DHA-KD: Dynamic Hierarchical Agent Based Key Distribution in Group Communication *

M. Amir Moulavi
Networking Department
Information Technology Services (ITS) Center
Ferdowsi University of Mashhad, Iran
moulavi@acm.org

Jalal A. Nasiri
Computer Engineering Department
Ferdowsi University of Mashhad, Iran
j.nasiri@wali.um.ac.ir

Behnam Bahmani
Computer Engineering Department
Islamic Azad University of Firoozkooh, Iran
b.bahmani@iaufb.ac.ir

Hossein Parvar
Department of Computer Engineering
Islamic Azad University of Mashhad, Iran
parvar@mshdiau.ac.ir

M. Sadeghizadeh
Computer Engineering Department
Ferdowsi University of Mashhad, Iran
ma_sa638@stu-mail.um.ac.ir

Mahmoud Naghibzadeh
Computer Engineering Department
Ferdowsi University of Mashhad, Iran
naghibzadeh@ferdowsi.um.ac.ir

Abstract

Confidentiality is one of the most nontrivial issues in secrecy of group communication. In order to satisfy confidentiality, symmetric cryptography must be exploited. All symmetric algorithms require a shared key between the members of group. High frequency in the number of joins and leaves of members can cause huge number of messages of data between authorized group members. Although hierarchical group communication is a prominent model, it has not been investigated in security literature as well as other models. There has been an extensive research on satisfying confidentiality in group communications on different architecture other than the hierarchical architecture and the proposed hierarchical models do not use the concept of agents. A concept that brings its own advantages. In this paper, we investigate a new approach to secure key management in hierarchical group communication by means of using intelligent agents. The simulation results show that using agents reduce network bandwidth and improve and achieve more performance comparing with other models. Moreover, using agents makes the architecture more flexible and dynamic preparing it for Grid Computing technologies.

*This work was supported by Communications and Computer Research Center, Ministry of Information Technology, Mashhad, Iran.

1. Introduction

Many group communication applications such as video conferencing, interactive group games and video on demand are based upon Multicasting which is an efficient communication model [12]. They require packet delivery from one or more authorized sender(s) to a large number of authorized receivers. Since IP multicast sends the packets over a multicast tree that spans all the member, network bandwidth is saved efficiently. In order to protect the top-secret data from being accessed by non-authorized members, an appropriate confidentiality mechanism should be proposed in the group. This method needs that only valid members could encrypt/decrypt data or generally modify data. Symmetric/Asymmetric cryptography mechanisms can be exploited. However for lowering the complexities of confidentiality and security, it can be assumed that members of groups use symmetric approach. In this case, a symmetric key is used by the sender to encrypt the data and also is used by the receiver to decrypt it. This key is called *Traffic Encryption Key* (TEK).

In order to satisfy the secrecy criteria, a rekeying process should be performed after each join/leave in the group because a member which leaves the group should not have access to transfered data among previous members. This also applies to the join scenario. This process consists of generating a new TEK and distributing it among the

978-0-7695-3263-9/08 $25.00 © 2008 IEEE
DOI 10.1109/SNPD.2008.23

current members of the group. A critical issue in this naive approach is maintaining the scalability. Since the rekey process should be performed after each modification in the group vision which occurs dynamically, it might be nontrivial in the case of high frequency of joins and leaves. Many solutions have been proposed in this area for this problem that can be divided into: *centralized*, *hierarchical*, and *distributed* architectures. In the *centralized* approach [3][11][16][19][20][21][8], a specific entity assures the key generation, distribution and rekeying process. Goal of this approach is to minimize the storage requirements and computational power. In the *hierarchical* approach [4][7][9][10][14][17][15], a hierarchy of mangers perform the key distribution. Finally in *distributed* approach [5][6][13][18], a set of entity collaborate to agree on a group key and distribute the key in the group. There is no explicit KDC in this approach.

In this paper, we investigate a new approach for dynamic hierarchical agent-based key distribution in group communication (DHA-KD). The proposed algorithm saves huge amount of network bandwidth efficiently comparing to Hi-KD algorithm [12]. This is achieved by considering *Computational Capacity* (CC), *Load* and *Local Network Traffic* (LNT).

The remainder of this paper is organized as follows: in section 2, we present an architectural model. The related works is discussed in 3. In section 4, we propose our agent-based model for different scenarios. Simulation results and comparison with other similar model is described in section 5. Finally we conclude in section 6.

2. Architectural Model

In this section we discuss the basic architecture as building block which will be used in our algorithm. Let G be the set of all members of group. The group is divided into J subgroups (classes) G_i, $0 < i < J$. Every class i has N_i members and the k^{th} member of the n^{th} class is indicated by $m_{n,k}$. In every class G_i there is an agent $agent_i$ that acts as a coordinator in that class. This agent distribute the new key among members and also manages that class membership. These agents from different classes can communicate with other agents to update their key management information.

To satisfy the basic rules which are mentioned in the previous section, every class G_i and every member $m_{i,j}$ should maintain a secret key k_i for encryption/decryption or generally any modification of data. In order to achieve this goal, every key distribution method can be used. In this paper, we focus on TEK management, however.

We have two type of essential problems in hierarchical group communication. The first one is meeting the new security requirements which was discussed in the previous section. The other issue is the method of regenerating the keys which will be discussed in forthcoming sections.

3. Related Works

In this section we briefly discuss two main approaches for regenerating the keys after every changes in group vision which is referred as *Rekeying Process*: *Naive* and *Hi-KD*.

3.1. Naive Approach Rekeying

In this approach rekeying process is fired whenever a change is occurred in group vision. In H. R. Hassan et al. [12] a naive solution is discussed to satisfy the confidentiality requirements. Its main idea is that in order to allow a given member $m_{n,k}$ to decrypt messages of class w, $w > n$, he should know the key k_w. To accomplish that, it is sufficient that member $m_{n,k}$ has all the keys k_g, such that $n \leq g$. Hence $m_{n,k}$ should store $J - n + 1$ traffic encryption keys. For example, if G consists of 5 classes, then $m_{2,i}$ should have the keys of classes 2, 3, 4 and 5: k_2, k_3, k_4, k_5 respectively.

The problem here is that if a key k_w is renewed, members of classes g, $g \leq w$ should update their keys. A naive solution can be that on every join or leave, a new key is generated for that class and this new key is distributed among higher and lower ranks. Hence, members of a class like G_i, should store all the keys k_w, $i \leq w$. Rekeying process must be started in four different cases: *join*, *leave*, *promotion* and *degradation* of a member. All of the four cases are discussed in the following.

3.1.1 Join and Leave

Join of a new member to class r causes the rekeying process for all classes u, $r \leq u$. If this rekeying process does not performed, then a new member can decrypt old messages sent before his membership. Join or backward secrecy has the following rekeying scheme. First, generate a new key for r^{th} class. Then, generate new keys for all lower rank classes u, that is $r \leq u$ and the last is to distribute the keys to lower rank classes.

Leave or forward secrecy of a member from the class r causes the rekeying process for all classes u, $r \leq u$ like the previous scheme.

3.1.2 Promotion and Degradation

Promotion or upward secrecy of a member from class r to the class u, $u < r$, consists of two rekeying process: a join to class u and a leave from class r. Thus the rekeying

scheme is as follows. First, generate new keys for classes g, $u \leq g \leq r$ and then distribute the new keys.

When degradation or downward secrecy of a member from class r to class u occurs, the member should no longer have access to the key. Thus the rekeying scheme for this case is the same as the promotion.

The naive solution has a critical problem which is the high overhead and frequency of key exchange. There is a redundancy between keys maintained by different classes. This is because the naive approach is based on this assumption that in order to know a key, one must posses it. Simply, communication between different classes becomes crowded because of high frequency in the exchange of keys. On each leave or join, keys should be regenerated to meet the requirements of hierarchical group communication confidentiality.
As a result, this solution needs extra bandwidth for rekeying process and also storage capacity for storing group keys.

3.2. Hi-KD Approach

In [12], an alternative solution is proposed which is called Hi-KD. Hi-KD reduces the overhead proposed by naive solution. It bases on random key and uses hash function to compute a chain of keys. With every changes in group vision (join, leave, etc.), a random key is generated and is sent to the same group and upper group members. Base on hash function, this newly generated key is the input to hash function that again generate the key for its successor. This pattern is repeated for lower groups. This mechanism is illustrated in Fig. 1.

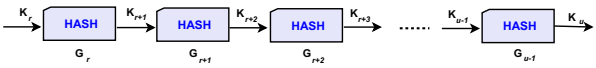

Figure 1. Key Generation Mechanism in Hi-KD approach

Comparing to Naive solution, the problem of bandwidth requirement and storage capacity has been considerably reduced by this mechanism. However, The problem with Hi-KD is that on every join and leave, new keys should be generated and send over members of all hierarchical groups. Our approach has solved this recent problem.

4. Our Agent-Based Approach

The proposed method mainly involved with key distribution and improving the join rekeying scheme. The former is accomplished by agents and the latter is done on particular periods.

Our agent-based approach uses a random key K^r and hash function H to compute a chain of keys like Hi-KD by using this following formula

$$\delta^1_{c+1} = H(\delta^1_c) \tag{1}$$

where $\delta^1_1 = K^r$, and δ^p_t is the t^{th} class key to use after the p^{th} key renew. Then each key k_n is sent to the corresponding class n. Once the member $m_{n,k}$ of the class n receives his key, he can compute other classes w, $w > n$ if needed, by applying $w - n$ times hashing function on k_n. This solution exploits a new join and leave rekeying scheme that reduces the bandwidth. These schemes are discussed in the following sections.

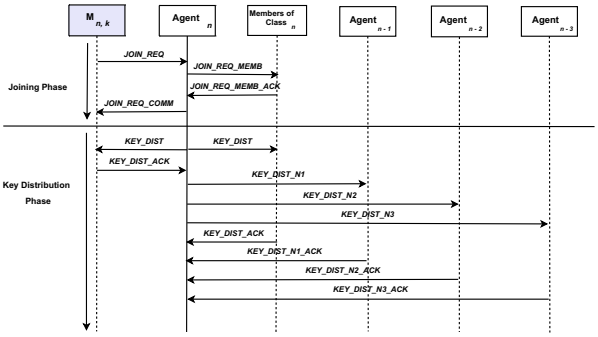

Figure 2. Sequence of Join Mechanism

4.1. Join Rekeying Scheme

Join Rekeying scheme is consist of two main phases: *Joining Phase* and *Key Distribution Phase*. The whole sequence diagram is illustrated in Fig. 2. In Joining phase, each agent in each class is responsible for managing the join and leave of members as described earlier. When a new member wants to join to class G_i, he sends a request to the $agent_i$ indicating his willing to join to that class (*JOIN_REQ*). The corresponding agent collects all incoming requests for joining to that class, and applies the rekeying generation process on the end of each τ periods. The coordinator then sends a message to other members of group indicating the new member (*JOIN_REQ_MEMB*). Every member replies to the coordinator with its acknowledgment message (*JOIN_REQ_MEMB_ACK*) showing that he knows the existence of the new member. The last step in this phase is that the agent should commit the join of this new member by replying back to him (*JOIN_REQ_COMM*). According to real simulation results in [1], on average a number of 1706 new members want to join to a class like G_k, which their requests are sent one after another with a very little time left between. Using other schemes, the corresponding agent should generate a new key for that class and send

the key to other lower rank classes for each separate new request. This becomes more important and serious, when the classes are dispersed in a wide-area system. But by using periodic rekeying generation, used bandwidth is reduced excessively.

In our solution, a new member should wait to receive the new key until the end of current τ period. Fig. 2 shows only one join request for member $M_{(n,k)}$. At the end of each τ period, the corresponding agent $agent_k$ of G_k class check to see whether there are any request for join. If there exists one or more, it acts as follows:

- The key generation mechanism generates a new random key δ_k^{p+1} and send this key to other members of group (*KEY_DIST*);
- It computes the chain δ_s^{p+1}, $k > s$ using H function;
- Finally, it sends every key δ_s^{p+1}, to corresponding class k, $k > s$ (*KEY_DIST_N*$_s$).

Instead of sending an update message to upper-ranked classes in order to update their key data bases, we use pull-based protocol to fulfill the requirements of confidentiality. There are two general approaches for pull-based protocol. The first one is *Periodic* in which each agent pulls new information updates from lower rank classes in τ periods of time. The second one is *Event-driven* in which each agent trigger the rekeying process for generating new keys and sends an update message to upper-ranked classes.

Our proposed algorithm exploits periodic pull-based protocol. Since the average number of requests that is sent to a group is considerable, event-driven approach can conduct to Hi-KD mechanism. As a result, we choose periodic pull-based protocol, in which every agent, $agent_i$, pulls new update regarding key information in τ periods of time from its lower rank class G_k, $k > i$.

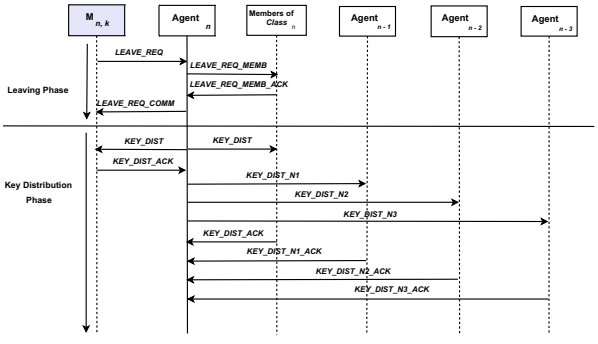

Figure 3. Sequence of Leave Mechanism

4.2. Leave Rekeying Scheme

Leave Rekeying scheme is also consist of two major steps: *Leaving Phase* and *Key Distribution Phase*. This is depicted in Fig. 3. As it might be thought, the leave rekeying scheme is similar to the join scheme, but we can not apply the scheme used for join. When a member send a message to its class's agent requesting to leave the current class (*LEAVE_REQ*), it should not have access to the key from that time, because it will be marked by the agent as a removed member, but he can still decrypt messages from the class.

In order to accomplish this, we can not use updating in τ periods of time. Instead new generated key from every leave of a class should be propagated immediately (*LEAVE_REQ_MEMB*). Agents corroborate leave requests immediately to satisfy the requirements of confidentiality. Hence, the scheme is used in leave mechanism is as follows:

- The key generation mechanism generates a new random key δ_k^{p+1} and send this key to other members of group (*KEY_DIST*);
- It computes the chain δ_s^{p+1}, $k > s$ using H function;
- Finally, it sends every key δ_s^{p+1}, $k > s$ to its corresponding class k (*KEY_DIST_N*$_s$).

4.3. Promotion and Degradation Rekeying Scheme

As discussed earlier in the paper, every promotion and degradation consists of a join and leave scheme. Hence to apply the solution for promotion and degradation and satisfy the requirements for confidentiality, explained schemes for join and leave in previous sections must be exploited.

4.4. Agent Election

There might be some conditions that the agent, $agent_i$, itself willing to leave a class. In this case a new agent should be elected from the remaining members of class. If the agent is selected randomly, there might be situations that the selected agent is not powerfull enough to take the responsibility of a coordinator. Moreover, network traffic changes over time and there may be some bottlenecks in the whole group. In order to overcome this problem, an appropriate agent should be selected with specific factors from the remaining members. There are some issues that should be taken into account when electing the new coordinator:

- *Rekeying Process Construction*: Since this process is a time consuming task, the selected coordinator should have a high Computational Capacity (**CC**) and low working set (**load**) for a faster key regeneration.

- *Network Traffic*: Since the generated key should be sent to other members of the current group and agents of upper groups, the Local Network Traffic (**LNT**) of selected agent should be low in order to reduce the time to send the keys.

Hence, we should compute the ability of each agent and select the agent with the maximum ability. The ability is calculated from the following equation:

$$ability = \frac{1}{3}(CC + \overline{Load} + \overline{LNT}) \quad (2)$$

Where CC is Computational Capacity, $Load$ is the amount of load which is executed on the system and LNT is the local network traffic. All of the parameters are normalized to the range $[0, 1]$. $\overline{Load}$ and $\overline{LNT}$ is the negative of them (i.e. $1 - Load$). The produced $ability$ value may a duplicate. This does not cause any problem to the main algorithm because one of these values is selected. $ability$ is computed on every change in group vision. Since the equation is computed on dynamic values, in every rekeying process, the best coordinator among other members is elected.

To describe the algorithm [2] for agent election, we must first define the $k - neighbors$ as the set of members with k distance from the $m_{n,k}$ member in the n class. The algorithm operates in phases. If a member becomes winner in the k^{th} phase, it can proceed to $(k+1)^{th}$ phase. In order to become a winner in the k^{th} phase, it must have the largest id in its 2^k-neighbors. Hence, fewer members proceed to higher phases and at the end only one member wins the last phase and that is the agent in that class. The pseudocode for this algorithm is indicated in Fig. 4.

In general, in phase k, a member $m_{n,k}$ that is the winner of $k - 1$ phase, sends <prob> message with its identifier to its 2^k neighborhood. A probe is swallowed by a member if it contains an identifier smaller than its own identifier. If the <probe> message arrives at the last member in the neighborhood without being swallowed, then the last member sends back a <reply> message to $m_{n,k}$. If $m_{n,k}$ receives reply from both directions, it becomes the winner in that phase. A member that receives its own <probe> message terminates the algorithm as the leader.

Efficient Agent Election

```
1:  ability = 1/3 (CC + Load-bar + LNT-bar)
2:  Initially, asleep = true
3:   Upon receiving no message
4:    if asleep
5:     asleep ← false
6:     send <probe, ability, 0, 1> to left and right
7:   Upon receiving < probe, j, k, d > from left
8:    if j = ability terminate as the leader
9:    if j > ability and d < 2^k
10:    send < probe, j, k, d + 1 > to right
11:   if j > ability and d ≥ 2^k
12:    send < reply, j, k > to left
13:  Upon receiving < reply, j, k > from left
14:   if j ≠ id send < reply, j, k > to right
15:   else
16:    if already received < reply, j, k > from right
17:     send < probe, ability, k + 1, 1 >
```

Figure 4. Agent Election, Pseudocode for $m_{n,i}$

5. Simulation Result

In this section, we provide an overview of our simulation model which is obtained from NS2 by comparing to previous solutions and our proposed algorithm. The result of this comparison is indicated in Fig. 5. The simulations are provided as follows; to generate a real multicast session, we used the modules presented by Almerath et al. in [1]. These models suggest that the arrival of members follows a Poisson process and the membership duration follows as Exponential distribution.

We have found no statistical studies or distribution of hierarchical group membership yet, but we assume that distribution in each class is exponential and changes over time. Each subgroup has its own agent that acts as a coordinator and manages key distribution and membership services.

In our simulation, based on real applications, sending new key and update message are made by one group class step. Each member starts the session by joining the group and at the end of membership in some group class either leave or rejoin (promotion or degradation) to another group class with different probability. We consider sessions of 3 hours, a joining member rate of 20 second and average membership duration of 30 minutes.

In this simulation, we have compared bandwidth performances of our solution and previous. Bandwidth refers to number of keys and messages sent per member for rekeying when membership changes. Leave rekeying scheme is like previous one but there is different in joining process which save up to 22.5% of the bandwidth overhead.

6. Conclusion

In this paper we have investigated the hierarchical group communication model which is a very promising and reliable model comparing with other proposed architecture for group communication. We have proposed a new approach

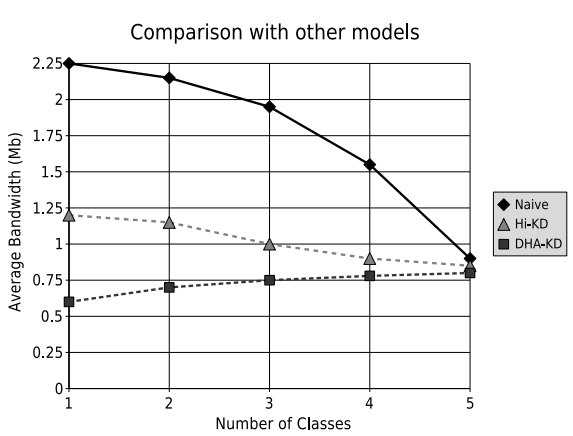

Figure 5. Comparsion with other models

to secure key management in hierarchical group communication by means of using intelligent agents. These agents act as coordinators in group communication and handles all sort of responsibilities for their group. A new algorithm is also proposed in this paper for the case of a leaving agent from the group which another member should be elected from the remaining members. The selected agent should have high ability for accomplishing various tasks. In order to meet these requirements, we used a dynamic formula to choose the best member for substituting the coordinator.

Simulations have shown that our agent-based algorithm is improved comparing to previous proposed mechanisms such as Hi-KD. It reduces the bandwidth by more than 22.5%. Further works can focus on an efficient protocol.

References

[1] K. C. Almeroth and M. H. Ammar. Collecting and modeling the join/leave behavior of multicast group members in the mbone. In *HPDC*, pages 209–216, 1996.

[2] H. Attiya and J. Welch. Distributed computing, fundamentals,simulations, and advanced topics. In *Wiley-Intersicence*, 2004.

[3] D. Balenson, D. McGrew, and A. Sherman. Key management for large dynamic groups: One-way function trees and amortized initialization. In *Internet-Draft*, Feburary 1999.

[4] A. Ballardi. Scalable multicast key distribution, May 1996.

[5] C. Becker and U. Wille. Communication complexity of group key distribution. In 5^{th} *ACM Conference on Computerand Communications Security*, November 1998.

[6] C. Boyd. On key agreement and conference key agreement. In *Information Security and Privacy: Australasian Conference, LNCS(1270)*, pages 294–302, 1997.

[7] B. Briscoe. Marks: Multicast key management using arbitrarily revealed key sequences. In 1^{st} *International Workshop on Networked Group Communication*, November 1999.

[8] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast security: A taxonomy and some efficient constructions. In *In Proceedings of the IEEE INFOCOM. Vol. 2. (New Yok, N.Y., Mar.)*, 708-716. 1999.

[9] L. R. Dondeti, S. Mukherjee, and A. Samal. Scalable secure one-to-many group communication using dual encryption. In *Computer Communications, 23(17)*, pages 1681–1701, November 2000.

[10] T. Hardjono, B. Cain, and I. Monga. Intra-domain group key management for multicast security. In *IETF Internet draft*, September 2000.

[11] H. Harney and C. Muckenhirn. roup key management protocol (gkmp) architecture. In *RFC 2093*, July 1997.

[12] H. R. Hassan, A. Bouabdallah, H. Bettahar, and Y. Challal. An efficient key management algorithm for hierarchical group communication. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks*, 2005.

[13] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In 7^{th} *ACM Conference on Computer and Communications Security*, pages 235–244, November 2000.

[14] S. Mittra. Iolus : A framework for scalable secure multicasting. In *ACM SIGCOMM*, 1997.

[15] M. A. Moulavi and H. Parvar. Agent based bandwidth reduction for key management in hierarchical group communication. In 2^{nd} *IEEE/Create-Net/ICST International Conference on Communication Systems Software and Middleware*, pages 1–5, 2007.

[16] A. Perrig, D. Song, and J. Tygar. Elk, a new protocol for efficient large-group key distribution. In *IEEE Security and Priavcy Symposium*, May 2001.

[17] S. Rafaeli and D. Hutchison. Hydra: a decentralized group key management. In 11^{th} *IEEE International WETICE: Enterprise Security Workshop*, June 2002.

[18] O. Rodeh, K. Birman, and D. Dolev. Optimized group rekey for group communication systems. In *Network and Distributed System Security*, February 2000.

[19] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner. The versakey framework : Versatile group key management. In *IEEE Journal on Selected Areas in Communications (Special Issues on Middleware), 17(8)*, pages 1614–1631, August 1999.

[20] C. Wong and S. Lam. Keystone: A group key management service. In *International Conference on Telecommunication*, May 2000.

[21] C. K. Wong, M. Gouda, and S. S. Lam. Secure group communications using key graphs. In *IEEE/ACM Transactions on Networking, 8(1)*, pages 16–30, February 2000.