculate the rekeying cost analytically during join/leave operation. Almeroth [1], [2] proved that the multicast sessions follow the Markov Model. The equation (1) gives the steady state probability of a multicast communication Markov process model.

$$P_k = ((\lambda / \mu)^k / K!)e^{(-\lambda/\mu)}, k = 0,1,2,..., \quad (1)$$

where λ is the arrival rate of members and service mean $1/\mu$ in that multicast group. The group membership time rate follows an exponential distribution. The arrival rate of the group members follows a Poisson process. λ/μ is the average number of group members at a time in a subgroup. ERM(λ) denotes the expected number of rekeying messages in a group S with λ members. ERM_J_k and ERM_L_k denotes the average number of rekeying messages during join and leave procedure respectively. The expected number of rekeying messages in a group with λ members is given by the function in (2).

$$ERM(\lambda) = \lambda \sum_{o}^{\infty} P_k(ERM\_J_k(\lambda) + ERM\_L_k(\lambda)) \quad (2)$$

Kin-Ching Chan [4] approximates the steady state probability of a multicast communication Markov process model to an unit function as in (3).

$$\delta(K - (\lambda / \mu)) = \begin{cases} 1 & if \quad k = \lambda / \mu \\ 0 & otherwise \end{cases} \quad (3)$$

We calculate the average number of rekeying messages using the unit function given by (3) as

$$ERM(\lambda) = \lambda \sum_{o}^{\infty} \delta(K - (\lambda / \mu))(ERM\_J_k(\lambda) + ERM\_L_k(\lambda))$$
$$ERM(\lambda) = \lambda(ERM\_J_{\lambda/\mu}(\lambda) + ERM\_L_{\lambda/\mu}(\lambda)). \quad (4)$$

4.2 Computation Of Rekeying Cost

Every subgroup of this proposed work follows EOFT group rekeying scheme. The number of join and leave rekeying messages in EOFT are

$$ERM\_J_{\lambda/\mu}(\lambda) = 1$$
$$ERM\_L_{\lambda/\mu}(\lambda) = \log_2 \lambda + 1.$$

Therefore the expected number of rekeying messages in a group following EOFT given by (4) reduces to (5).

$$ERM(\lambda) = \lambda(2 + \log_2 \lambda) \quad (5)$$

The Rekeying cost in a subgroup is measured in terms of the number of rekeying messages generated during a join and leave. Equation (6) gives the rekeying cost in the subgroup S_i with members λ_i as

$$C_i^R = ERM(\lambda_i) * t_g \quad (6)$$

where t_g is the average time taken to generate and send a rekey message. Equation (6) reduces to (7) by substituting ERM(λ_i), the expected number of rekeying messages in a group S_i with λ_i members from (5).

$$C_i^R = \lambda_i(2 + \log_2 \lambda_i) * t_g \quad (7)$$

If a subgroup divides into two subgroups, the rekeying cost includes the rekeying cost in the new born child subgroup and the cost induced due to data translation between the parent and the child. Suppose the subgroup S_i splits into S_i with λ_i members and S_j with λ_j members, then split rekeying cost is

$$C_{ij}^S = C_j^R + C^{TL} \quad (8)$$

where C^{TL} is the cost induced by the translation latency.

In data translation, the incoming message is decrypted and then encrypted with outgoing stream key. So it executes a symmetric key cryptographic algorithm like DES /IDEA twice. t_E denotes the time taken to encrypt a message of payload size 1KB using DES / IDEA. Then the translation cost C^{TL} is

$$C^{TL} = 2 * L * t_E \quad (9)$$

where L is the actual size of the payload in KB.

Hence substituting C_j^R from (7) and C^{TL} from (9) , the split rekeying cost is

$$C_{ij}^S = \lambda_j(2 + \log_2 \lambda_i) * t_g + 2 * L * t_g. \quad (10)$$

When a subgroup merges with its immediate parent subgroup, the rekeying cost includes the mutual key generation cost in the merged group. When the subgroup S_i with λ_i members merges with the subgroup S_j with λ_j members, the merge rekeying cost is

$$C_{ij}^{m} = C_{i+j}^{R}. \quad (11)$$

The resulting merged subgroup has members $\lambda_i + \lambda_j$. Hence, the merge rekeying cost is

$$C_{ij}^{m} = (\lambda_i + \lambda_j)(2 + \log_2(\lambda_i + \lambda_j)) * t_g. \quad (12)$$

5. Simulation Results

The simulation of this work runs on Ns-2. This simulation has at the most 65 dynamic members at an instance. This simulation runs the application MFTP (Multicast File Transfer Protocol) that needs low synchronization between the source and destinations. We generate the multicast sessions according to the stochastic model described in the chapter V.

The inter arrival time of the group members is 24 sec. The membership duration is 25 minutes. This simulation runs for one hour with joining phase, joining/leaving phase and leaving phase. The encryption algorithm used in this work is DES. We calculate the encryption time delay for every 1KB payload as 0.032ms using OPENSSL in UNIX environment. The simulation runs for payload of 32KB as well as for 16KB. The time taken for data translation after splitting into two subgroups is $C^{TL} = 2 * L * 0.032$ where L is the size of the payload. The average time t_g taken to generate and send a rekey message is 0.0412ms with respect to the generated topology.

This simulation starts with a single GKC. As a member joins the group, the GKC calculates the splitting cost using (10). The pay load size is 32 KB. The performance of this adaptive architecture using EOFT is compared with Iolus, a decentalised architecture and the Centralized architecture. It splits the group to reduce the attenuation of 1 affects all phenomena if the splitting cost is less. Until t = 384 seconds, the centralized architecture continues. Then, the GKC splits the group into two subgroups. The number of subgroups increases to three at t = 744 sec. At the end of the joining phase, the number of subgroups increases to seven. At t =1500 sec, the members start leaving the group. The GKC calculates the merging cost using (12) and tries to merge the subgroups to reduce the data translation latency. At t = 1872 sec a subgroup merges with its parent. The simulation draws out the graphs given in Fig. 2 and Fig. 3 at the end t = 3600 sec. The simulation runs for 16 KB payload also. This simulation runs for one hour with joining phase, joining/leaving phase and leaving phase for the payload of 16 KB.

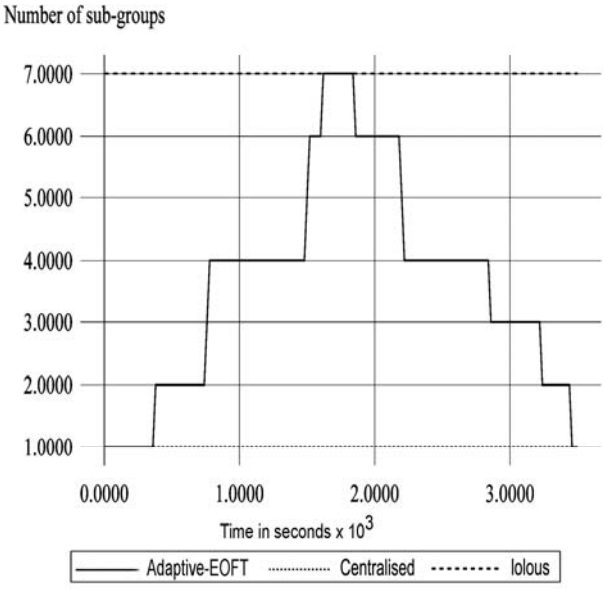

Fig. 2 Comparison of Data Translation Latency (Payload = 32 KB)

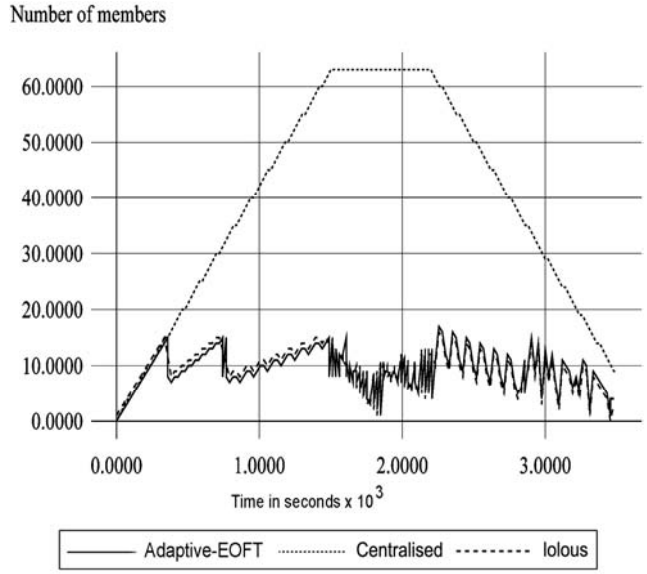

Fig. 3 One Affects All Phenomena (Payload = 32 KB)

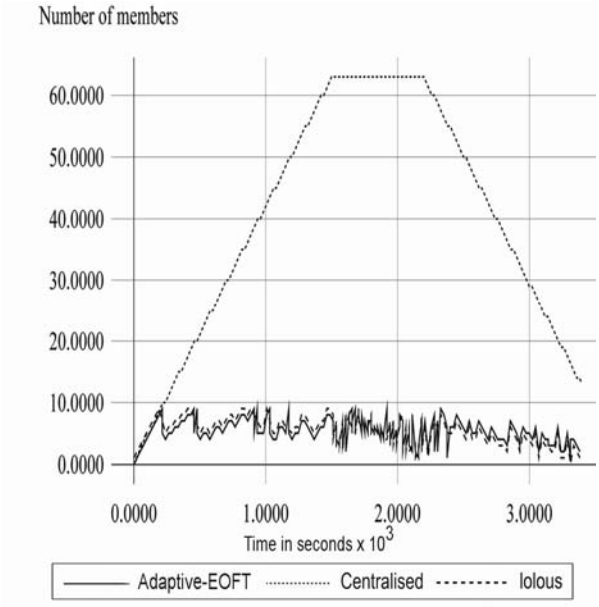

Fig. 4 One Affects All Phenomena
(Payload = 16 KB)

The performance of this adaptive architecture using EOFT is compared with Iolus (a decentalised architecture) and the Centralized architecture. Fig 2 discusses the latency caused by the data translation due to the formation of subgroups in the three architectures with the payload size of 32 KB. The number of subgroups gives the number of times the reencryption is done while the data is sent between the subgroups. The data translation latency coincides with Iolus when the load is heavy but is lesser than Iolus in all other cases. Fig. 3 compares the attenuation of the one affects n phenomena of the above said three architectures with the payload size of 32 KB. It is measured by the number of members that gets affected during the rekey procedure. It proves that this adaptive architecture possesses the efficiency that is very close to Iolus with respect to the one affects n phenomena. Fig. 4 shows that the 1affects n attenuation is lesser in the proposed work even if the payload size is 16 KB, compared to Iolus and centralized architecture.

Conclusion

This work uses an adaptive group rekey architecture that does group rekeying efficiently using EOFT protocol. It starts with a centralized architecture. As the members join the group, it splits the group into more subgroups to solve the scalability issue. This architecture controls the number of subgroups by calculating the rekeying during the join or leave of a group member. According to the group dynamism, it allows the subgroups to merge or a subgroup to divide into two sub groups by comparing the cost of one affects n phenomena due to merging and cost of data translation latency due to splitting. It maintains a good balance between one affects n and data translation latency compared to the existing rekeying algorithms.

REFERENCES

[1] K. Almeroth and M. Ammar, "Collecting and Modelling the join/leave behaviour of multicast group members in the Mbone", in *proceedings of Symposium on High Performance Distributed Computing*, Aug 1996, pp. 209-220.

[2] K. Almeroth and M. Ammar, "Multicast group behaviour in the internet's multicast backbone (Mbone)", *IEEE Communications Magazine*, volume 35, number 6, June 1997, pp. 124-129.

[3] T. Hardjono, B. Cain and I. Monga, "Intra Domain Group Key Management Protocol", *draft-ielf-ipsec-infragkm-02.txt, IETF*, Feb 2000.

[4] Kin-Ching Chan and S. H. Gary Chan, "Distributed Servers Approach for Large-Scale Secure Multicast", *IEEE Journal On Selected Areas in Communications*, volume 20, number 8, October 2002, pp. 1500- 1510.

[5] S. Mittra, "Iolus: A framework for scalable secure multicasting", in *Proceedings of the ACM SIGCOMM*, Vol. 27, 4, Sept. 1997, pp. 277-288.

[6] B. Parvatha Varthini and S. Valli, "EOFT: An Enhanced one way Function Tree rekey Protocol based on Chinese Remainder Theorem.", in *Proceedings ISCIS05 Istanbul, Turkey, Lecture Notes in Computer Science*, Vol. 3733, October 2005, pp. 33-44.

[7] Rafaeli and D. Hutchison, "Hydra: A distributed group key management", in *proceedings of the 11th IEEE International WETICE Enterprise Security Workshop*, June 2002, pp. 62-67.

[8] D. Wallnner, E. Harder, and R. Agee, "Key Management for Multicast: Issues and Architecture", *RFC 2627*, June 1999.

[9] M. Waldvogel, G. Caronni, D. Sun, N. Weiler and B. Plattner, "The VersaKey framework: Versatile group key management", *IEEE Journal. Selected. Areas in Communications Special Issue on Middleware* Aug. 1999, pp. 1614–1631.

[10] Wei-Chi Ku and Shuai-Min Chen, "An Improved Key Management Scheme for Large Dynamic Groups Using One-Way Function Trees", *Proceedings of ICPPW'03*, oct.2003, pp. 391-396.

[11] C.K. Wong, M. G. Gouda, and S. Lam, "Secure group communications using key graphs", *IEEE/ACM Transactions on Networks.*, Feb. 2000, pp. 16–30.

[12] Yacine Challal, Hatem Bettahar and Abdelmadjid Bouabdallah, "SAKM: A Scalable and Adaptive Key Management Approach for Multicast Communications", *ACM SIGCOMM Computer Communications Review*, Volume 34, Number 2: April 2004, pp. 55-70.

Parvathavarthini B. received M.Sc and M.Phil degree in 1988 and 1989 respectively. She received M.B.A and M.E degree in 1998 and 1999 respectively. She is working as Professor in the department of Master of Computer Applications, St. Joseph's College of Engineering. Her research includes Computer Networks, Security, multimedia applications and Graphics.

Valli S. received the B.E, M.E and Dr.Eng degrees in 1990, 1991 and 2001 respectively. She is working as Assistant Professor in the Department of Computer Science, Anna University. Her research interest includes software engineering, networks, parallel computing and object oriented systems.