An Efficient Improved Group Key Agreement Protocol Based on Diffie-Hellman Key Exchange

YANG Guang-ming, CHEN Jin-ming, LU Ya-feng
Department of Information Security, Software School
Northeastern University
Shenyang 110819, R. P. China
e-mail: ygm5@hotmail.com

MA Da-ming
Dalian's Net Management Center of Shenyang Branch
The Electron Devices and System Engineering Company of China
Dalian 116021, R.P. China
e-mail: madaming@sina.com

Abstract—Traditional group key generating methods using binary-tree performs efficiently when adding or deleting nodes dynamically. However, when dealing with paroxysmal needs for group key among multi-nodes in the Internet, their time complexity grows to O(Nlog2N). We propose an improved group key agreement protocol based on Diffie-Hellman key exchange, which can reduce the time complexity of this situation to O(N). It can largely reduce overlapping computation and data packages' sending times, and meanwhile, it remains binary-tree in each node, enabling it efficiently to support dynamically adding and deleting nodes as well. And then the security of this method is been analyzed and a basic idea for preventing MITM attacks is been presented.

Keywords*：*Group key; Diffie-Hellman key exchange;Group Cmmunication

I. Introduction

With the rapid development of the Internet, it is becoming more and more common to apply the group key. For example, in order to secure the multi-point communication in the Internet in the cases of multicasting or to take a videoconference inside a company, it is necessary to negotiate the common key[1]. In the past researches, most generating and managing methods of the group key are based on the Diffie-Hellman key-exchange protocol[2], and maintain the key via the structure of binary-tree[3] which intends to reduce the time complexity to $\log_2 N$ of group key's dynamically changing. However, more often than not, the need for multicasting is paroxysmal, for example when N computers want to hold a real-time videoconference immediately, the time complexity for maintaining the group key using the traditional binary-tree method increases to $N\log_2 N$, along with a lot of packages sent for key exchange, and in addition, there is lots of redundant computing which delays the procedures of generating group key.

This method for generating group key rapidly is designed to meet the need of the security in paroxysmal multi-node communication. Compared with traditional binary-tree generating methods, this method points at reducing negotiating steps, cut down overlapping computation, decrease the traffic of data exchange, make the time complexity down to O(N), and meanwhile maintain basic binary-tree in each node in order to remain the $\log_2 N$ time complexity of dynamically adding and deleting nodes as the traditional methods .

II. Algorithm Descriptions

This method of rapidly generating group key is still based on the Diffie-Hellman key exchange protocol, maintaining by binary-tree. Nevertheless, it has some differences from traditional methods in both generating and maintaining procedures. There are two major differences: first there is a managing node which is responsible for gathering and dispatching information of generating group key; second the algorithm is different between managing node and non-managing node to maintain its binary-tree.

Say here are m nodes, numbered as N_1, N_2, N_3, …, N_{m-1}, N_m, that want to negotiate a common key immediately. There are two rounds of sending data packages, *Send1* and *Send2* respectively.

A. Select Managing Node

First of all, we should select one node as managing node, which will be responsible to maintain the topology structure of a complete binary-tree from the m nodes. Generally it is preferred to select the computer with the strongest capability as the managing node, since it will improve the performance of generating and maintaining the group key if this node has strong computing capability according to our algorithm.

If the *m* nodes are equal in their computing capability, selection will be carried on according to IP addresses. Before communication begins, they need to know other members' IP addresses which are inputted by the manipulators. IP address is an unsigned 32 bits integer, so we can select the one with the smallest or the largest as the managing node.

According to the selection criteria above, now say the node N_m is selected as the managing node.

B. Generate Complete Binary-tree by Managing Node

During the *Send1* phase, each non-managing node sends its Diffie-Hellman key exchanging "ingredient" to the managing node. After receiving m-1 pieces of ingredients from others and adding its own ingredient, the managing node can generate and maintain a complete binary-tree inside its memory, also calculate the group key using the binary-tree.

978-1-4244-5848-6/10/$26.00 ©2010 IEEE

In accordance with whether *m* is the positive integer power of 2, the procedure can be divided into two different situations, shown as follows.

1) m is x power of 2 ($m = 2^x$): When *m* is the power of 2, the binary-tree for managing node is a full binary-tree. All the host nodes are in the same layer in the tree.

Take 8 nodes ($8 = 2^3$) as an example. Fig. 1 illustrates the binary-tree's structure.

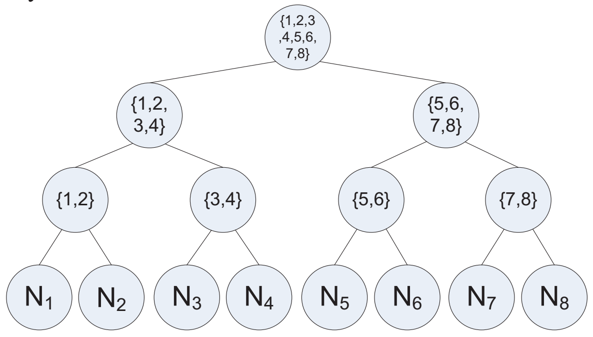

Figure 1. Binary-tree's structure of 8 nodes

2) m is not x power of 2 ($m != 2^x$): When *m* is not the power of 2, the binary-tree is a complete binary tree instead of a full binary tree, which means all the host nodes are not in the same layer.

Take 7 nodes as an example. Fig. 2 illustrates the binary-tree's structure.

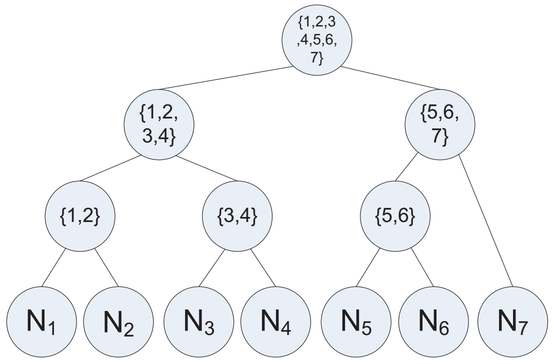

Figure 2. Binary-tree's structure of 7 nodes

C. Managing Node Sends Information to Non-managing Nodes

In the *Send2* phase, according to different positions of non-managing nodes, the managing node selects necessary information from its binary-tree and sends the different information to each non-managing node. Take m = 8 as an example. N_8 is the managing node presumed, and Fig. 3 shows the information it sends to N_1.

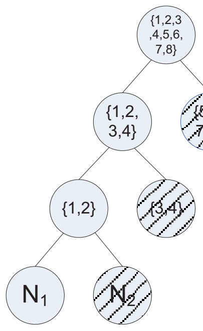

Figure 3. Information sent to N_1

In Fig. 3, the shaded part is the information sent by the managing node. Compared with the complete binary-tree, N_1 maintains a simplified binary-tree like Fig. 3.

This dispatching way greatly reduces the redundant computation in non-managing node, besides it shrinks the times of sending data packages to the minimum, because the managing node only selects necessary part of information from the complete binary-tree to each non-managing node.

III. Dynamically Adding and Deleting

After m nodes having come up with the group key, there are occasions when some nodes are needed to be added into or deleted from the group dynamically. When the composition of nodes is modified, the group key should be renegotiated, aiming at ensuring the security.

Since each node remains a binary tree inside its memory in this method, it becomes possible to support a quick and dynamic node adding and deletion.

There are two different situations of dynamically adding and deleting: first nodes' dynamically adding; second nodes' dynamically deleting.

A. Nodes' Dynamically Adding

No matter whether it is the managing node or non-managing node, the updating operation for adding new node will be completed within $\log_2 N$ layers in itself binary-tree for each node. Regarding to the managing node, the dynamic updating operation includes the following procedures:

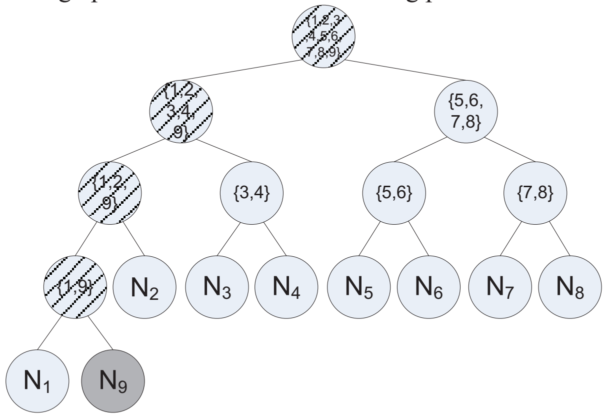

Figure 4. Update the managing node

Fig. 4 illustrates the updating procedures of managing node. Node N_9 (the grey circle) is the new added node, and the managing node still maintains a complete binary tree. The binary tree will be updated down-to-up, and the shaded area is the inner nodes that should be updated.

After updating itself, the managing node takes its responsibility to send necessary updating information packages to other non-managing nodes. In accordance with non-managing nodes' previously existing data, these packages' contents are selected from the managing node's updated complete binary-tree. Among all the non-managing nodes, N_1, N_2, N_4, N_7 are took for examples.

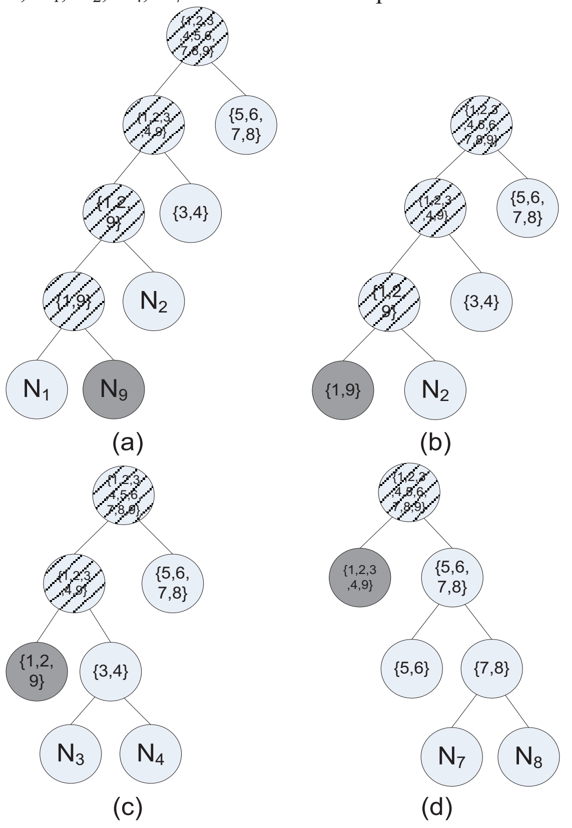

Figure 5. Update non-managing nodes

Fig. 5 illustrates the updating of non-managing nodes. The grey part is the updating information sent by the managing node, while the shaded part is the updating path where the non-managing node tries to maintain its simplified binary-tree. In Fig. 5 (a), node N_1 takes 4 steps in updating its binary tree after receiving the information of new added N_9, which can be figured out from the shaded part. In Fig. 5 (b), N_2 receives the packed information of nodes N_1 and N_9, so it only needs 3 steps in updating. The same goes for node N_4 and N_7 that they generate the new group key with only 2 and 1 updating steps respectively. However, it is known that the traditional binary tree method has to trouble to update thoroughly its complete binary tree for every node, in which it inevitably results in lots of redundant computing. On the contrary, this new method only to update the managing node's binary tree completely and then send selected information to non-managing nodes, in this way it avoids the many identical time-consuming calculations effectively and speeds up the procedures of generating the new group key.

B. Nodes' Dynamically Deleting

The deletion of nodes can also be divided into two situations: first the deletion of non-managing node; second the deletion of managing nodes.

1) Deletion of Non-Managing Nodes: The deletion of non-managing nodes is similar to the adding, beginning with updating the information stored in the managing node, followed by sending the necessary information selected by managing node to every non-managing node. Those non-managing nodes then update themselves' binary trees respectively, and finally complete the whole updating.

2) Deletion of Managing Node: Before deleting the managing node, a new managing node should be selected first based on the selection regulation. And then, the previous managing node transfers its binary-tree structure to the new managing node using the old group key. Eventually, the deletion of the managing node is carried on as the way of non-managing one.

IV. PERFORMANCE ANALYSES

We will evaluate this method in two aspects: the initial negotiating procedure of group key and dynamically node adding and deletion. Since this method aims at enhancing the speed of paroxysmal generating group keys majorly, the initial negotiating procedure of group key is the main analysis target.

A. 1Initial Negotiating of Group Key

Having received other *m*-1 nodes Diffie-Hellman key exchange ingredients, the managing node begins to construct its binary tree. Unlike the traditional method of maintaining binary tree, this method starts computing after gathering all the needed information, which means that each inner node is calculated only once. As a consequence, the time complexity is $\log_2 N$ rather than the traditional $N\log_2 N$.

Now there are m nodes as presumed, so the binary tree has $\lceil \log_2 m \rceil$ layers. It has m leaves and *m*-1 inner nodes (non-leaves), because it is a complete binary tree. Say that the i^{th} layer Diffie-Hellman calculation needs μ_i of time for each. The i^{th} layer has 2^i nodes (supposed that the last layer is complete), so the time for computing in the nodes is calculated as follows:

$$T_1 = \sum_{i=1}^{\lceil \log_2 m \rceil} \mu_i \cdot 2^i \qquad (1)$$

Since the consuming time in each layer is similar, μ is taken to stand for it. So T1 may be defined as follows as well:

$$T_1 = \mu \cdot (N-1) \qquad (2)$$

If each data sending phase consumes *t* of time, the total time consumed by two sending phases is computed as follows:

$$T_2 = 2t \qquad (3)$$

It only takes one time of computing in each layer of non-managing nodes, so the computing time is equal to the following:

$$T_3 = \sum_{i=1}^{\lceil \log_2 m \rceil} \mu_i \tag{4}$$

It may be simplified using the following:

$$T_3 = \lceil \log_2 m \rceil \cdot \mu \tag{5}$$

So at last, the total consuming time for initial generating group key may be defined as follows:

$$T = T_1 + T_2 + T_3 = 2t + \mu \cdot [(m-1) + \lceil \log_2 m \rceil] \tag{6}$$

If the capability of managing node is strong enough, T1 may be largely reduced. Overall, the time complexity of this method is O(N), while the one of the traditional method is $N\log_2 N$ due to the fact that each node maintains a complete binary tree.

Even though both this method and the traditional method use binary-tree to generate the group key, this method begins its computing when all the necessary ingredients have been gathered, which enables the node to get rid of overlapping computing workload, and meanwhile reduces the redundant computing in non-managing nodes, and eventually makes the overall time complexity down to linear.

B. Nodes' Dynamic Adding and Deletion

Regarding with adding and deleting nodes dynamically, the time complexity is same for the managing node compared with the traditional method, which is $\log_2 N$. However the average time complexity for non-managing nodes may be much smaller than $\log_2 N$, because the information stored in each non-managing node is limited to the minimum received from the managing node after its selection, which largely reduces the time cost in the non-managing nodes. Considering that the computing procedures cannot be parallel between the managing node and non-managing nodes, theoretically this method consumes more time when dealing with these dynamic operations. However, if the capability of the managing node is strong enough, this shortcoming can of course be covered.

V. Security Analyses

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper.

In terms of traditional Diffie-Hellman key-exchange protocol, there is a well-known attack --- MITM (Man-In-The-Middle Attack)[4]. And there have already emerged many distinguishing security attacks to the group key based on DH protocol[5].

In fact, the traditional MITM attack can hardly validate to this method of group key generation. Seen from the micro level, the managing node uses traditional DH protocol with one non-managing node to negotiate the group key, however, considered at the macro level, the managing node communicates with a number of other nodes. Even if the MITM succeeds in one communicating path, it will be detected immediately after the negotiation completes, because the node being attacked will find its key different from other nodes' key since it cannot decrypt the packages received from other nodes correctly.

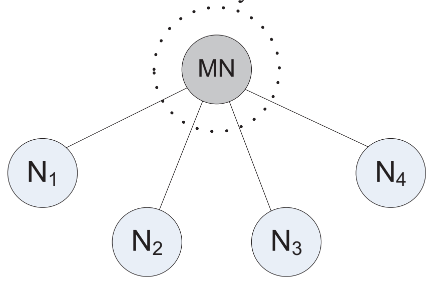

Figure 6. An improved MITM attack

One improved MITM can hijack all the communicating paths between the managing node and other non-managing nodes, which works by hijacking the managing node. In Fig. 6, MN is the managing node, the dotted line stands for that the attacker has successfully hijacked all the session between the managing and non-managing nodes.

In fact, we can use digital signature technology to defeat this attack. Adding digital signature based authentication technology to the DH based key exchange protocol also has been quite common and sophisticated.

VI. Conclusions

This quick group key generating method is raised in order to meet paroxysmal need of group key for multi-node in the Internet. It can effectively reduce overlapping computing, decrease the packages sending times, and cut the time complexity from $O(N\log_2 N)$ of traditional binary-tree method down to O(N). Meanwhile, the non-managing nodes still maintain basic binary-tree structure, which enhances the speed of dynamically adding and deleting nodes. Additionally, this paper briefly discusses about the famous MITM attack, and come up with an effective solution to this attack.

References

[1] LIU Xiao-hu, GU Nai-jie, LU Yu-liang, BI Kun, REN Kai-xin, Optimal Key Tree Structure in Secure Multicast [J], Mini-Micro Systems, 2006, 12 (in Chinese)

[2] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Transaction on Information Theory 22 (6) (1976) 644–654.

[3] Goshi J, Ladner R E. Algorithms for Dynamic Multicast Key Distribution Trees .10thACM Conference on Computer and Communications Security[C]. Boston,USA. 2003, :240-246 .

[4] XU Heng, CHEN Gong-liang, YANG Fu-xiang, Prevention of Man-In-The-Middle Attack in Diffie-Hellman Key Exchange under Specific Environment [J], Information Security and Communications Privacy, 2009, 02:036

[5] Igor E. Shparlinski. Security of polynomial transformations of the Diffie–Hellman key [J]. Finite Fields and Their Applications, 2004, 10: 123–131