文章编号：2096-1618(2016)04-0377-05

外包数据中基于链表的B+树数据完整性检测算法研究

方 睿，廖 勇，方 欣，周 敏，吴 强
(成都信息工程大学信息安全工程学院，四川 成都 610225)

摘要：外包数据是数据拥有者将自己的数据存储在第三方服务提供商的服务器上，数据完整性检测的目标是防止存储在不可信的第三方服务器上的数据被篡改，验证数据是否可信。在基于B+树完整性检测算法的研究基础上，提出一种基于链表的B+树完整性检测算法，给出算法的数据结构定义及分析验证过程。该算法数据结构是通过单循环链表与B+树相结合，动态的获取链表中数据的验证信息，同时在B+树的叶子结点存储一个链表，增加了存储空间效率。实验表明，当数据增大时，基于链表的B+树与B+树在查询时间上相比略有优势，在插入数据时，优势明显。

关 键 词：外包数据；链表；B+树；完整性检测；算法；数据结构定义；数据安全
中图分类号：TP309.2 **文献标志码**：A

1 背景

外包数据是用户选择将数据外包给第三方服务提供商，即存储服务提供商(storage service provide, SSP)，进行数据的管理、存档、备份和恢复，并保证其安全性[1]。但数据存储在第三方服务器上，数据就会受到安全挑战，主要分为两大类：隐秘性和完整性。前者主要采用加密技术，抵御攻击者获取到隐私数据，后者是防止数据丢失或篡改，通过合理的完整性检测方案对数据进行完整性检测，目前外包数据存储的完整性问题将是用户关注的首要安全问题，也是论文研究的重点。

数据外包模型如图1所示，数据拥有者通过数字签名与完整性验证算法将数据安全地存储在第三方不可信服务器上，而数据拥有者存储验证码和验证码生成规则，验证码用来验证不可信服务器上的数据是否被更改，用户从服务器上获取到数据拥有者的数据是真实的，而文中的研究主要是数据拥有者将数据存储到不可信服务器上，并对数据进行完整性检测研究。

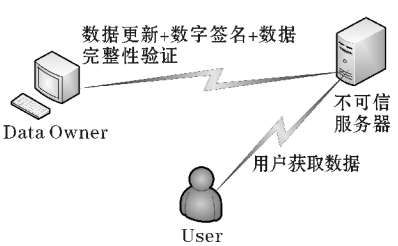

图1 数据外包模型

2 B+树完整性检测算法的研究现状

B+树数据结构，具有扩展性好、查询和更新快等特点，基于B+树为基本数据结构的数据完整性检测的研究是近几年数据完整性检测方案的研究热点[1]。基于前人研究基础上，Li F F[2]提出复杂的嵌入式MB-Tree的方案，在每个内部节点上增加一个内嵌的MB-Tree，用其根节点的Hash值取代传统MB-tree的节点Hash值。咸鹤群等[3]提出一种以带掩码的验证树(masked authenticating B+-Tree, MAB-Tree)作为核心数据结构的完整性检测算法，通过抽取两个掩码向量，避免使用大量幂指数运算，降低查询验证过程的计算代价，减少查询验证时间。耿纪昭[4]提出在B+Tree的数据结构上采用RSA累加器技术构造的完整性验证方案，与RSA树相比，该方案更新时间更为高效。

表1 不同Tree完整性检测算法的相关比较

	数字签名次数	安全性	更新时间效率	数据存储的空间	同一数据有无多个验证证据
MB-Tree	多次	Hash	一般	叶子结点	无
MAB-Tree	一次	Hash与RSA	中	叶子结点	无
B+Tree	无	Hash与RSA	高	叶子结点	无

收稿日期：2016-07-06

通过上述对现有的基于 B+树完整性检测算法的研究分析[2-4],从表 1 可知都是从时间效率以及数据安全上对数据完整性检测算法进行改进,然而原始数据存储在叶子结点之中,内部结点不存储原始数据,这样的数据结构存储空间利用率低,树结构建立后在不更新数据的情况下,一个数据仅有一个验证证据。

3 基于链表的 B+树完整性检测算法

针对上述情况,提出基于链表的 B+树数据完整性检测算法,将 m 个数据分割成固定大小为 n 的块,其中 $m \geq n$,块数量为$[m/n]$,每个块设计为一个链表,链表最大的存储空间为 n,通过变换链表中数据位置,同一数据获取到不同的验证值,来抵御数据存储在不可信服务器的数据受到的重放攻击[6],同时能保证查询验证过程效率是高效的,与现有方法比较,主要改进如下:

(1)存储空间的改进:由于经典 B+tree 结构总是将数据存储在叶子结点上,而内部结点并不存储实际数据,这无疑增加了服务器上的存储空间开销。而 SCLB+tree 优化了这一问题,减少分层结构,增加了叶子结点存储的数据。

(2)插入数据的改进:SCLB+tree 插入数据是插入到链表的链首,插入链表的时间为 $O(1)$,而查询到该链表的时间为 $O([m/n]/2)$,与经典 B+tree 插入数据的时间相比有明显的优势。

(3)叶子结点改进:采用单循环链表,保证每个值的位置是唯一的,同时每个值都有前后结点,保证每个结点都有 List(hi,hj)验证值,防止攻击者判断原始数据在链表中的位置。

3.1 数据结构定义

在大数据系统中,直接利用 B+树来验证数据完整性,查找操作都要经历一条从根结点到叶子结点的路径,查询时间效率低[5]。文中将 B+树的结点设计为固定大小的数据块,其阶树为变量,将数据存储在单循环链表中,数据分块的区间数值建立 B+树,链表存储空间可以随着存储的数据变化而变化,存储空间≤n,同时利用随机变量旋转链表里面的数据,将其命名为单循环链表 B+树(Single Circular List B+tree, SCLB+tree)。图 2 为一个 3 层 6 阶 SCLB+tree 实例,将数字 1 到 60 进行分组存储。

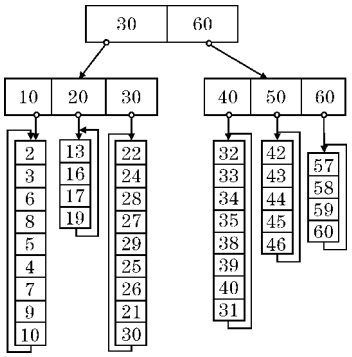

图 2 SCLB+tree 数据结构

3.2 验证树和计算方法定义

定义 1. 基于 SCLB+tree 的验证树。SCLB+tree = < Root, INODE, List, $H1$, , $X(v)$, List(hi, hj), $N1$ >。Root 是根结点,INODE 为内部结点,List 为叶子结点链表里面的数据集合(以下简称叶子结点链表),$H1$ 是数据结构中各数据的 hash 值,$X(v)$为结点摘要值,List(hi, hj)中的 hi 和 hj 分别对应叶子结点链表中对应数据的前一个数据和后一个数据的 hash 值,定义如下:

(1)对于每个结点 $H1$,定义 2:

$$H1 = h1(v)$$

(2)对于叶子结点链表的摘要值,定义 3:

$$X(v) = g^{y1(x1)y1(x2)\cdots y1(xn)} \bmod N$$

其中 v 为结点,$h1$ 为数据拥有者所选择的 hash 函数,$h1(v)$表示对应结点的 hash 值,$x1$,$x2$,…,xn 是对应该叶子链表包含的全部数据元素值,而 $y1(xi)$是元素值 xi 的素代表,N 是可信数据源选择大素数 $p1$ 和 $q1$,计算 $N=p1q1$,g 是选择与 N 互素的大基数,$y1(xi) = h1(xi)$,其中文章出现的 i 表示数据结构中的结点。

(3)对于非叶子结点的摘要值,定义 4:

$$X(v) = g\pi_{u \in N(v)}^{(x(u))\ rj} \bmod N$$

其中 $r(x(u))$是通过使用哈希函数 $h1$ 得出的$x(u)$的素代表,$N(v)$指的是结点 v 的子结点的集合,j 对应结点在验证树中所在的层数,$0 \leq j$,叶子结点层数为 1。

(4)对于 $N1$,$N1$ 是一个 $1 \leq N1 \leq n$ 的随机整数,用来改变链表元素的位置关系。

3.3 数据完整性验证过程

原始数据发送到第三方不可信服务器上后,用户拥有者将与 N 互素的大基数 g、$h1$ 以及 N 广播给不可信服务器和用户,将 $p1$ 和 $q1$ 秘密持有,第三方服务器根据提供的值建立验证树,然后对根结点的验证值进行数字签名处理。

当数据拥有者首次查找数据 xi 是否存储在数据集 S 中,全文以 3 层 SCLB+tree 为例,不可信服务器首

先从根结点通过二分法查找到存储该数据的叶子链表结点,然后在链表中查找该数据,直到查找到 xi,然后不可信服务器返回给用户的证据信息为从根结点到叶子结点的内部结点一串信息 $w1,w2,w3$,以及该元素对应的 List$[hi,hj]$值。

$$W1=(g^{y1(x1)y1(x2)\cdots y1(xn)} \bmod N, r1(x(u)))$$
$$W2=(g^{w1r2(x(u))} \bmod N, r2(x(u)))$$
$$W3=(g^{w2r2(x(u))} \bmod N, r3(x(u)))$$

当数据拥有者接收到不可信服务器的回应,就可以判断出数据存储在该服务器上,并接收到该元素的验证信息,然后数据拥有者获取验证信息进入验证阶段,数据验证分为以下几步:

(1)首先,验证获取到的数据消息是否来自于外包数据的第三方服务器,对 $w3$ 进行数字签名验证,并验证与数据拥有者存储的 $w3$ 是否相等,若都验证成功,则进行下一步。

(2)其次,数据拥有者检测该数据对应叶子链表的验证信息是否相同,即 $w1$。并验证获取到的 List(hi,hj)值确认该数据在链表中的位置,若与数据拥有者存储在本地的 List(hi,hj)相同,则验证成功,继续一步验证,否则结束验证

(3)验证 $w2$ 的值与数据拥有者对应该数据的 $w2$ 的值是否相同

当这 3 步判断全部成立时,数据拥有者才认为不可信服务器返回的答案是正确的。

当数据拥有者非首次验证数据 x 是否存储在第三方时,便增加一个随机变量 $N1$,将链表里面的位置随变量 $N1$ 重新排序,改变同一个数据 x 对应的 List(hi,hj)值,防止重放攻击[6],其他验证方式与首次验证数据是否存储在第三方是一样的。

3.4　SCLB+tree 的更新方法

3.4.1 删除数据

由于数据都存储在叶子结点的链表中,首先找到存储该数据的叶子结点的链表 a。删除操作跟单循环链表一样[7],查询时间效率为 $O(n)$,进行如下操作,其中 p 为要删除的数据的结点,pre 为要删除数据的前驱结点:

```
p = p->next;
p->next = p->next->next;
free(p);
```

如删除后该链表没有数据则收回存储空间 free(a)。

数据完整性验证信息变化如下:List(hi,hj)中 hi 与 hj 与 q 相邻的结点,hj 需要变成 $p->next$ 的哈希值,hi 需要变成 p 的前一个结点的哈希值。而对于验证值 wi,由于当代表值 xi 与 $p-1$ 和 $q-1$ 互为素数的时候,根据扩展的欧几里得算法[8],可以在常量级时间内删除 x,计算方法如下:

$x<\!\!-x_{i-1} \bmod \phi(N)$,$A_{new}<\!\!-A_{xold} \bmod N$,一次删除操作只需要 $O(1)$的时间即可完成。

3.4.2　插入数据

首先找到数据在 SCLB+tree 的叶子结点链表,若无链表,则创建链表,若存在已有链表,则申请一个结点空间,将数据 q 插入到链表的首节点,操作如下,其中 q 为插入的数据,$head$ 为首结点,$rear$ 为尾结点:

```
q->next = rear->next;
rear->next = q;
```

若没有链表,则申请头结点空间,将数据存入其中。

数据完整性验证信息变化如下:q 生成的 hash 值为 $h(q)$,head 的 List(hi,hj)中的 hi 变成 $h(q)$,read 的 List(hi,hj)中的 hj 变成 $h(q)$,而 q 的 List(hi,hj)中 hi 为 rear 的哈希值,hj 为 head 的哈希值。而对应该链表的验证值 $wi,x(q)$是元素值 q 的素代表 $w1_{new}=w1_{old}{}^{x} \bmod N$,$w2_{new}=w2_{old}{}^{x} \bmod N$,$w3_{new}=w3_{old}{}^{x} \bmod \bmod N$。时间复杂度为 $O(1)$。

4　算法分析

4.1　安全分析

安全问题主要是对存储在第三方服务器上的数据进行篡改或者丢失[9]。算法进行完整性检测的安全性主要有数字签名,hash 值以及变换链表数据位置,前面两点文献[10]已经给出了证明。

变换链表数据位置进行安全性分析:由于数据对应的 hash 值生成后不再进行改变,而服务器可以更改原始数据,但是验证码还是以前生成的值,而数据拥有者会认为原始数据没被更改,导致用户获取到错误的数据,因此通过随机生成一个值来更改数据位置,通过更改验证码中该值的 List(hi,hj)中的 hi,hj 防止这种攻击,当验证值相同时,说明更改无效。同时根据 RSA 累加器原理[11],更改链表中数据的 List(hi,hj)不会更改非叶子结点的验证码,从而更改验证码只与链表有关,同时可以很自由的更改某个部分的链表的验证码,因此减少了更改时间效率。在验证某个值在某个链表中的时候,验证值相同的概率为 $1/n$,所以对于某个链表中的某个值增加了$(n-1)$个验证码,n 越大时,验证码便会越多。

4.2　时间分析

文中验证码第一次生成后,在不更改数据的前提

下,便不再重新生成,文献[12]对这部分时间已经给出了证明,验证过程只需进行数据拥有端与服务器上的验证码进行比较即可,该文主要分析查询时间效率和验证时间,影响查询执行时间的因素主要有:查询验证过程中的数据结构和数据量,硬件设施的计算速度等,影响验证时间的因素主要有:随机变量 $N1$ 旋转链表时间、更改链表中 List(hi,hj) 的时间以及从根结点到叶子链表路径上的所有验证值进行比较的时间。

4.2.1 查询时间效率分析

在不旋转链表的情况下,通过 SCLB+tree 与 B+tree[13] 对数据查询时间进行对比,令他们时间效率分别为 $O(1)$ 和 $O(2)$ 分析如下:

(1)当数据较少时,由于 SLB+tree 是通过二分法对数据进行分组,因此 $O(1)$ 查询时间稍微要高于 $O(2)$。

(2)当数据增多时,$O(1)$ 的时间趋向链表查询时间,其复杂度为 $O(n)$,$O(2)$ 的时间趋向二叉树的时间复杂度为 $O(\log_2 m)$,由于 n 是固定不变的,而 m 是数据量,则随着 m 增加,$O(2)$ 将大于 $O(1)$。

通过实验证明,增加两者树中的数据,在相同的条件下,同时令 $n=1000$,对数据范围在 $10^3 \sim 10^7$ 进行平均查询时间比较,实验结果如图 3 所示。

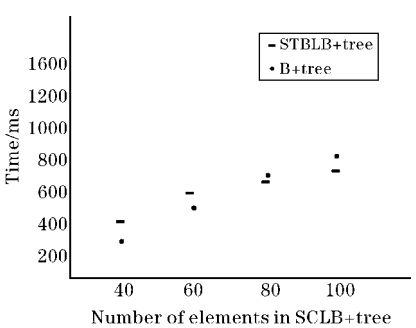

图 3 SCLB+tree 与 B+tree 在查询时间上的比较

4.2.2 更改链表验证值时间效率分析

由于经典 B+tree 的验证值一旦生成便难以更改,SCLB+tree 通过旋转链表增加旋转时间与更改每个链表结点的 List(hi,hj) 中 hi 和 hj 的时间,令数据块大小为 n,数据量为 m,用户可以灵活地更改需要验证数据所在块里的验证值,这样可以把时间控制在 $O(n)$。

4.2.3 验证数据结构更新时间效率分析

验证数据结构的更新效率是评价数据完整性保护方案的另一个重要性能指标,尤其是对于大数据量而言,不便于更新数据。

修改数据库元组内容时,分析 B-tree 更新验证数据结构的代价,令 H 表示树的高度,n 表示树的度,h 表示 hash 运算,s 表示签名运算,r 表示读取结点中一个键值的操作,l 表示修改链表中的 List(hi,hj),修改一条元组所需的验证数据结构更新操作代价分别为:

MB-tree 方案,令更新操作为 update

update $=H\times(n\times r+h)+s$;

VB-tree 方案

update $=H\times(n\times r+h+s)$;

SCLB+tree 方案

Update $=H\times(n+h)+s+l$;

MB-tree 方案每次计算新的 hash 值都需要读取内部结点上所有的键值,最后对根结点进行签名,VB-tree 方案在每层结点上比 MB-tree 方案多执行了一次数字签名操作,SCLB+tree 更新过程不需要读取内部结点的所有键值,仅读取每个叶子链表对应的内部结点然后生成验证值,同时,由于 SCLB+tree 的每个叶子是一个链表,很明显 H 要小于 MB-tree 与 VB-tree 里面的 H,但是由于修改每个链表中的一个值,则它的前结点与后结点的 List(hi,hj) 分别修改一次,但是很明显是小于 H 的变化的,因此 SCLB+tree 中 update 更新代价最小。

插入操作,在不考虑 B+-tree 内部结点分裂的情况下,令插入的数据生成 hash 值代价相同,由于 SCLB+tree的 H 要明显小于其他两种树的 H,所以查找代价要小于其他两种 B-tree,因此 SCLB+tree 插入数据代价要小一些。

5 结束语

通过对现有树在外包数据的完整性保护的研究与分析[14-15],提出一种基于链表的 B+树完整性检测算法,该算法提出 SCLB+tree 数据结构以及更新操作与完整性验证过程,结合链表与 B+树两者的特点,减少了服务器存储开销,同时能够动态的更改数据的验证码,从而能有效地防止重放攻击,同时与 MB-tree 和 VB-tree 在验证数据结构更新时间效率上进行对比,可知在同等条件下 SCLB+tree 花费时间较少,通过这些分析可知,SCLB+tree 该数据结构用于数据完整性验证是可行的。

由于是通过 n 进行 SCLB+tree 划分的,当数据较少时,SCLB+tree 在验证与查询效率上并没有什么优势,因此 SCLB+tree 不适合小数据量的完整性检测,同时用二分法的方法建立该数据结构,因此前提是需要知道数据的范围,才能更好地有效划分,这两部分的问题,需要进一步进行解决。

参考文献:

[1] Rao J, Ross K A. Cache Conscious Indexing for Decision-Support in Main Memory[C]//Processdings of the 25th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc. 1999:78-89.

[2] Li F F, Hadjieleftheriou M, kollios G. Dynamic authenticated index structures for outsourced databases[C] //Proc of ACM SIGMOD 2006. New York :ACM,2006:121-132.

[3] 咸鹤群,冯登国.外包数据库模型中的完整性检测方案[J].计算机研究与发展,2010,47(6):1107-1115.

[4] 耿纪昭。云存储中数据完整性验证机制的研究与实现[D].成都:电子科技大学,2013.

[5] 耿庆田,狄婧,常亮,等.基于 B+树的数据索引存储[J].吉林大学学报:理学版,2013,(11).

[6] 孙岚,吴英杰,罗钊,等.路由环境下防止重放攻击的位置隐私保护算法[J].华中科技大学学报,2013,(12).

[7] 胡传福.从链表的实现论 C/C++与数据结构教学[J].东莞理工学院学报,2013,(6).

[8] 王宏俊,丁群.RSA 公钥算法研究与快速模幂运算设计[J].黑龙江大学工程学报,2013,(5).

[9] 张敏.外包存储环境下数据共享的安全方案研究[D].成都:电子科技大学,2010.

[10] 孔范新.云环境下的数据安全研究[D].济南:山东大学,2014.

[11] 文辉灰.基于动态累加器的直接匿名证明方案研究[D].兰州:兰州大学,2014.

[12] 姚戈.云存储数据完整性验证机制研究[D].北京:北京交通大学,2016.

[13] 王东.基于动态结点流行度的 B+树索引研究[D].郑州:郑州大学,2014.

[14] 苏兰.面向计算的数据完整性检验方法研究与实现[D].成都:电子科技大学,2013.

[15] 江捷斯。云环境下远程数据完整性认证数据结构研究[D].广州:华南理工大学,2014.

Data Integrity Checking Algorithm Research in Outsource Data based on B+Tree of the Linked List

FANG Rui, LIAO Yong, FANG Xin, ZHOU Min, WU Qiang

(College of Information Security Engineering, Chengdu Universityof Information Technology, Chengdu 610225, China)

Abstract: Outsourcing data is what the users store their own data in a third-party service provider´s server, and the goal of data integrity checking protection is to prevent data which is in storage in a distrusted third-party server from being tampered and verify the data whether it is credible or not. Based on B+tree integrity checking algorithm in this paper, a data integrity checking algorithm based on B+tree of the linked list has been put forward and the data structure definition of algorithm and the verification of analysis have been given. The scheme obtains the authentication information dynamically in linked list by means of combining single circular linked list with B+tree, at the same time, it can increase the storage space and efficiency by means of storing a linked list in leaf nodes of the B+tree. Experiment shows that when the data increases, the B+tree based on linked list has a slight advantage in comparison to B+ in the query time, while when inserting data, the advantage is obvious.

Key words: outsource data; linked list; B+tree; integrity checking; algorithm; data structure definition; data security