

An Optimized approach for Multicast Rekeying

¹S.Sasikala Devi, ²Dr. Antony Selvadoss Thanamani

¹Dept. of CA, SS & IT CMS College of Science and Commerce, Coimbatore, Tamilnadu, India

²Dept. of Computer Science, NGM College of Arts and Science, Pollachi, India

Abstract

For any multicast group communication, group key agreement was found to be a vital challenge because of its dynamic nature. Although many solutions have been proposed to handle group key changes, this paper gives different aspect for rekeying. A new multicast key distribution scheme was proposed to reduce computation complexity by employing MDS codes, a class of error control codes, on a key tree to distribute multicast key dynamically instead of conventional encryption algorithms. But that scheme is managing key distribution in a centralized manner. In this paper we propose a distributed key distribution scheme, using a logical group key structure, PFMH tree, and the concept of virtual user position. The proposed scheme applied upon any single user join or leave and multi user joins or leave event. This approach ensures the forward secrecy and backward secrecy, reduces the rekeying complexity, communication, computation and storage complexity and time cost.

Keywords

Multicast, Mds Code, Logical Group Key Structure, Rekeying, Erasure Decoding

I. Introduction

Multicast is an effective method for distributing information to multiple users in a group communication; it reduces the consumption of network resources. Multicast is supported on the internet, or via satellite communication, wireless network, sensors etc., in multicast group communication, all the authorized members share a session key, which will be changed dynamically to ensure forward and backward secrecy referred as “group rekeying”.

The forward secrecy ensures that the members who left the group cannot get access to future group data, and the backward secrecy ensures that currently joined members cannot access past group data. For a multicast group with a large number of members, key-tree based schemes were introduced to decompose a large group into multiple subgroups with smaller sizes [2-6]. Using these schemes, communication complexity is reduced at the cost of increase in storage and computation complexity, very few efforts have been made to reduce computation complexity, Communication complexity, storage complexity, Time cost and Scalability.

In any key-tree based key distribution schemes, Group controller distributes secret data and keys to the group members, where each member shares a different individual key with the GC using conventional encryptions followed by unicasts. A new scheme called efficient computation multicast key distribution [1], realizes this operation using one erasure decoding of certain MDS code, followed by one multicast to all the members and are centralized in nature which uses MDS code-Based Rekeying on the key tree. This key tree based rekeying does not change communication complexity and storage complexity [7]. According to Wei Yu, Yan sun and K.J. Ray Liu [8], A PFMH tree, as well as cost-minimizing PFMH tree-based contributory group key agreement (PACK) protocol is used to achieve lower rekeying cost in a dynamic group membership events. In our paper, MDS code-based Rekeying on a key tree is been integrated into logical key tree structure,

called PFMH trees, to reduce rekeying cost, storage complexity, computation complexity and communication complexity.

In this paper, session keys information is encoded using error control codes rather than encryptions. We propose an efficient logical key tree structure, called PFMH tree with MDS code to optimize the rekeying.

II. MDS Code Based Key Distribution On Pfmh Key Tree

MDS (Maximum Distance Separable) codes are a class of block error control codes that meet the Singleton Bound, i.e., $d = n - k + 1$ for an (n, k, d) code over $GF(q')$. A k -symbol message block $m = m_1 \dots m_k$ is expanded to an n -symbol codeword block $c = c_1 \dots c_n$ [1, 10]. Using a proper erasure decoding algorithm, the message block m can be perfectly recovered from any k symbols of the codeword c . We choose the Reed Solomon codes (RS) [9], as the MDS codes, since it is the most widely used MDS code. A Vander monde Matrix is used as a generator matrix for the chosen RS code [10], for rekeying purpose, an RS decoding operation is equivalent to solving a group of linear equations. The rekeying process is done using the inverse Vander monde matrix in terms of decoding operations. Though researches were done using MDS code instead of conventional cryptography, these schemes was based on key tree which follows structure preserving protocol. In this, for a group of n members, communication complexity is $O(\log n)$ and storage complexity is $O(n)$ for the GC and $O(\log n)$ for group member.

In our basic scheme instead of a group controller the nodes itself constructs a nonsystematic (X, n) MDS code C over $GF(q)$. Thus the concept of group controller is exempted.

A. Structure of PFMH Key Tree

PFMH tree is an efficient logical key tree structure for contributory group key agreement schemes [8]. PFMH tree is a combination of two special key tree structures: Partially Full (PF) key tree and Maximum Height (MH) key tree. In this paper, the total number of leaf nodes indicates the size of a key tree. The function $\log()$ and $\log_2()$ will be used exchangeable, and if it is a “full (key) tree,” it mean a fully balanced binary (key) tree with size $2k$, where k is a nonnegative integer. Fig.1, shows the example of PH, MH and PFMH key tree.

1. PF Key Tree

Let T be a binary key tree of size n , and $n' = 2^{\lceil \log n \rceil}$. The left subtree of T is a full key tree with size n' , and the right subtree of T is a PF key tree with size $(n - n')$.

2. MH Key Tree

The right subtree of T is a leaf node, and the left sub tree of T is an MH key tree with size $n - 1$.

3. PFMH Key Tree

The left subtree of T is a PF tree, and the right subtree of T is an MH tree.

The height of a PF key tree with size n is $\lceil \log n \rceil$, the height of an MH tree with size n is $n - 1$. In PFMH tree T , main tree refer to the PF subtree, T_{main} and join tree refer to the MH subtree of T ,

and T_{join} . There are two basic procedures to manage and update PFMH tree they are unite and split.

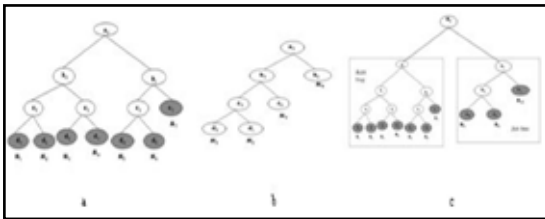


Fig. 1: Examples of (a) PF (b) MH (c) PFMH Key Tree

(i). Unite Procedure

Let $T = \{T_1 \dots T_L\}$ be a set of full key trees. Each key tree $T_i \in T$ represents a subgroup, and each leaf node of T_i is a member of this subgroup. If a group member belongs to T_i and $T_i \in T$, then this group member belongs to T . The procedure unite (T) is to combine those key trees in T into a single PF key tree.

(ii). Split Procedure

Given a key tree T , the procedure split (T) is to partition T into a set of full key trees with the minimum set size. This Procedure presents a way to locally and virtually split a key tree, where “locally” means that no intercommunication is needed among group members and each member only needs to update the key tree structure maintained by itself locally, whereas “virtually” means that no two-group is needed to perform “split.” fig. 2, shows example of key tree update after applying a) unite and b) split procedure.

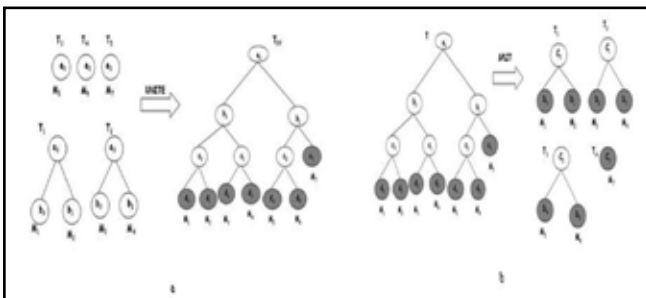


Fig. 2: Examples of a Key Tree Update After Applying, (a). Unite and (b). Split Procedures

B. Rekeying Using MDS Code on PFMH Tree

The PFMH tree follows a PACK protocol, in which each group member equally contributes its share to the group key, and this share is never relieved to the others. PACK includes a set of rekeying protocols to update the group key upon group membership change events for security purpose. The PACK protocol can achieve the minimum rekeying time cost upon membership change events. For any single-user Join event, the rekeying cost is $O(1)$, and for any single user leave event, the rekeying time cost is of $O(\log n)$. The communication and computation costs can still be reduced by adopting PFMH tree and by introducing phantom nodes in the key tree.

In this scheme, each member will maintain and update the global key tree locally. Each group member knows all the subgroup keys on its key path and knows the ID and the exact location of any other current group member in the key tree. In PACK, when a new user joins the group, it will always be attached to the root of the join tree to achieve $O(1)$ rekeying cost in terms of computation per user, time, and communication. When a user leaves the current

group, according to the leaving member's location in the key tree, as well as whether this member has a phantom location in the key tree, different procedures will be applied, and the basic idea is to update the group key in $O(\log n)$ rounds and simultaneously reduce the communication and computation costs.

Rekeying Using MDS with PFMH

```
MDS (k, n, key) // key generation using reed Solomon code
{
  nodei ← receives k-symbol message
  generate generator matrix using Vander Monde matrix
  n-symbol code ← applying generator matrix on k-symbol message
  other group members ← multicast n-symbol code
  nodej ← receives n-symbol code
  key ← decoding of n-symbol code using Inverse Vander Monde matrix
}
```

1. Rekeying on Single-User Join

When a new user M wants to join the group G , the PACK initiates the single-user join protocol by broadcasting a request message that contains its member ID, a join request, its own blinded key, some necessary authentication information, and its signature for this request message. After receiving this user join request message, the current group members will check and a new group key will be generated in order to incorporate a secret share from M . fig. 3, depicts the same and to guarantee the group keys' backward secrecy. The rekeying upon single-user join needs to perform two rounds of MDS code. Fig. 4, shows two examples of a key tree update upon single-user join events. In the first example tree consists of four members. After the new member $M5$ joins the group, a new node is created to act as the new root, and the node $(b1)$ becomes the new join tree that represents $M5$. In the second example, when $M6$ joins the group, at the first round, the MDS is first performed between $M5$ and $M6$ to generate a new join tree, at the second round, the MDS is performed between the new join tree and the main tree to generate a new group key.

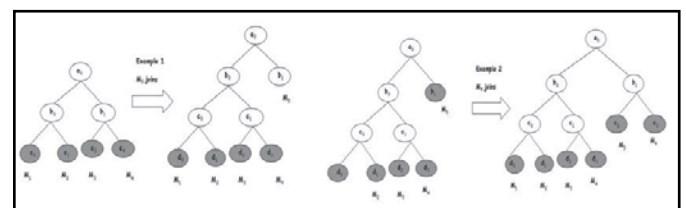


Fig. 3: Examples of a Key Tree Update Upon a Single-User Join Event

Rekeying for SUJ

```
G ← new user M
PACK ← Join request from M
Group members ← PACK broadcasts request message
MDS() ← Group members
New group key ← MDS()
New user ← new group key
```

2. Rekeying on Single-User Leave

When a current group member Y wants to leave the group, it broadcasts a leave request message to initiate the single user leave protocol, which contains its ID, a leave request, and a signature for this message. In order to reduce the rekeying cost upon a single-

user leave event, PACK creates a phantom node that allows an existing member to simultaneously occupy more than one leaf node in the key tree. Fig. 4, depicts the model of user leave and Fig. 6, shows one example of a key tree update upon single-user leave event. In this example, user M6 leaves the group where node (b0) is the root of the main tree and node (b1) is the root of the join tree. Since the size of the join tree is 2, the node representing M6 will be directly removed from the key tree, M5 changes its secret share, and a new group key will be generated by applying the MDS between M5 and the subgroup in the main tree.

Rekeying for SUL

```

PACK ← user M leaving G
Group members ← PACK broadcasts leave request
If (Join tree size > 2)
{
  Create phantom node
  Extend existing member to occupy left node
  MDS() ← Group members
  New group key ← MDS()
  Rest of group members ← new group key
}
Else
{
  Node is directly removed from tree
  New group key ← MDS() applied between subgroup in main
  tree and one node in join tree
}

```

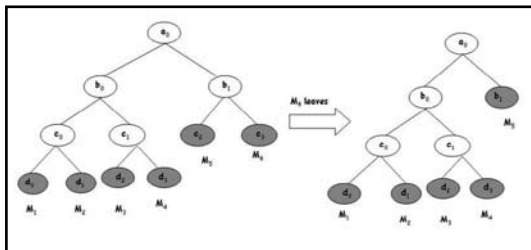


Fig. 4: An Example of Key Tree Update Upon Single-User Leave Event

3. Rekeying on Multi-User Join and Leave Protocol

PACK also has group merge and group partition protocols to handle simultaneously the join and leave of multiple users. Although multiple user events can be implemented by applying a sequence of single-user join or leave protocols, such sequential implementations are usually not cost-efficient. The group merge protocol, combines two or more groups into a single group, and returns a PF key tree. Group partition protocol, removes multiple group members simultaneously from the current group and construct a new PF key tree for the rest of the group members. In the group merge protocols, after removing all phantom nodes from those key trees corresponding to different subgroups, each key tree is split into several full key trees. The final result is obtained by uniting these full key trees into a PF tree using unite procedure. Similar to the group partition protocol, after removing all phantom nodes and leaving nodes, the original key tree is split into several full key trees, and the unite procedure is then applied on these full key trees to create a PF key tree. Since the height of the returned tree is $\log n$, where n is the group size after merging/partitioning, the time cost of group merge/partition is bounded by $O(\log n)$.

Rekeying for MUJ: // Group Merge Protocol

```

Combine two or more groups into single group
Construct the new PF tree
Remove phantom nodes in all groups
Split key tree into several full key trees
Apply unite() to obtain new key tree
Generate new key using MDS() and multicast it to group
members

```

Rekeying for MUL: //Group Partition Protocol

```

Remove two or more group members
Construct the new PF tree
Remove phantom and leaving nodes in all groups
Split key tree into several full key trees
Apply unite() to obtain new key tree
Generate new key using MDS() and multicast it to group
members

```

III. Result and Discussion

The experiments are carried out on an Intel Core 2Duo 2.80-GHz machine with a 2-Gbyte memory running windows xp. The implementation results of computations and communications are presented in fig. 5 and fig. 6. From these results; we can see that upon a single-user join event, PFMH has the lowest cost among all the schemes. Compared with GC, PFMH has more than 10 percent reduction in computation cost and a more than 65 percent reduction in communication cost and time cost. Compared with GC, the reduction is even more, about 50 percent in computation cost and about 80 percent in time and communication costs. Upon a single-user leave event, compared with GC, PFMH has about a 25 percent reduction in computation cost, about a 15 percent reduction in time cost, and a similar communication cost. Although PFMH has slightly higher computation and communication costs than GC upon a single user leave event, when averaged over both join and leave events, the reduction is still significant, with a 20 percent reduction in computation cost, 35 percent reduction in communication cost, and 40 percent reduction in time cost. Fig 7 and 8 shows the key distribution time and key recovery time of both the scheme under various multicast group sizes. . It is clear that using one-way hash functions adds none-trivial computation complexity. Nevertheless, the proposed scheme still outperforms the GC schemes by a significant margin. The computation time of the key distribution is also compared to GC for a selected multicast group size. Notice that the computation times of both the GC and the group member is significantly larger than proposed schemes.

Table 1: Sample Table

Multicast group size	PFMH Tree based key Distribution	Group Controller based key distribution
2	15.2 m. sec	27.6 m.sec
4	18.5m.sec	31 m.sec
6	21.5m.sec	35.8 m.sec
8	25.7m.sec	43.3 m.sec
10	27.5m.sec	46.6 m.sec

IV. Conclusion

We have optimized dynamic multicast key distribution scheme with MDS codes using PFMH tree. The computation complexity of key distribution is greatly reduced by employing erasure decoding of MDS codes instead of more expensive encryption and decryption computations. The MDS codes was combined with PFMH trees and performance of distribution time and key recovery time was evaluated, this scheme provides much lower computation complexity while maintaining low and balanced communication complexity and storage complexity for dynamic group key distribution. This scheme is thus practical for many applications in various broadcast capable networks such as Internet and wireless networks.

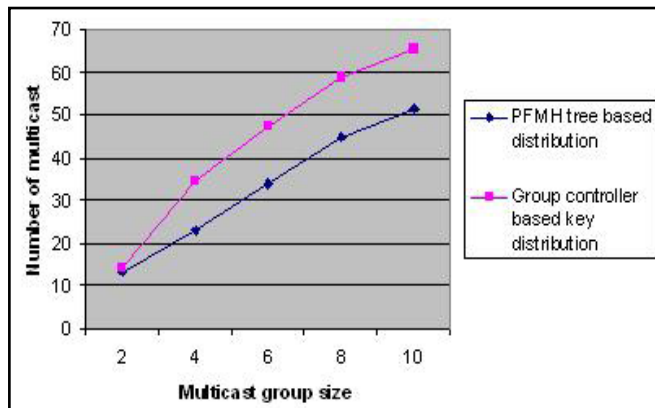


Fig. 5: Communication Cost

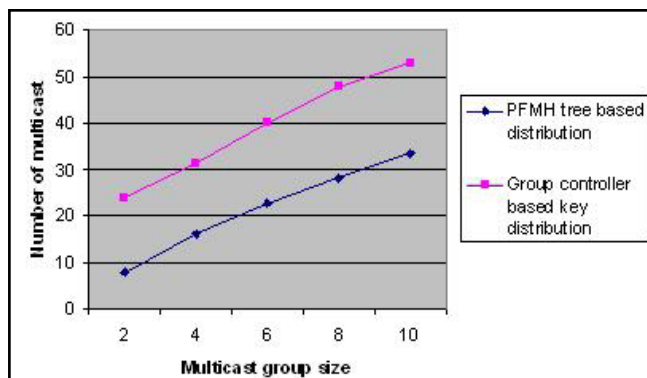


Fig. 6: Computation Time

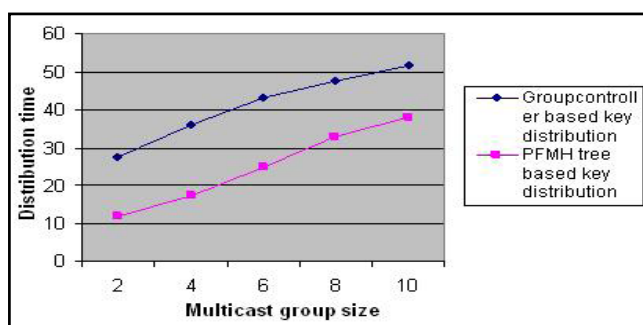


Fig. 7: Key Distribution Time

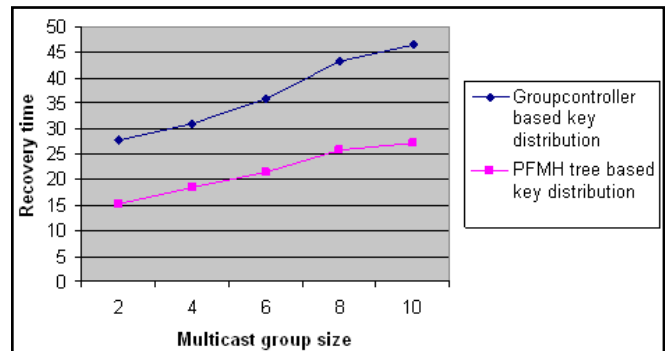


Fig. 8: Recovery Time

References

- [1] Lihao Xu, Cheng Huang, "Computation Efficient Multicast Key Distribution", IEEE Trans. Parallel And Distributed Systems, Vol. 19, No. 5, May 2008.
- [2] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, B. Plattner, "The VersaKey Framework: Versatile Group Key Management", IEEE Journal on Selected Areas in Communications, 7(8), pp. 1614-1631, August 1999.
- [3] S. Mittra, "Iolus: A Framework for Scalable Secure Multicasting", Proc. of ACM SIGCOMM'97, pp. 277-288, Sep. 1997.
- [4] D. M. Wallner, E. J. Harder, R. C. Agee, "Key Management for Multicast: Issues and Architectures", Internet Draft (work in progress), draft-wallner-key-arch-01.txt, Sep. 15, 1998.
- [5] C. K. Wong, M. Gouda, S. S. Lam, "Secure Group Communications Using Key Graphs", Proc. ACM SIGCOMM'98, Sep. 1998.
- [6] Y. Kim, A. Perrig, G. Tsudik, "Tree-Based Group Key Agreement", ACM Trans. Information and System Security, Vol. 7, No. 1, pp. 60-96, Feb.
- [7] S. Benson Edwin Raj, J. Jeffneil Lalith, "A Novel Approach for Computation-Efficient Rekeying For Multicast Key Distribution", IJCSNS, Vol. 9, No. 3, 2009.
- [8] Wei Yu, Yan Sun, K. J. Ray Liu, "Optimizing the rekeying cost for Contributory Group key Agreement Scheme".
- [9] I. S. Reed, G. Solomon, "Polynomial Codes over Certain Finite Fields", J. SIAM, 8(10), pp. 300-304, 1960.
- [10] F. J. MacWilliams, N. J. A. Sloane, "The Theory of Error Correcting Codes", Amsterdam: North-Holland, 1977.



Mrs. S. Sasikala Devi is a doctoral student in Manonmaniam Sundaranar University, Tirunelveli. She is working as an Assistant Professor in CMS college of Science and Commerce, Coimbatore. Her research interest lies in Network security, Cloud computing. She has presented 5 papers and published 2 papers in international journals.



Dr. Antony Selvadoss Thanamani, was an Associate Professor & Head, Department of Research and Development in Computer Science and presently working as a Reader in NGM college of Arts and Science Coimbatore, Tamilnadu, India. With a teaching experience of about 22 years. His research interest includes knowledge management, web mining, networks, mobile computing, and telecommunication. He has guided 41

M.Phil scholars, attended 15 conferences, presented 30 papers, published about 8 books and 16 papers.