云存储中基于SBT的数据完整性验证机制

钟 婷，耿纪昭，熊 虎，秦志光

(电子科技大学计算机科学与工程学院 成都 611731)

【摘要】云存储中数据完整性验证的主要思想是利用树形或类树形数据结构来存储用户外包的数据，并将该数据结构与适当的密码学技术结合使其具备可认证性，即认证字典。该文将基于结点大小平衡树(size balanced tree, SBT)结构来构建认证字典，并在该认证字典基础上设计出数据完整性验证方案。基于SBT结构的数据完整性验证方案能更好地支持动态数据集的完整性验证，避免了数据更新引起的周期性重构。实验结果表明，相对于基于AVL树、Treap的认证结构，基于SBT结构的认证字典具备更好的平衡性和更高的数据操作效率。

关 键 词 云存储； 数据安全； 完整性验证； 结点大小平衡树

中图分类号 TN918 **文献标志码** A doi:10.3969/j.issn.1001-0548.2014.06.023

A Data Integrity Verification Mechanism Based on SBT in Cloud Storage

ZHONG Ting, GENG Ji-zhao, XIONG Hu, and QIN Zhi-guang

(School of Computer and Engineering, University of Electronic Science and Technology of China Chengdu 611731)

Abstract Data integrity verification in cloud storage mainly uses tree or quasi-tree data structure to store outsourced data, and combines such data structure with appropriate cryptographic techniques to make it authenticated, which is named as authenticated dictionary. This paper designs a new authenticated dictionary based on the size balanced tree and proposes a new data integrity verification scheme based on the new authenticated dictionary. The new scheme supports integrity verification for dynamic data set and avoids periodic reconstruction of authenticated dictionary which can be caused by update operations. Experimental results show that the authenticated dictionary based on SBT has better balance and higher efficiency of data operation than the ones based on AVL tree and Treap.

Key words cloud storage; data security; integrity verification; size balanced tree

云存储中的数据完整性验证机制是以三方模型[1]为基础的。所谓三方，是指云存储用户、不可信服务器和可信数据源。在三方模型中，用户将数据外包到外部服务器上，包括一个可信数据源和很多不可信服务器。其中，可信数据源是完整而准确地保存着用户的数据，它将数据再发布到各个不可信服务器。用户需要使用外包数据时，只需访问其中一个或几个不可信服务器即可。由于用户取回数据时所访问的服务器是不可信的，这时就需要数据完整性验证机制来确保用户所取回的数据是未被篡改和伪造的原始数据。

数据完整性验证机制的核心思想是：采用成员问询策略，该策略最初被提出的动机是为了解决公钥体制中的证书撤销问题[2-5]。这种策略将存储数据的哈希表与一个具有可认证性的数据结构相结合，即认证字典，使得成员问询的回答具有可验证性，借此用户可验证所访问的外包数据是否完整。

成员问询可抽象为如下数学过程：将外包的数据抽象成一个数据集 $S=(e_1,e_2,\cdots,e_n)$，其中 e_i 表示一个大小固定的数据块。验证某一数据块是否完整的过程可抽象地看作为证明元素 x 是否属于数据集 S 的过程，其中 x 代表验证的数据块。具体实现过程步骤如下：

1) 用户给不可信服务器发送一个问询请求，询问元素 x 是否在数据集 S 中；

2) 不可信服务器在本地执行相应的查询操作，将得出的答案Yes或No返回给用户，同时还要返回查询过程中生成的“证据”；

收稿日期：2013 – 09 – 18；修回日期：2014 – 03 – 20
基金项目：青年科学基金(61103205)；中国移动科研基金(2012 MCM20121041)；广东省产学研重点项目(2012B091000054)
作者简介：钟婷(1977 –)，女，博士，副教授，主要从事云存储安全及无线通信安全方面的研究.

3) 用户根据不可信服务器返回的“证据”，进行一系列计算，重新得出数据的特征值；

4) 用户将上一步骤中得到的特征值与本地的数据特征值(由可信数据中心事先计算得出并传递给用户，用户将其预存在本地)进行对比，两者相同则用户接受不可信服务器返回来的答案，反之，则拒绝。

对于云存储的完整性验证，已有Merkle哈希树、跳表及RSA树等解决方案。Merkle哈希树是一种用以建立认证字典的基础数据结构[6-7]。以Merkle树结构存储数据集 S，将 S 中的元素都存储在哈希树 T 的叶子中，并在树的每一个结点上存储一个标签值。若结点为叶子结点，其标签值与其存储的数据相同；否则由其孩子结点的标签值通过抗碰撞哈希函数计算得出。用户保存数据集 S 的摘要值，即Merkle树根结点的摘要值。当用户向服务器提出问询某一数据块时，服务器返回该数据块及一系列的证据信息，用户根据这些证据信息重新计算数据集 S 的摘要值，并将其与之前所存储的摘要值相对比，若相同则认为数据未被破坏。

基于Merkle哈希树的认证字典设计中采用了简单而高效的哈希函数，其数据操作的时间复杂度为 $O(\log n)$，且它不支持数据的插入操作。

文献[8-9]提出采用跳表结构和可交换哈希函数实现认证字典方案。跳表是一种随机性的数据结构，最低层采用链表存储所有数据，上一层链表中的元素是下一层的概率性子集。跳表结构可看成哈希树结构的一种变形，其数据操作的时间复杂度也为 $O(\log n)$。文献[10]提出了一种基于Universal-hash的MAC树，由于采用了计算效率较高的Universal-hash，此方案效率高于普通的Merkle树方案。文献[11]将树型结构与RSA单向累加器技术相结合构造出RSA树，通过引入参数 ε 使得树高和树的分支数都为常量，所以它的证据大小、查找和验证的时间复杂度为 $O(l)$。但它采用的单向累加器技术需要应用复杂的幂模运算，所以其实际运行时间并不乐观，且RSA树在数据插入时并不灵活，可能会引起复杂的结构重构。

基于Merkle哈希树、跳表及RSA树的认证字典对动态数据集的支持都较差。一些改进的方案，如2-3叉树或Bucket[11]，也能支持数据的更新操作，但都难以从根本上避免更新操作带来的周期性重构问题。

1 新的完整性验证方案的设计

本文通过引入SBT数据结构并将其可认证化设计一种新的认证字典，在此基础上提出新的完整性验证方案。在基于SBT的认证字典中，数据不仅仅能存储在其叶子结点，树中的所有结点都可存储。

1.1 平衡二叉搜索树——SBT

SBT是一种二叉搜索树，具备以下性质：

1) 若左子树不空，则左子树上所有结点的值均小于根结点的值；

2) 若右子树不空，则右子树上所有结点的值均大于根结点的值；

3) 左、右子树也分别为二叉搜索树。

SBT是一种通过子树大小(即树中结点个数)来维持平衡的二叉搜索树。这种设计结构简单而巧妙，可高效的动态维护二叉搜索树，即使在最坏的情况下也可保持良好的运行速度，当树中的结点出现不平衡时，需要进行旋转操作来保持平衡。SBT结构示意图如图1所示。

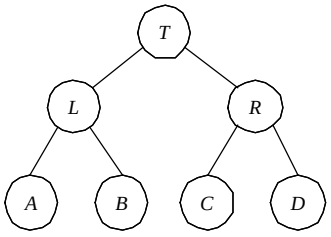

图1 SBT结构示意图

对于每个SBT中的结点 t，须满足性质：

$$s[\text{right}[t]] \geqslant s[\text{left}[\text{left}[t]]], s[\text{right}[\text{left}[t]]] \quad (1)$$

$$s[\text{left}[t]] \geqslant s[\text{right}[\text{right}[t]]], s[\text{left}[\text{right}[t]]] \quad (2)$$

式中，$\text{left}[t]$ 表示 t 的左孩子结点；$\text{right}[t]$ 表示 t 的右孩子结点；$s[t]$ 表示以 t 为根的子树的大小，即这棵树上结点的个数。

1.2 SBT的可认证化

以SBT为基础建立的全结点存储树 T，其中每一个结点 v 都须存储3项内容：

1) 存储在该结点的数据 D_v；

2) 该结点数据的摘要值 $H_v = h(D_v)$；

3) 联合摘要值 $H_{\text{sum}}(v)$。

联合摘要值 $H_{\text{sum}}(v)$ 的计算方法如下：

1) 若节点 v 是叶子节点，则联合摘要值 $H_{\text{sum}}(v) = H_v$；

2) 若节点 v 不是叶子节点，且只有左孩子节点，则 $H_{\text{sum}}(v) = h(H_v, H_{\text{sum}}(\text{left}[v]))$，其中 $H_{\text{sum}}(\text{left}[v])$ 代表节点 v 的左孩子节点 $\text{left}[v]$ 的联合摘要值；

3) 若节点 v 不是叶子节点，且只有右孩子节点，则 $H_{\text{sum}}(v)=h(H_v,H_{\text{sum}}(\text{right}[v]))$，其中 $H_{\text{sum}}(\text{right}[v])$ 代表节点 v 的右孩子节点 right[v] 的联合摘要值；

4) 若节点 v 不是叶子节点，且兼有左右孩子节点，则联合摘要值 $H_{\text{sum}}(v)=h(H_v,H_{\text{sum}}(\text{left}[v])$，$H_{\text{sum}}(\text{right}[v]))$。

整个外包数据的摘要值 Digest(T) 即是全节点存储树 T 根节点的联合摘要值。

1.3 数据完整性验证过程

首先用户将数据存储到可信的第三方服务器，可信服务器计算出数据集的摘要值 Digest(T)，并将该值提供给用户存储在本地。之后可信服务器将数据和认证结构交由云存储服务提供商存储，即三方模型中的不可信服务器。当用户发送一个成员问询到不可信服务器时，假设不可信服务器在全结点存储树中查找到存储着目标数据的相应目标结点 target，target 到根结点 head 的查找路径 $\text{Path}=\{v_0(\text{target})\to v_1\to,\cdots,\to v_{l-1}\to v_l(\text{head})\}$。此时，不可信服务器向用户回答Yes并返回相应证据 $\text{Proof}=\{\eta,\pi_1,\pi_2,\cdots,\pi_l\}$，其中 η 定义为：

1) 若目标节点 v_0 是叶子节点，则 $\eta=\{D_{v_0}\}$；

2) 若目标节点 v_0 不是叶子节点，则：

① 若目标节点 v_0 只有左孩子节点，则 $\eta=\{D_{v_0},H_{\text{sum}}(\text{left}[v_0])\}$，其中 $H_{\text{sum}}(\text{left}[v_0])$ 代表目标节点 v_0 的左孩子节点 left[v_0] 的联合摘要值；

② 若目标节点 v_0 只有右孩子节点，则 $\eta=\{D_{v_0},H_{\text{sum}}(\text{right}[v_0])\}$，其中 right[$v_0$] 代表目标节点 v_0 的右孩子节点，$H_{\text{sum}}(\text{right}[v_0])$ 代表目标节点 v_0 的右孩子节点 right[v_0] 的联合摘要值；

③ 若目标节点 v_0 兼有左右孩子节点，则 $\eta=\{D_{v_0},H_{\text{sum}}(\text{left}[v_0]),H_{\text{sum}}(\text{right}[v_0])\}$。

对于 $1\leqslant i\leqslant l$，π_i 的定义如下：

1) 若节点 v_{i-1} 为左孩子节点，则：$\pi_i=(H_{v_i},H_{\text{sum}}(v_{i-1}),H_{\text{sum}}(\text{right}[v_i]))$，其中节点 v_{i-1} 为节点 v_i 的左孩子节点，与节点 v_i 的右孩子节点 right[v_i] 为兄弟节点；

2) 若节点 v_{i-1} 为右孩子节点，则：$\pi_i=(H_{v_i},H_{\text{sum}}(\text{left}[v_i]),H_{\text{sum}}(v_{i-1}))$，其中节点 v_{i-1} 为节点 v_i 的右孩子节点，与节点 v_i 的左孩子节点 left[v_i] 为兄弟节点。

当用户收到不可信服务器返回的证据Proof后，将进行如下验证；

1) 计算 $a=h(D_{v_0})$，其中 D_{v_0} 在证据信息Proof的元素 η 中。

若节点 v_0 是叶子节点，则验证 $a=H_{\text{sum}}(v_0)$ 是否成立，$H_{\text{sum}}(v_0)$ 在证据信息 Proof 的元素 π_1 中。

若节点 v_0 不是叶子节点：

① 若节点 v_0 只有左孩子节点，则验证 $h(a,H_{\text{sum}}(\text{left}[v_0]))=H_{\text{sum}}(v_0)$ 是否成立；

② 若节点 v_0 只有右孩子节点，则验证 $h(a,H_{\text{sum}}(\text{right}[v_0]))=H_{\text{sum}}(v_0)$ 是否成立；

③ 若节点 v_0 兼有左右孩子节点，则验证 $h(a,H_{\text{sum}}(\text{left}[v_0]),H_{\text{sum}}(\text{right}[v_0]))=H_{\text{sum}}(v_0)$ 是否成立；

2) 对于 $\pi_i(i<l)$，若 v_{i-1} 为左孩子结点则计算 $h(H_{v_i},H_{\text{sum}}(v_{i-1}),H_{\text{sum}}(\text{right}[v_i]))$，若为右结点则计算 $h(H_{v_i},H_{\text{sum}}(\text{left}[v_i]),H_{\text{sum}}(v_{i-1}))$，并验证该值与联合摘要值 $H_{\text{sum}}(v_i)$ 是否相等，其中 $H_{\text{sum}}(v_i)$ 在 π_{i+1} 中；

3) 对于 π_l，若 v_{l-1} 为左孩子结点，则计算 $h(H_{v_l},H_{\text{sum}}(v_{l-1}),H_{\text{sum}}(\text{right}[v_l]))$，若为右结点则计算 $h(H_{v_l},H_{\text{sum}}(\text{left}[v_l]),H_{\text{sum}}(v_{l-1}))$，并将结果与用户之前所存储的数据集的摘要值 Digest(T) 相对比。

若以上所有验证过程全部成立，则证明不可信服务器所返回的答案是正确的，即所问询的数据是完整的。反之则认为该数据已被篡改或伪造，这时需在不可信服务器上采用数据恢复等技术恢复用户原始数据。

1.4 SBT上的数据操作

在云存储的现实应用环境中，存储在服务器端的数据需要经常进行查找、插入、更新和删除操作。其中，查找是最基础的操作，无论是插入、更新还是删除，首先都需要进行查找操作以定位到目标结点。结点大小平衡树的查找过程如下：

1) 若树是空树，则搜索失败；

2) 若目标值等于树的根结点的数值，则查找成功；

3) 若目标值小于树的根结点的数值，则递归的查找左子树；

4) 递归的查找右子树。

由于结点大小平衡树是以链表的存储结构实现的，在插入和删除的操作中，并不需要移动结点的位置，只需改变结点的指针即可。所以其时间复杂度跟查找操作的时间复杂度保持一致。

在基于 SBT 的认证数据结构中插入一个数据元素时，首先通过查找找到插入结点的位置，然后执行插入。由于插入的结点必然处于 SBT 的叶子结点

位置，若插入操作没有引起旋转调整操作，那么只需重新计算 SBT 的根结点到新的叶子结点的路径上的所有结点的摘要值；若插入操作引起了旋转调整操作，那么需在旋转调整后对所有影响到的结点(包括进行旋转的子树中的所有结点，以及从该子树的根结点到 SBT 的根结点的路径上的所有结点)摘要值进行重新计算。

在基于 SBT 的认证数据结构中删除一个数据元素时，分 3 种情况讨论：

1) 欲删除的结点 x 是叶子结点，则直接删除即可，需要修改信息的结点是从根结点到 x 的父结点的路径上的所有结点；

2) 欲删除的结点 x 是单支结点，即只有左子树或右子树。这种情况下，将 x 的父结点与 x 的子结点相连，然后删除 x 即可，需要修改信息的结点是从根结点到 x 的父结点的路径上的所有结点；

3) 欲删除的结点 x 同时具有左子树或右子树，这时就需要改变树的结构，可通过两种方式完成删除：① 用 x 的左子树中最大值的结点 y 替代 x，然后删除 y；② 用 x 的右子树中最小值的结点 z 替代 x，然后删除 z。这时，需要修改信息的结点是从根结点到 y 或 z 的父结点的路径上的所有结点。

在以上3种情况中，只需重新计算从SBT的根结点到所删除结点(或其子结点)的路径上的结点的摘要值，若引起了旋转调整操作，其结点摘要值更新过程与插入操作过程中的描述一致。

2 结果分析

通过上述分析，可得到以下结论：

1) 对于查询操作及验证操作，SBT认证字典及Merkle树认证字典的算法复杂度在同一数量级，均为 $O(\log n)$。与RSA树认证字典相比，这两种认证字典均使用哈希函数而没有复杂的模幂运算，因此计算更加简单。

2) 在对动态数据集的支持方面，Merkle树认证字典采用2-3树支持动态数据集，而RSA树认证字典则将成员哈希分组到固定大小的Bucket来支持动态数据集。这两种方式都不可避免认证结构的周期性重构。此外，数据成员的保序存储对云存储非常重要，而Bucket的存在使得数据集无法实现保序。

3) SBT具有良好的平衡性，能更好地控制树高，使其趋近于完全二叉树。二叉树上数据更新和查询的时间复杂度都与树高相关，相比于非平衡树而言，SBT的数据操作效率大大提高，可将数据操作的时间维持在 $O(\log n)$。同时，SBT认证字典也可完全避免认证结构的周期性重构。

因此，对于成员数量大且需经常更新的数据集，基于SBT的方案有明显的优势。为了验证SBT这种平衡树结构更适合构建认证字典，下面将通过实验对SBT与AVL树、Treap进行对比，评估基于SBT的数据完整性验证方案中数据查询和更新的时间复杂度。AVL树和Treap都是平衡二叉树，其中AVL树通过定义左右子树的高度差不大于1来维持平衡性，Treap则引入优先级域来维持平衡。

2.1 数据更新分析

针对插入操作的时间分析，在SBT、AVL和Treap结构中分别插入100万、200万、500万、1 000万和2 000万个随机值，插入操作的时间统计图如图2所示。

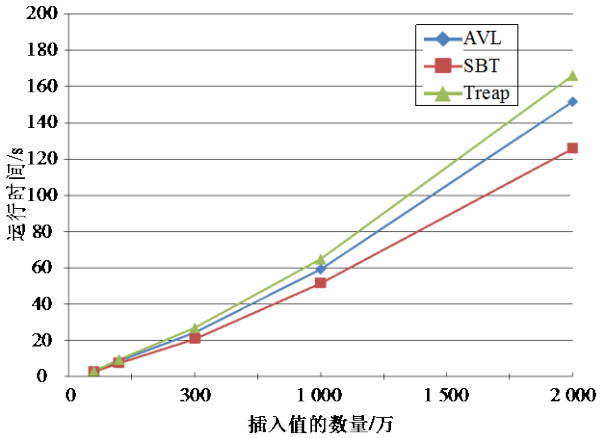

图2 插入操作的时间统计图

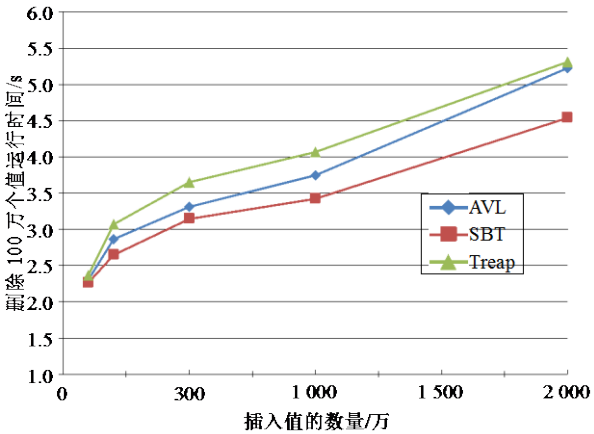

图3 删除操作的时间统计图

针对删除操作的时间分析，在具有100万、200万、500万、1 000万和2 000万个随机值的SBT、AVL和Treap结构中，分别计算删除100万个值的运行时间，删除操作的时间统计图如图3所示。

2.2 验证过程分析

对于用户发送的成员问询，不可信服务器在为其生成证据信息的过程中，需从认证字典中找到相

应的数据结点，并返回根结点到该数据结点的路径上的结点摘要值，及其相关的子结点和兄弟结点的摘要值。返回的摘要值的个数与SBT树的高度成线性关系，用户在对返回的证据进行验证计算时，需进行 $O(\log n)$ 次的哈希计算。

针对数据查询操作的时间分析，在具有100万、200万、500万、1 000万和2 000万个随机值的SBT、AVL和Treap结构中，分别查询100万个值的运行时间，查询验证操作的时间统计图如图4所示。

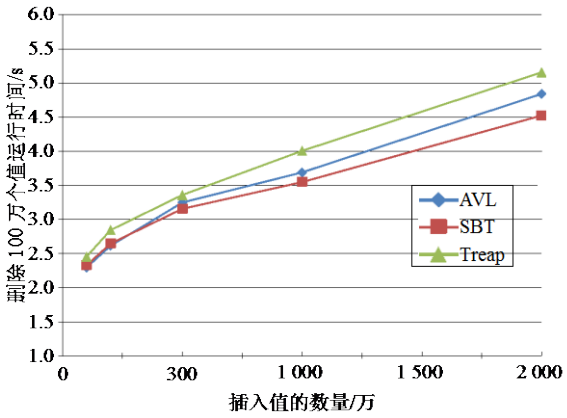

图4　查询验证操作的时间统计图

2.3　安全性分析

基于SBT的认证字典的安全性与基于Merkle树的认证字典相同，其安全保证都基于其所采用的哈希函数的抗碰撞性。

3　结　论

本文主要介绍了云存储中基于SBT树的数据完整性验证方案。该方案避免了传统认证字典在数据集更新时的周期性重构问题，从而能更好地支持动态数据集。通过实验对比发现，相较于其他两种常用的平衡树结构，对于数据更新、查询及验证操作，基于SBT的认证字典具有更高的效率。

参　考　文　献

[1] DEVANBU M, GERTZ, MARTEL C, et al. Authentic third-party data publication[C]//Proceedings of the IFIP TC11/WG11.3 Fourteenth Annual Working Conference on Database Security: Data and Application Security, Development and Directions. The Netherland: Kluwer, B.V. Deventer, 2000.

[2] NAOR M, NISSIM K. Certificate revocation and certificate update[C]//Proceedings of the 7th USENIX Security. Berkeley, USA: ACM, 1998.

[3] MICALI S. Efficient certificate revocation[R]. Berkeley: MIT Laboratory for Computer Science, 1996.

[4] GUNTER C, JIM T. Generalized certificate revocation[C]// Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, USA: ACM, 2000.

[5] GASSKO I, GEMMELL P S, MACKENZIE P. Efficient and fresh certification[C]//Proceedings of International Workshop on Practice and Theory in Public Key Cryptography. Australia: Springer-Verlag, 2000.

[6] MERKLE R C. A certified digital signature[J]. Lecture Notes in Computer Science, 1990(435): 218-238.

[7] MERKLE R C. Protocols for public key cryptosystems[C]//Proceedings of Symposium on Security and Privacy. USA: IEEE Computer Society Press, 1980.

[8] GOODRICH M T, TAMASSIA R. Efficient authenticated dictionaries with skip lists and commutative hashing[R]. USA: Johns Hopkins Information Security Institute, 2000.

[9] GOODRICH M T, TAMASSIA R, SCHWERIN A. Implementation of an authenticated dictionary with skip lists and commutative hashing[C]//Proceedings of DARPA Information Survivability Conference & Exposition II. Anaheim, California: IEEE , 2001.

[10] YUN A, SHI C, KIM Y. On protecting integrity and confidentiality of cryptographic file system for outsourced storage[C]//Proceedings of the 2009 ACM Workshop on Cloud Computing Security. Chicago, USA: ACM, 2009.

[11] PAPAMANTHOU C, TAMASSIA R, TRIANDOPOULOS N. Authenticated hash tables[C]//ACM Conference on Computer and Communications Security. New York, USA: ACM, 2008.

编　辑　叶　芳