Membership Proof and Verification in Authenticated Skip Lists Based on Heap

Shuanghe Peng * , Zhige Chen, Deen Chen

School of Computer and Information Technology, Beijing Jiaotong University, 100044, Beijing, China

Abstract: How to keep cloud data intact and available to users is a problem to be solved. Authenticated skip list is an important data structure used in cloud data integrity verification. How to get the membership proof of the element in authenticated skip list efficiently is an important part of authentication. Kaouthar Blibech and Alban Gabillon proposed a head proof and a tail proof algorithms for the membership proof of elements in the authenticated skip list. However, the proposed algorithms are uncorrelated each other and need plateau function. We propose a new algorithm for computing the membership proof for elements in the authenticated skip list by using two stacks, one is for storing traversal chain of leaf node, the other is for storing authentication path for the leaf. The proposed algorithm is simple and effective without needing plateau function. It can also be applicable for other similar binary hash trees.

Keywords: authenticated skip list; max heap; membership proof; stack; algorithms

I. INTRODUCTION

Cloud storage as a service provides convenience for business users because of its economy and flexibility. The cloud is a multi-tenant environment, where resources are shared. Placing data in the hands of a vendor intuitively seems risky. The problem is that the cloud server may be malicious, and even if the server is trustworthy, hardware/software failures may cause data corruption. The clients should be able to verify the integrity of their data efficiently and securely without downloading the entire data from the server. How to keep cloud data intact and available to users is a problem to be solved.

PDP (Provable Data Possession) was such a model that can be used to solve the problem mentioned above [1]. In a PDP model, the clients can challenge the server on random blocks and verify the data integrity through a proof sent by the server. The key point in integrity verification process is how to get the proof information. In some previous algorithms, hash chain[2],[3], Merkle hash tree [4] and authenticated skip list [5] were employed as data structure in PDP to get the proof information.

Here we focus on authenticated skip lists. Authenticated skip lists were presented in 2001 by Goodrich and Tamassia [5], where skip lists [6] and commutative hashing are employed in the data structure for authenticated dictionaries.

Skip list was first proposed in 1989[6]. It is an interesting probabilistic data structure that

consists of a set of ordered lists. Skip Lists were first used for their simple implementation and optimal update and search time.

Authenticated skip list has advantages in dynamic operations, such as insertion and deletion, and it is widely used in certification revocation list [9], digital time stamping systems and outsourced data storages [10].

Kaouthar Blibech and Alban Gabillon defined membership proof for element e as information needed to compute the label of the ending node E_t from the value of element e [12]. The membership proof for element e contains a head proof and a tail proof, where head proof is the minimal set of nodes that summarizes the state of the half-constructed skip list for that moment and tail proof contains all the extra nodes that are needed to re-compute the summary hash from the request.

Kaouthar Blibech and Alban Gabillon gave two algorithms[12] to compute head proof and tail proof respectively. The computation of head proof needs know the insert order of nodes in the authenticated skip list and plateau function *hasPlateauOnLeft*(*index*, *level*) that indicates if the node of index *index* and level *level* has a plateau node on its left. While the computation of tail proof re-visited the nodes on the traversal chain from bottom to top. Further more, the head proof and tail proof can't be used in an isolation way, they have to be merged to create a whole to prove integrity of an element.

In this paper, we first map authenticated skip list to max heap and propose an algorithm for the computation of membership proof for leaf nodes in the max heap by using two stacks, one is for storing traversal chain of leaf node, the other is for storing authentication path for the leaf. The proposed algorithm can solve the problems mentioned above and can also be applicable for other similar binary hash trees. Here we call membership proof as authentication path in binary hash trees.

II. Preliminaries and Notations

Definition 1: [*node*. *flag*] A flag stores the current status of the visited node of the tree in pre-order traversal.

$$node.flag = \begin{cases} 0 & node\text{'s children have} \\ & \text{not been visited yet,} \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

Definition 2: [*Authentication Path*] We will make use of the notion of the *AuPath* as authentication path for a given node in a hash tree. For a node n, $AuPath(n)$ is the set of siblings of the nodes on the path from n to the root. More formally, if we let $Sib(n)$ and $Parent(n)$ denote n's sibling and parent respectively, then:

$$AuPath(n) = \begin{cases} \varphi & \text{if } n \text{ is the root,} \\ Sib(n) \cup AuPath(Parent(n)) & \\ \text{otherwise} & \end{cases} \quad (2)$$

Definition 3: [*stackTraversal*] A stack stores the current visited node in pre-order traversal of the tree.

Definition 4: [*stackAuth*] A stack stores the authentication path nodes for the current visited node of the tree.

Definition 5: [*plateau*] Skip list is viewed as a two dimensional collection of positions arranged horizontally into levels and vertically into towers. The highest element on tower is called plateau. For example, plateau nodes are drawn with thick line in Fig.1.

Definition 6: [*node. level*] A number stores the current level of the visited node of the tree

In this paper, the author proposed a new algorithm for the computation of membership proof in the authenticated skip lists.

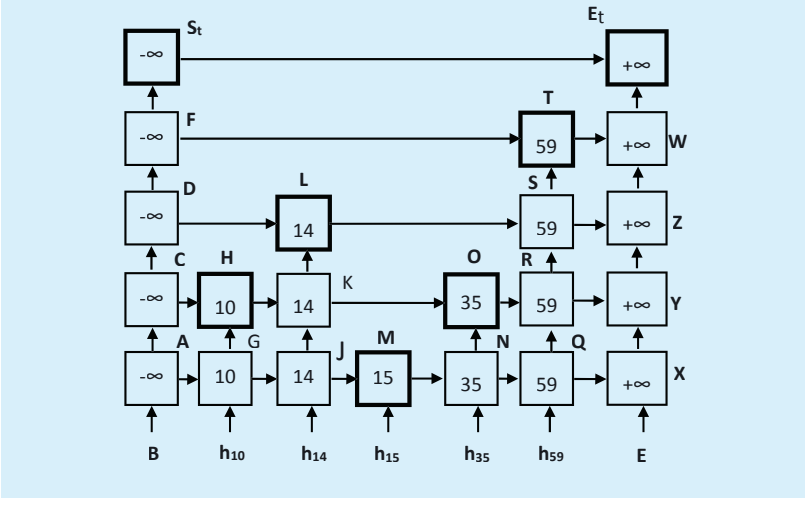

Fig.1 *Skip list and plateau nodes. Plateau nodes are drawn in thick line*

$$node.level = \begin{cases} 0 & \text{if } node \text{ is the root,} \\ Parent(node).level + 1 & \\ \text{otherwise} & \end{cases} \quad (3)$$

Definition 7: [$node.linkflag$] A flag stores the valid status of the right (or after) link of the node in the process of building Authenticated Skip List from original Skip List.

$$node.linkflag = \begin{cases} 0 & node\text{'s right link is invalid} \\ & \text{in authenticated skip list,} \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

We denote left child of node x by $lchild(x)$, right child of node x by $rchild(x)$, value of node x in the authenticated skip list by $value(x)$ and label of node x in the authenticated skip list by $label(x)$. Other notations are shown in Table I.

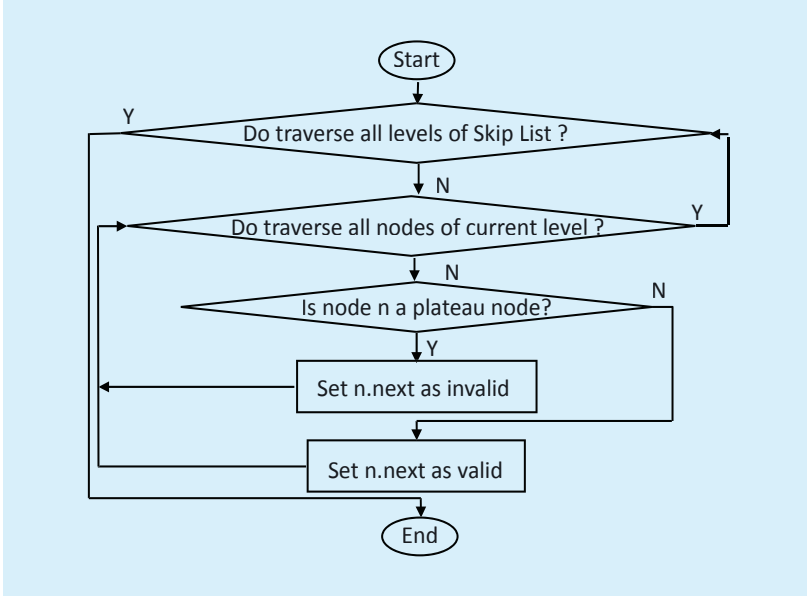

Fig.2 *Flow of deletion of unnecessary links in the Skip List*

Table I *Notations*

Function	Description
stack. init	initialize an empty stack
stack. empty()	judge if the stack is empty
stack. push()	push an element into the stack
stack. Pop()	pop an element from the stack
stack. getTop()	get the top element of the stack
stack. output()	output all the elements of the stack
movTop ($stack_1$,$stack_2$)	move top element of stack $stack_2$ to the top of stack $stack_1$
exchTop ($stack_1$,$stack_2$)	switch the top element of two stacks
clearFlag(x)	clear the flag field of node x
clearLevel(x)	set the level field of node x to 0

III. Max Heap Built From Authenticated Skip List

3.1 From skip list to authenticated skip list

Let || denote the concatenation operation, and $h()$ denote the hash function.

In authenticated skip list each node stores a hash value calculated with the use of its own fields and the hash values of its neighboring nodes. The hash value of the root is the authentication information (meta data) that the client stores in order to verify responses from the server. In order to build authenticated skip list from skip list, a label field is added to every node of skip list to store integrity value. The label field is computed as follows:

1) For left most node n in the authenticated skip list, $label(n)$:=0;

2) For nodes with *level* equal to 0, there are two cases:

a) If node n's left node is not a plateau $label(n) = h(value(n))$;

b) If node n's left node is a plateau, $label(n) = h(value(n)||label(left(n)))$;

3) For nodes with *level* not equal to 0, there are also two cases:

a) If node n's left node is not a plateau $label(n) = label(down(n))$;

b) If node n's left node is a plateau, $label(n) = h(label(down(n))||label(left(n)))$;

In the process of building authenticated skip list from original skip list, some links are useless for searching node. The unnecessary links should be marked as invalid. Formally:

- A link is necessary if and only if it is on any search path.
- A node is necessary if and only if it is at the leaf level or has a necessary after link.

According to the rule of the label computation of node n in the authenticated skip list, the deletion algorithm 1 of unnecessary link in the original skip list is shown in Fig.2.

The authenticated skip list built from skip list in Fig.1 is shown in Fig.3. In the horizontal direction, all links are marked invalid ex-