文章编号：2096-1618(2018)01-0044-06

基于链表多分支路径树的云存储数据完整性验证机制

方 欣[1]，方 睿[1]，刘雪涛[2]，廖 勇[1]，浦 东[1]，贾 川[1]
(1.成都信息工程大学网络空间安全学院，四川 成都 610225；2.云南省气象台，云南 昆明 650042)

摘要：低成本、高扩展性云存储服务的出现，用户将自己的数据转移到云服务器上，这很大程度地节约了用户对数据的存储和后期维护开销。与此同时，用户也不再拥有对数据的绝对管控权，这意味着用户无法确认存储在云服务器上的数据信息是否完整。在目前的研究基础上改进，提出基于链表多分支路径树(single linked list large branching tree，SLBT)的完整性验证机制，该方案将多分支路径树中单一的叶子节点设计为单向链表节点，使其更有利于插入和删除数据块。分析结果证明得到，该方案在支持数据全动态更新的基础上，还提高了数据的空间存储效率，减少了动态数据结构重构代价。

关 键 词：云存储；数据完整性验证；SLBT树；动态更新
中图分类号：TP393 **文献标志码**：A
doi：10.16836/j.cnki.jcuit.2018.01.009

0 引言

随着信息技术的迅猛发展，以及各个应用领域日益增长的庞大信息量，传统的数据存储技术已经满足不了用户的存储空间需求。而眼下正兴起的云存储技术则为用户提供了一个高性价比的存储方式。云存储是一个以数据存储和管理为核心的云计算系统，能够为用户提供动态可伸缩性的数据存储服务，其最大的特点是存储即服务(SaaS)，即用户可以通过服务的方式，按需计量随时随地使用云存储中丰富的资源[1]，云存储技术由于其低成本、高扩展性、易运维等优点已然成为当前的研究热点与重点。然而云存储技术带来方便的同时，面临的各种安全问题不可小觑。如2010年，Google两名员工侵入租户Google Voice、Gtalk等账户，引起隐私数据泄露[2]；2011年3月，谷歌Gmail邮箱出现故障，而这一故障造成大约15万用户的数据丢失[3]。故存储在云端的数据有可能遭到窥视、篡改、恶意损坏等威胁，甚至数据遭到攻击后，云服务商出于自身利益角度的原因，不会第一时间通知用户甚至直接隐瞒用户。因此，用户无法确定存储在云端的数据是完整的。而数据完整性保护是数据存储在云端上对其进行完整性检测，防止数据丢失、篡改及查询结果被非法更改。目前，用户将数据存储在云端上引起数据存储安全问题是用户的担忧所在，而数据完整性保护是外包数据存储安全保护的重要环节之一。

在云存储环境下，数据的完整性检测是数据完整性保护的主要课题。数据完整性检测是指只需要用户从云服务器上取回少量的存储数据，通过某种知识证明协议等方法即可达到验证数据完整性的目的。Ateniese等[4]首先提出可证明数据持有(provable data possession，PDP)机制，但是该方案不支持公开验证，并且实际应用性不强。Wang等[5]提出了一种支持全动态操作的数据完整性验证机制。该方案将BLS签名的同态认证标签和Merkle[6]哈希树(MHT)相结合构造出一种完整性验证机制，数据结构的引入使方案支持数据的动态更新，同时利用MHT树保证数据块在位置上的正确性，利用BLS签名保证数据块的完整性[7]。然而在该方案中，MHT树会随着数据块数目的增多而大量消耗时间和空间。针对这一问题，李勇等[8]引入多分支路径树(large branching tree，LBT)结构保证数据块位置的正确性，提出了基于多分支路径树的数据完整性验证机制，简化了数据动态更新过程。但该方案存在LBT树结构的空间存储效率低下和用户更新数据时LBT树结构重构所需计算开销较大两个方面问题。

针对LBT树的数据存储空间效率低和重构开销大这两个问题，在LBT树的基础上作出改进，提出一种新的数据结构，将其命名为单链表多分支路径树(single linked list large branching tree，SLBT)。与LBT树相比，SLBT树主要改进有：(1)数据存储空间的改进：由于LBT树是将所有数据块的哈希映射值存储在叶子节点中，而中间层节点不存储和数据相关的信息，这必然会额外消耗云存储服务器大量的存储开销。而SLBT数将LBT树中底层的叶子节点设计为单向链表，降低了树的分层结构，减少了中间节点的输出，增

收稿日期：2017-06-26
基金项目：四川省科技厅资助项目(2017GZ0331)

加了数据存储空间效率。(2)插入数据的改进:当用户需要插入文件 f 时,若文件长度过小,则需要构造空的数据块叶子节点插入到 LBT 树中,而 SLBT 树采用链表结构可以免去插入空数据块这一操作,提高了数据更新效率。(3)树重构问题的改进:每当叶子结点被更改时,从叶子结点到根节点路径上的所有结点值都要重新计算一遍,数据结构也就需要重构一次,从叶子结点到根节点之间的路径越长,重构所需的开销越大。由于 SLBT 树减少了分层结构,故 SLBT 树从叶子节点到根节点的路径变短,SLBT 树重构所需的开销降低。因此提出的方案优化了基于 LBT 树的云存储数据完整性验证机制。

1　相关预备知识

1.1　单链表多分支路径树 SLBT

在 LBT 树中,若树的出度为 $n(n\geqslant 2)$,即除了叶子结点,每个中间层节点和根节点都有 n 个子节点,树高为 h,则叶子结点的总量为 n^h。若定义叶子结点所在的层数为 0,那么在处于树中第 i 层的所有节点个数则为 n^{h-i}。在完整性验证方案中,每个叶子结点都对应着一个数据块的哈希值,树中的每个父节点都是其 n 个子节点的哈希值相互链接而成,以此类推,直到根节点值。和 LBT 相比,SLBT 具有以下特征:

由于链表在树结构的最底层,若定义链表所在位置为第 0 层,那么链表 L 的数量和第 1 层节点的数量值相等为 n^{h-1}。

每条链表初始装载的节点个数为 $n^h/n^{h-1}=n$,链表的存储空间足够大,链表中的内部节点依次存储着数据块的散列函数值,而不再像 LBT 树那样为每一个数据块的映射值构建一个树节点的分支。

在第 1 层节点的哈希值是链表内所有节点的哈希值链接而成,在第 2 层每个父节点中的哈希值为其所有分支子节点的哈希值相互链接,然后以此类推,不断向上链接直到根节点 R。

如图 1 是一个树高 $h=2$,出度为 n 的 SLBT 树实例,若数据块数目为 n^2,该数据结构底层则由 n 个单向链表$(L_1,L_2,\cdots,L_n)$组成,节点 M_1、M_2、M_n 以及根节点 R 的值依次为:

节点 $M_1=h(h(F_1)||h(F_2)||\cdots||h(F_n))$

节点 $M_2=h(h(F_{n+1})||h(F_{n+2})||\cdots||h(F_{2n}))$

节点 $M_n=h(h(F_{n^2-n})||h(F_{n^2-n+1})||\cdots||h(F_{n^2}))$

根节点 $R=h(h(M_1)||h(M_2)||\cdots||h(M_n))$

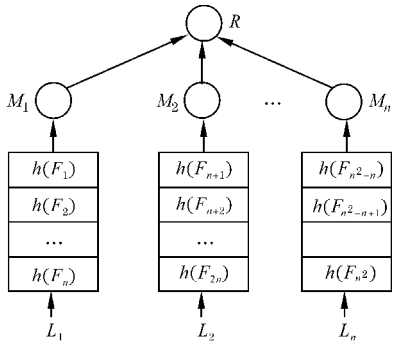

图 1　SLBT 树数据结构

1.2　系统验证模型

基于挑战应答协议的数据完整验证机制,其系统模型包含 3 部分。用户:即数据的拥有者,但计算存储能力有限,用户将数据存储到云服务器上后,用户随时可以对自己的数据进行完整性检测及增删改查等更新操作;云存储服务器(CSS):提供对用户数据的管理服务以及计算服务,接受验证方发送的挑战请求,并返回应答信息;第三方审计员(TPA):接受用户的授权后对 CSS 发送挑战信息,并代替用户审核数据,然后将审计结果返回给用户,是专业的数据审计权威机构;在用户将所有数据发送到 CSS 之前,首先将数据的元数据输出,然后同 CSS 和 TPA 协商密钥;由于用户不可能随时保持在线,当用户处于离线状态时,由授权后的 TPA 对 CSS 发起挑战,CSS 将响应的证据发送给 TPA,然后 TPA 对返回的凭据进行审核。

在整个过程中,系统涉及的相关算法如下:

KeyGen( ):输入→安全的参数,输出→公钥 PK 和私钥 SK。

SigGen( ):将数据预处理,计算出元数据用于后续验证工作。输入→公钥 PK、私钥 SK 和文件 F,输出→每个数据块的标签值 T,汇合成标签集合 $\boldsymbol{\Phi}$。

GenPro( ):CSS 端根据验证方发送的挑战消息,生成证据。输入→公钥 PK、文件 F、标签集合 $\boldsymbol{\Phi}$,输出→数据块证据 P,标签证据 $\boldsymbol{\Theta}$。

VerPro( ):由用户或者 TPA 运行,输入→数据块证据 P、标签证据 $\boldsymbol{\Theta}$ 和文件 F,输出→1 代表验证通过,否者为 0 未通过。

2　基于 SLBT 树的数据完整性验证方案

2.1　方案构造

在构造的方案中,分以下几个过程。

(1)初始化:数据文件 F 被用户划分成长度相等

的数据块 $\{F_1,F_2,\cdots,F_n\}$，接着运行算法 *KeyGen*，选取非负随机数 $x \leftarrow Z_p, u \leftarrow Z_p$，其中 g 为有限域生成元，令 $y=g^x$，然后运行算法 *SigGen*，为每一个数据块 F_i 计算出相应的标签值 $T_i=(h(F_i)\cdot u^{F_i})^x$，将所有标签值汇合成一个集合 Φ：$\{T_1,T_2,\cdots,T_n\}$，随后用户根据数据块构造一棵 SLBT 树，通过不断迭代计算出根节点值 Root，并对根节点进行数字签名 $Sig_{sk}(h(Root))=h(Root)^x$。最后将数据块 $\{F_1,F_2,\cdots,F_n\}$、标签集合 Φ、SLBT 树根节点签名值 $Sig_{sk}(h(Root))$ 发送给 CSS 端。CSS 端存储接收到的数据同时构建 SLBT 树结构。

(2)发送问询：此过程由用户或者授权后的 TPA 不定时地向 CSS 发起挑战来验证数据的完整性。TPA 从数据块索引值集合 $\{1,2,\cdots,n\}$ 中随机选择 e 个数值，将 e 个随机数值构建索引集合 $I=\{s_1,s_2,\cdots,s_e\}$，同时为 s_i 选取对应的随机数 v_i。随之 TPA 向 CSS 发送挑战信息 $chal=\{(s_1,v_1),(s_2,v_2),\cdots,(s_e,v_e)\}$。

(3)返回证据：此过程将运行算法 *GenPro*。CSS 根据挑战信息中索引值找到链表中对应的数据块的映射节点，通过对树结构自下而上地回溯，找到该节点到根节点之间路径上的所有兄弟节点，将所有的兄弟节点汇聚成一个集合即为辅助信息 $\{\Omega_i\}_{s_1\le i\le s_e}$。另外 CSS 进行以下计算：

$$P=\sum_{1}^{s_e} v_i F_i$$

$$T_i=\prod_{i=s_1}^{s_e} T^{v_i}$$

CSS 将证据 $Pro=\{(h(F_i)),P,T,\{\Omega_i\}_{s_1\le i\le s_e},Sig_{sk}(h(Root))\}$ 返回给验证方。

(4)TPA 收到证据后，运行算法 *VerPro*，结合双线性映射性质[9]，进行以下验证计算。

(i)根据 $(h(F_i))$ 和 $\{\Omega_i\}_{s_1\le i\le s_e}$ TPA 重新计算出根节点值 $Root'$。

(ii)验证 $e(Sig_{sk}(h(Root)),g)=e(h(Root'),g^x)$。若验证通过则进行下一步验证计算，否则验证错误。

(iii)检验 $e(T,g)=e(\prod_{i=s_1}^{s_e} h(F_i)^x.u^P,y)$。通过验证输出 1，未通过输出 0。

2.2 数据的动态更新

当用户需要对数据文件更新时，首先计算出新的数据块的认证标签值，然后向 CSS 发送更新指令(插入、删除和修改等)，直接体现在对 SLBT 树树底层的链表节点的各种数据操作，CSS 会根据接收到用户的更新指令调整 SLBT 的树结构，同时将链表节点到根节点那条路径上所经过节点的全部兄弟节点值重置，并重新生成 SLBT 树的根节点值，接下来则根据过程(3)和(4)继续完成验证工作. 和 LBT 比较，SLBT 树结构的特殊性可以支持插入更多的数据块，下面以更新一个数据块的实例来详细介绍。

2.2.1 插入操作

假设用户需要在 F_n 后插入数据块 F_n^* 时，按数据块的索引标号找到对应的链表，链表 L_1 执行插入操作。根据用户的具体需求决定是将数据节点插入表首还是链表的中间，当 $h(F_n^*)$ 数据节点插入到 LBT 树的正确位置后，则形成新的 SLBT 树，所以在旧的 SLBT 树中的蓝色部分节点(如图 2 所示)均需要重新生成，即 CSS 要更新从链表节点到根节点的路径上的所有节点。故：

$$M_1^*=h(h(F_1)||h(F_2)||\cdots||h(F_n)||h(F_n^*))$$

$$R^*=h(h(M_1^*)||h(M_2)||\cdots||h(M_n))$$

此时 SLBT 树则变成如图 2 所示。

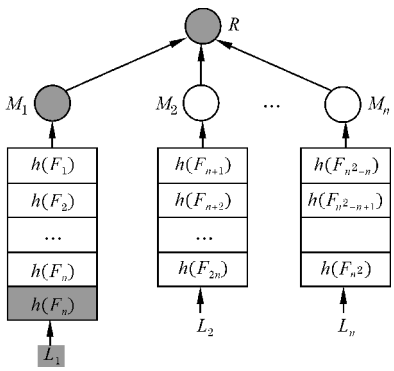

图 2 在数据块 F_n 后插入 F_n^*

2.2.2 删除操作

假设用户需要将数据块 F_{2n} 删除，首先按数据块的索引标号找到对应的链表，然后链表 L_2 查询到存储 $h(F_2)$ 的内部节点并执行删除操作，同时 CSS 需要重新计算认证路径上的所有节点值，图 3 中蓝色部分即为 SLBT 树需要更新的节点，均需要重新计算：

$$M_2^*=h(h(F_{n+1})||h(F_{n+2})||\cdots||h(F_{2n-1}))$$

$$R^*=h(h(M_1)||h(M_2^*)||\cdots||h(M_n))$$

此时 SLBT 树则变成如图 3 所示。

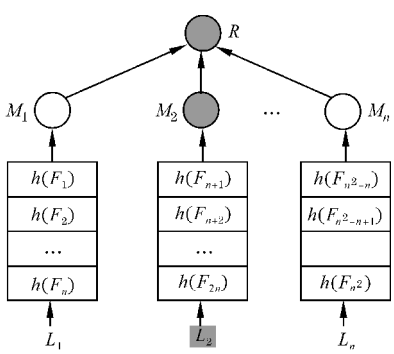

图 3 删除数据块 F_{2n}

2.2.3　修改操作

若用户需更新数据块 F_{n+3},则 CSS 端修改链表 L_2 中节点 $h(F_{n+3})$ 为 $h(F_{n+3})^*$,具体过程和插入、删除操作方法类似,在此就不再赘述。

3　安全分析

文献[8]已经证明了没有敌手可以在不可忽略的概率内让验证方接受证据,除非返回正确值。除此之外,用户数据存储在 CSS 上还面临着两种常见的攻击。

3.1　防止丢失攻击

丢失攻击是指云服务商故意删除、遗忘或者损坏用户使用频率不高的某个数据所造成的一种攻击[10]。假设不法云服务商丢失了数据块 F_j,当用户或者 TPA 又向 CSS 端发起挑战时,这时 CSS 端就得伪造一个数据块标签 $T_j=(h(F_j).u^{F_j})^x$,然而 x 作为私钥只有用户才知晓,所以CSS无法伪造出正确的数据标签糊弄过去,同时在验证 $e(T,g)=e(\prod_{i=s_1}^{s_e}h(F_i)^x.u^P,y)$ 等式时也会不通过,故云服务商不可信。

3.2　防止篡改丢失

篡改攻击是指云服务商未在用户授权的情况下就私自改动用户存储在 CSS 的数据信息,并试图掩盖。假设云端在未经用户的允许下修改数据块 F_j 为 F_j^*,这时候当验证方发送挑战信息给 CSS 端后,和上述同理,CSS 端不知道密钥信息,无法生成一个正确的标签 $T_j=(h(F_j)\cdot u^{F_j})^x$,验证方在审计该等式 $e(T,g)=e(\prod_{i=s_1}^{s_e}h(F_i)^x\cdot u^P,y)$ 的时候不成立,也就知道了云服务商是不可信的,存储在 CSS 上的数据已经不完整。

4　性能分析

将本方案与现有方案[5,8]进行对比,结果如表 1 所示,其中 N 表示数据块数量,n 表示 LBT 数和 SLBT 树的出度,h 表示 SLBT 数的高度。

表 1　方案性能比较

方案类型	支持公开验证	支持全动态更新	额外空间存储开销	根节点计算复杂度	通信复杂度
方案[5]	否	否	无	$O(\log_2 N)$	$O(\log_2 N)$
方案[8]	是	是	$O(\frac{N-1}{n-1})$	$O(\log_n N)$	$O(\log_2 N)$
本方案	是	是	$O(n)$	$O(h)$	$O(N)$

从表 1 得知,本方案在空间存储开销和根节点计算复杂度方面都是优于现有方案的。

4.1　空间存储效率分析

经典的 MHT 结构是将所有的数据存储在最底层的叶子节点,当数据量 N 增多时,叶子节点也就越多,树的深度 $\log_2^N$ 也呈对数函数增长,内部节点也随之增多,云服务器的存储负担越大,。虽然 LBT 树通过增加树的出度 n,降低了将树的高度降低为 $\log_n^N$。但当数据量 N 数值很大时,毋庸置疑,LBT 树结构的中间层节点数量也很大。

(1)假设 LBT 树的出度为 n,深度为 H,那么叶子节点数为 n^H 个,LBT 树的中间层节点(除去根节点和叶子节点)数为 $n^{H-1}+\cdots+n+1$。

(2)SLBT 树叶子节点用链表存储,假设链表的长度为 M,SLBT 树的层数为 $h(h\geq 2)$(链表的层数为 0),出度为 n,则它的叶子节点数为 $n^{h-1}\times M$。

当 $h=2$ 时,SLBT 树的中间层节点数为 n。

当 $h>2$ 时,SLBT 树的中间层节点数为 $n^{h-1}+\cdots+n+1$。

其中 M 是一个固定值,这是在数据块均为 N,并且链表存储节点用完的情况下,显然 $H>h$,所以 SLBT 树需要存储中间层节点数量更小,故提出的方案所需的存储空间开销值更小。当增加数据块时,SLBT 树通过增加链表中的内部节点,控制数树的高度,而不会增加多余的中间层节点,进而减少了不必要的存储开销,增加了数据的空间存储效率。

4.2　根节点计算复杂度分析

每当数据块执行了更新操作时,如图 2 和图 3 的标志蓝色的节点,验证树需要更新从链表节点到根节点路径上的所有节点值,树的深度越高,从链表节点到根节点的所历经的路径长度越长,叶子节点从下至上计算至根节点的开销就越大,故根节点的计算复杂度就越大,和树的高度有直接关系,。由4.1可知 $h<H$,其中 $H=\log_n^N$,而 $h\geq 2$,则 $O(h)<O(H)$。所以 SLBT 树根节点的计算复杂度更低,更新效率更高,从而减少树结构更新时产生的重构问题。

4.3 通信效率分析

当 CSS 端向验证方发送证据 $Pro=\{(h(F_i)),P,T,\{\Omega_i\}_{s_1\le i\le s_e},Sig_{sk}(h(Root))\}$，其中辅助信息 $\{\Omega_i\}_{s_1\le i\le s_e}$ 的数据量影响着整个通信过程开销。假设审计方向 CSS 发起挑战想要验证数据块 F_j，CSS 会根据数据块的索引号搜索 SLBT 树找到链表中对应的数据块所映射的节点 $h(F_j)$，然后找到该节点直到根节点这条路径上的所有兄弟节点。

对于 LBT 树来说，Ω_j 包含的数据节点数量有 $H(n-1)$ 个，其中 $H=\log_n^N$，$Hn=K\log_n^N$，故 Ω_j 空间复杂度为 $O(\log N)$。

对于 SLBT 树来说，在最坏的情况下即将设置的链表存储节点用完，即 $M=N/n$ 则 Ω_j' 包含的数据节点数量为：$(M-1)+(n-1)=N/n+n-2$，故 Ω'_j 的空间复杂度为 $O(N)$。

由此可见，随着数据量的递增，在 CSS 端响应产生证据时，SLBT 树所产生的辅助验证信息反而比 LBT 要多，故通信效率相对要低，TPA 验证所花费的时间更多。

5 结束语

随着越来越多的用户选择将数据移植到云存储空间中，云存储服务的安全成为保证用户资源对服务忠诚度的重要指标[11-13]。在数据存储安全中数据完整性的验证又是其中必不可少的部分[14]。如何确保云存储环境下用户数据的完整性成为进近来学术界研究热点，数据完整性证明则是解决这一问题的有效手段[15]。在文献[8]的研究基础上，提出基于链表多分支路径树的完整性验证机制，将单一的叶子节点改进为链表节点，使其更有利于插入和删除数据块，减少动态数据结构重构时间，增加了数据的空间存储效率。但在验证时，CSS 端所花费的通信开销有所增长，同时该方案仍然和其他方案面临着同样的问题。即一旦更新数据，整个树结构将必不可少地面临着重构所带来的存储、计算和通信开销。另外，树结构均只有叶子节点存储着数据相关信息，不能从根本上解决重构问题。因此，是否可以尽可能减少中间节点的存在或者考虑全节点存储的情况等这类，所以动态树结构的重构问题仍需要作进一步的研究和改进。

参考文献：

[1] 胡德敏，余星. 一种基于同态标签的动态云存储数据完整性验证方法[J]. 计算机应用研究，2014，31(5)：1362-1365，1395.

[2] 林闯，苏文博. 云计算安全：架构、机制与模型评价[J]. 计算机学报，2013，36(9)：1766.

[3] Mell P，Grance T. The NIST definition of cloud computing. National Institute of Standards and Technology (NIST)[J]. Washington，USA：Technical Report Special Publication，2009，(1)：800-145.

[4] Ateniese G，Burns R，Curtmola R. Provable data possession at untrusted stores[C]. Proceedings of the 14th ACM conference on Computer and communications security(CCS2007). Alexandria USA，2007：598-609.

[5] WANG Qian，WANG Cong，LI Jin，et al. Enabling public verifinability and data dynamics for storage security in cloud computing [C]. Proceeding of 14th European Symposium on Research in Computer Security (ESORRICS 2009). Berlin，Germany：Springer-Verlag Press，2009：355-370.

[6] Merkle R C. Digital signature system and method based on a conventional encryptionfunction：U. S. Patent 4，881，264 [P]. 1989-11-14.

[7] 成鹤群，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.

[8] 李勇，姚戈. 基于多分支路径树的云存储数据完整性验证机制[J]. 清华大学学报(自然科学版)，2016，56(5)：504-510.

[9] 闫丽. 云计算环境下完整性验证方法研究[D]. 合肥：安徽大学，2016.

[10] 何冀. 基于云存储的动态数据公开审计[D]. 深圳：深圳大学，2015.

[11] Zhang X，Du H，Chen J，et al. Ensure data security in cloud storage[C]，Proceedings of IEEE 2011 International Conference on Network Computing and Information Security，Guilin，2011：284-287.

[12] AlZain M A，Pardede E，Soh B，et al. Cloud computing security：from single to multi-clouds [C]. Proceedings of IEEE 2012 45th Hawaii International Conferenceon System Science (HICSS)，Maui，HI，2012：5490-5499.

[13] Hwang J J，Chuang H K，Hsu Y C，et al. A business model for cloud computing based on a separate encryption and decryption service [C]. Proceedings of 2011 IEEE International Conference

on Information Science and Applications (ICISA), Jeju Island, 2011:1-7.

[14] 张亮. 云存储数据完整性检测技术研究[D]. 大连:大连理工大学,2014.

[15] 谭霜,贾焰等. 云存储中的数据完整性证明研究及进展[J]. 计算机学报,2015,38(1):164-170.

Data Integrity Verification Mechanism based on SLBT Tree

FANG Xin[1], FANG Rui[1], LIU Xue-tao[2] LIAO Yong[1], PU Dong[1], JIA Chuan[1]

(1. College of Cybersecurity Chengdu University of Information Technology, Chengdu 610225, China; 2. Yunnan Provincial Meteorological Observatory, Kunming 650042, China)

Abstract: With the emergence of low-cost, highly scalable cloud storage services, users transfer their data to the cloud server, this greatly saves users' expenses of data storage and later maintenance. At the same time, users have no absolute control of the data, which means that the user can not ensure whether their data stored in the cloud server is complete. On the basis of the present research, this paper proposes a multiple branch path tree based on the list (single linked list large branching tree, SLBT) integrity verification mechanism. In this mechanism, a single leaf node in a large branching tree is designed as a singly linked list, so it is more conducive to the insertion and deletion of data block. Analysis results show that this scheme not only supports data dynamic updating, but also improves the efficiency of data storage space and reduces the cost of refactoring dynamic data structure.

Keywords: cloud storage; data integrity verification; SBT tree; dynamic update