

Optimally-balanced Hash Tree Generation in Ad Hoc Networks

V. R. Ghorpade¹, Y. V. Joshi² and R. R. Manthalkar²

1. Kolhapur Institute of Technology, Kolhapur, India, vijayghorpade@hotmail.com

2. SGGSI&T, Nanded, India

Abstract – Ideally a hash tree is a perfect binary tree with leaves equal to power of two. Each leaf node in this type of tree can represent a mobile node in an ad hoc network. Each leaf in the tree contains hash value of mobile node's identification (ID) and public key (PK). Such a tree can be used for authenticating PK in ad hoc networks. Most of the previous works based on hash tree assumed perfect hash tree structures, which can be used efficiently only in networks with a specific number of mobile nodes. Practically the number of mobile nodes may not be always equal to a power of two and the conventional algorithms may result in an inefficient tree structure. In this paper the issue of generating a hash tree is addressed by proposing an algorithm to generate an optimally-balanced structure for a complete hash tree. It is demonstrated through both the mathematical analysis and simulation that such a tree is optimally-balanced and can efficiently be used for public key authentication in ad hoc networks.

Index Terms – Hash Tree, Ad Hoc Network, Authentication

I. INTRODUCTION

In recent years, ad hoc networks have received more attention, because of their easy deployment. However the characteristics of ad hoc networks are more prone to physical security threats than the wired network environment. It includes infrastructure-less architecture, dynamic network topology, energy constrained devices and shared wireless channel. Therefore it has become a primary concern of securing an ad hoc network. Authentication is one of the major topics in network security. For authentication, certificate based approaches are popular in classical networks. Certificate based public key authentication schemes are complicated and requires more computational power and energy. In a resource constrained ad hoc network public key authentication can be done using a hash tree [1] which requires less computational power and energy. A complete binary tree also finds

applications in various group key generation protocols.

A hash tree is perfect binary tree having n leaf nodes with the height of the tree as l , where l is given by $l = \lceil \log_2 n \rceil$. Each leaf in the tree contains hash value of mobile node's ID and PK. The value of each internal node of the tree is a hash value of the concatenated values of its two child nodes. Such a tree can be used for authenticating PK in ad hoc networks.

Most of the previous works that employed the hash trees for public key authentication scheme [2,3,4], assumed perfect trees with 2^{l-1} leaf nodes, to represent a node from an ad hoc network and did not investigate the tree structure itself. While this assumption may hold in some special cases with 2^{l-1} mobile nodes, in other cases when the number of mobile nodes varies dynamically, the conventional algorithms may result in an inefficient tree structure. To solve this problem

an algorithm is presented in [5] for optimal tree construction for any network size. The scheme presented in [5] is analyzed in this paper and a simple algorithm for generating optimally-balanced complete hash tree is proposed for any number of leaf nodes (i.e. $n > 0$).

The paper is organized as follows. Section II, presents an analysis of the scheme presented in [5]. In section III, an algorithm to build an optimally-balanced hash tree is proposed. Section IV presents mathematical proofs followed by performance analysis in section V; and section VI concludes this paper.

II. PREVIOUS WORK

Veronika and Seo [5] in their proposal have presented an algorithm to generate an optimal hash tree structure which can be used in sensor networks for PK authentication; where the number of sensors are not of the order of power of two. If hash trees are used for PK authentication then it is important to minimize the number of nodes with the maximum authentication path (AP), which is a set of siblings by which a root can be reconstructed from the leaf values; and the tree must be as balanced as possible. In [5] a parameter P_0 is presented to measure the degree of optimality by the following probability:

$$P_0 = 1 - \frac{n_0}{n} \quad (1)$$

where P_0 is the probability that a node has AP length less than maximum, and n_0 is the number of nodes in a group with the maximum AP length. It is claimed that the algorithm presented in [5] constructs a tree with the maximum possible probability P_0 for

given n , where n is network size. To justify their claim they have given an example of a network with $n = 21$ and showed that the value of P_0 is maximum in this case. That is, minimum number of nodes have maximum AP length and the value of P_0 becomes 52%.

While analyzing the scheme it is observed that, when the same algorithm is applied in a group with $n = 17$ and $n = 31$, then the value of the parameter P_0 becomes 88% and 3.2% respectively. From such and similar extreme cases it is clear that with $n = 2^{l-1} + 1$ the parameter P_0 has maximum value and hence the tree is optimized. On the other hand with $n = 2^l - 1$ the value of P_0 is minimum and therefore the tree is not optimized.

In [5] it is concluded that, the number of nodes with longest AP is made minimal. These results could not be confirmed in the proposed paper with an appropriate experimental setting. The parameter P_0 presented in [5] does not give any correct information about the optimality of the tree. Also the complexity of the tree generation algorithm presented in [5] is more due to following two reasons: First, the algorithm demands that the tree must be represented in the suggested polynomial form and secondly, it creates sub-trees and then concatenates these sub-trees to form the resulting tree.

III. PROPOSED SCHEME

While using binary trees for security related applications, such as

PK authentication and group key generation, in ad hoc network; it is important to generate the binary tree as balanced as possible. A complete binary tree is said to be optimally balanced if and only if the leaves are distributed at level l and $l-1$ only; where l is the height of the complete binary tree and all the leaf nodes on the last level must occupy the leftmost spots consecutively. In this paper a simple algorithm is proposed to generate an optimally-balanced complete hash tree for any group size (i.e. $n > 0$). The proposed algorithm is easy to implement and does not demand to represent the tree in the polynomial form. It generates a single tree from the root node and there is no need of forming sub-trees and then concatenating them to obtain the resulting tree. The proposed algorithm is shown in Figure 1.

Algorithm

Given: number of leaf nodes (n), height of the complete binary tree ($l = \lceil \log 2n \rceil$) and l satisfies $2^{l-1} < n < 2^l$, node index (v)

Output: Tree(T) with n leaf nodes

1. if $T = NULL$ then create root node
2. set node counter $k = 1$; set level $d = 1$;
3. perform step 3 through 12 while($d < l$)
4. take the leftmost node at level d and make it as current node. set $v = 0$;
5. perform steps 5 through 10 while($v < 2^{d-1}$)
6. create left and right child nodes to the current node v

7. increment node counter k by 2
8. check whether nodes created (k) exceeds the total nodes in the tree
if($k \geq (2n - 1)$) stop
9. move to the next node, if present, at the same level d ; make that node as current node;
10. increment v and go to step 5
11. increment d
12. go to step 3
13. end

Figure 1: Optimally-balanced complete hash tree generation algorithm

It is proved in Section IV that the tree generated with the proposed algorithm is optimally-balanced in the sense that the leaf nodes of the tree are either at level l or $l-1$ and the deepest nodes are on the extreme left hand side of the tree.

IV. MATHEMATICAL ANALYSIS

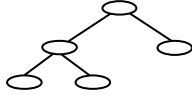
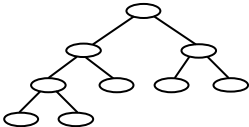
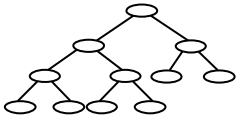
Theorem: The hash tree built by above algorithm is optimally balanced and complete.

Let n be number of leaf nodes in a given complete binary tree. Prove that iff n is not of the order of power of two; then the leaf nodes exist on last and second last level in the tree.

Assume, $n = 2^p + r$ where $p \in \mathbb{N}$ (\mathbb{N} is a set of natural numbers) and

$\left\lfloor \frac{n}{2^p} \right\rfloor = 1$. Then it is to prove that $2r$ leaves exists on level l and $n - 2r$ leaves exist on level $l-1$, where l is the height of the complete binary tree given by $l = \lceil \log 2n \rceil$.

Intuition:

Tree (T)	Leaves (n)	Height (l)	n_l	n_{l-1}
	3	3	2	1
	5	4	2	3
	6	4	4	2
n_l : # leaves on l^{th} level		n_{l-1} : # leaves on $l-1^{\text{th}}$ level		

Observation:

For every increment in leaf node count, the number of leaf nodes on last level increases by 2 and the number of leaf nodes on second-last level decreases by 1.

Proof:

Induction is useful to prove the theorem with trees.

Basis:

Let $n = 3$: $p = 1, r = 1,$
 $l = 3$, therefore;
 $n_l = 2r = 2$ leaves exist
 on 3rd level and
 $n_{l-1} = n - 2r$ leaves exist
 on 2nd level, and
 $n = n_l + n_{l-1}$

Inductive Hypothesis:

Assume that the theorem is true for leaf nodes $n \geq k$ where $k > 3$.

Inductive step:

Let us prove that the inductive hypothesis is true for leaf nodes $n = k + 1$

Let $n = 2^p + r_1$
 where $r_1 = r + 1$
 $n = 2^p + (r + 1)$

From the basis and inductive hypothesis;

$n_l = 2r_1 = 2r + 2$ leaves exist on l^{th} level and,

$n_{l-1} = n - 2r_1 = n - 2(r + 1)$
 $n_{l-1} = n - 2r - 2$ leaves exist on $l - 1^{\text{th}}$ level
 but, $n = k + 1$ therefore,
 $n_{l-1} = k + 1 - 2r - 2$
 $n_{l-1} = (k - 2r) - 1$

leaves exist on $l - 1^{\text{th}}$ level
 Thus, the inductive hypothesis is true for $n = k + 1$ leaf nodes and, hence

(by induction), true for all leaf node counts. Hence it is proved that the tree built by the proposed algorithm is optimally-balanced and complete.

V. PERFORMANCE ANALYSIS

Simulation setup:

The proposed algorithm is implemented using C++ to build a hash tree and used along with the network simulator (ns-2) [6] simulating a Mobile Ad hoc Network (MANET) with different network sizes. Standard cryptographic function SHA1 from OpenSSL Library [7] is used for computation of hash values which generates a 160 bit hash. The tree thus built is used for public key authentication. The leaf nodes of the tree are attached to the respective mobile nodes from the simulated MANET.

Performance Evaluation:

This section provides a detailed quantitative analysis of the proposed algorithm. The performance of the proposed algorithm is evaluated using communication overhead and memory requirement per node for public key authentication. The algorithm builds a binary tree which is optimally-balanced for any number of leaf nodes. These balanced trees tend to be 'bushy': they have few levels and spread out width-wise. Trees that are wide and shallow have only a few levels, precisely $\lceil \log 2n \rceil$; recall that n is the number of

leaf nodes. The experiments are conducted for network sizes ranging from 8 to 2048 nodes. The proposed scheme is compared with a basic scheme in which the number of nodes are exactly of the order of power of two.

A. Communication overhead vs. network size

Experimental work shows that, the communication overhead in MANET depends upon the distance and number of hops between two communicating peers, the traffic scenario, quality and bandwidth of the channel etc. In the proposed scheme, in addition to the above factors; the communication overhead is proportional to the height of the tree, while the height of the tree is decided by the number of mobile nodes in the network. Therefore, the size of a network will affect the performance of the scheme. Figure 2 depicts how the network size affects the communication overhead.

B. Memory requirement vs. network size

Each node holds hash value of the root node and its AP. Since a balanced tree is rather 'bushy', the number of nodes in the AP are minimal consuming less memory per node. Memory requirement per node changes with the increase in network size. Figure 3 clearly shows the trend of memory requirement with the increase of network size.

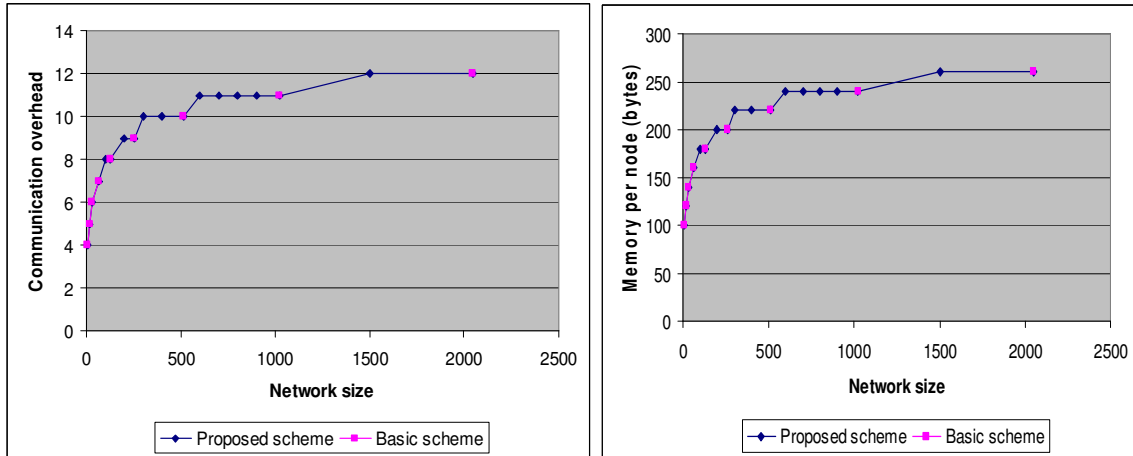


Figure 2 : Communication overhead vs. network size Figure 3: Memory requirement vs. network size

VI. CONCLUSIONS

A simple algorithm for generating an optimally-balanced complete hash tree is proposed in this paper. Given any number of leaf nodes, the algorithm builds a binary tree. Through mathematical analysis it is shown that the leaf nodes of the generated tree exists only on last and second-last level. Hence, it is concluded that such a tree is optimally-balanced complete hash tree. The complexity and memory requirement of the algorithm is $O(n)$ and it does not demand to represent the nodes in polynomial form to construct the tree. Experimental results show that, when the proposed algorithm is used for public key authentication in MANET; the AP and per node memory requirement remains optimal even though the number of leaf nodes are not of the order of power of two.

REFERENCES

- [1] R. C. Merkle, "Protocols for Public Key Cryptosystems," in *Proc. IEEE Symposium on Research in Security and Privacy, 1980*.
- [2] W. Du, R. Wang, and P. Ning, "An Efficient Scheme for Authenticating Public Keys in Sensor Networks," *ACM, MobiHoc, 2005*.
- [3] L. Zhou and C. V. Ravishankar, "Dynamic Merkle Trees for verifying privileges in Sensor Networks," in *Proc. IEEE ICC 2006*.
- [4] M. Jakobsson and S. Wetzel, "Efficient Attribute Authentication with Applications to Ad hoc Networks," *ACM, VANET, 2004*.
- [5] V. Kondratieva and S-W Seo, "Optimized Hash Tree for Authentication in Sensor Networks," *IEEE Communications Letters, Vol. 11, No. 2, Feb. 2007*.
- [6] ns-2 : Network Simulator, available at <http://www.isi.edu/nsnam>
- [7] OpenSSL : Standard Cryptographic Library, available at <http://www.openssl.org>