摘要

大数据一词反应了现在社会的趋势。据调查，到2020年每人每秒将产生1.7MB的数据，全球数据量将达到44万亿GB。在如此庞大的数据面前，如何存储变成了我们必须思考的问题，云存储就这样诞生了。

然而当我们放弃本地存储，选择更便捷的云存储的时候，是否考虑过数据的安全问题呢。在实际过程中、云服务并不一定是完全可信的，当用户面临数据泄露、服务器宕机等问题,怎样让用户确认云端数据存储不泄露，所以云存储服务器需要向用户提供完整性证明、还必须支持动态数据操作、同时必须提高通信、计算的性能。

本文针对以上问题：分析了两种数据完整性模型PDP和POR模型，并介绍了基于MHT的数据认证，并实现了基于跳表结构的认证。并给出了跳表结构的时间空间复杂度。本文的主要工作如下：

1. 介绍PDP和POR模型，分析了两种模型优缺点，并给出了支持动态数据操作和公开验证的关键信息。
2. 介绍了实现数据完整性认证的俩个关键的数据结构跳表和二叉树。并介绍了它们的相关操作，同时实现了跳表数据结构，并测试了这种数据结构的性能。
3. 介绍了基于MHT的数据认证，并实现了基于跳表结构的数据认证的两种算法，实现了基于跳表结构的数据认证，并给出了测试结果。

**关键词**：数据完整性；认证跳表；云存储；数据认证结构

Abstract

The term "big data" reflects the current trend of society. According to the survey, 1.7 MB per person per second will be generated by 2020, and the global data volume will reach 44 trillion GB. In the face of such a huge amount of data, how to store has become a problem we must think about, and cloud storage was born.

However, when we abandon local storage and choose more convenient cloud storage, have we considered the security of data? In the actual process, cloud service is not always completely trusted. When users face problems such as data leakage and server downtime, how to let users confirm that cloud data storage is not leaking? So cloud storage server needs to provide users with integrity proof, also must support dynamic data operation, and must improve the performance of communication and computing.

In this paper, two data integrity models, PDP and POR, are analyzed, data authentication based on MHT is introduced, and authentication based on jump table structure is realized. The time and space complexity of the jump table structure is also given. The main work of this paper is as follows:

1. The PDP and POR models are introduced, their advantages and disadvantages are analyzed, and the key information supporting dynamic data operation and public validation is given.

2. Two key data structure jump tables and binary trees for data integrity authentication are introduced. At the same time, the data structure of jump table is realized, and the performance of the data structure is tested.

3. Introduces data authentication based on MHT, and implements two algorithms of data authentication based on jump table structure.

The data authentication based on jump table structure is realized, and the test results are given.

**Keywords**: data integrity；authentication jump table；cloud storage； data authentication structure

**目录**

[摘要 1](#_Toc5201393)

[Abstract 2](#_Toc5201394)

[目录 4](#_Toc5201395)

[一、课题介绍 6](#_Toc5201396)

[1.1课题背景 6](#_Toc5201397)

[1.2课题意义 6](#_Toc5201398)

[1.3国内外研究现状 6](#_Toc5201399)

[1.4本文的主要工作 7](#_Toc5201400)

[二、相关基础知识 8](#_Toc5201401)

[2.1困难问题 8](#_Toc5201402)

[2.1.1大整数因数分解问题 8](#_Toc5201403)

[2.1.2离散对数问题 8](#_Toc5201404)

[2.1.3椭圆曲线 8](#_Toc5201405)

[2.2认证 9](#_Toc5201406)

[2.2.1常见的攻击方式： 9](#_Toc5201407)

[2.2.2Kerberos认证系统 9](#_Toc5201408)

[2.3Hash函数 10](#_Toc5201409)

[2.3.1MD5 10](#_Toc5201410)

[2.3.2SHA-1 11](#_Toc5201411)

[2.4可证明安全 12](#_Toc5201412)

[三 云存储的完整性验证 13](#_Toc5201413)

[3.1数据完整性方案模型： 13](#_Toc5201414)

[3.1.1PDP模型 14](#_Toc5201415)

[3.1.2POR模型 15](#_Toc5201416)

[3.2公开验证 16](#_Toc5201417)

[3.3动态数据更新 17](#_Toc5201418)

[四 跳表和Merkle Tree 17](#_Toc5201419)

[4.1跳表 17](#_Toc5201420)

[4.1.1跳表的基本原理 18](#_Toc5201421)

[4.1.2跳表的相关操作： 18](#_Toc5201422)

[4.1.3跳表的性能 19](#_Toc5201423)

[4.2Merkle Tree 20](#_Toc5201424)

[4.2.1Merkle Tree的相关知识 20](#_Toc5201425)

[4.2.2Merkle Tree的相关操作： 21](#_Toc5201426)

[五 基于跳表和Merkle Tree结构的认证 21](#_Toc5201427)

[5.1基于跳表的认证原理和算法 22](#_Toc5201428)

[5.2基于Merkle Tree的认证原理 23](#_Toc5201429)

[5.3基于跳表的认证实现 23](#_Toc5201430)

[参考文献 25](#_Toc5201431)

**一、课题介绍**

1.1课题背景

密码学诞生于20世纪70年代，在此之后形成一门学科，这受益于计算机的发展[4]。计算机为加密技术提供了新的工具，也让破译者更容易暴力破解。

1949年“保密系统的通信理论”的著名论文问世，将信息引入了密码，激发学者研究密码学的兴趣，奠定了密码学的理论基础。密码学除了信息的保密之外，还有认证即抵御对手的主动攻击，这在近几年体现的淋漓尽致。

近年来，网络和信息的覆盖越来越广，数据产生已成指数增长趋势。为了存储数据，企业也不得不购买更多的存储设备。这样不仅导致资金的浪费也带来了管理难度。

云存储的概念就诞生了，并逐渐代替了传统的存储方式。各大IT商提出了自己的云服务，比如阿里云、华为云、百度网盘等等。云服务器宕机、物理机损坏、数据删除等事件的发生，云存储上的数据完整性立刻成为企业关注的要点。

1.2课题意义

云存储的安全面临的问题：

1. 黑客攻击
2. 物理故障
3. 数据损坏不告知用户

云服务器提供的服务不一定是完全可靠的，所以必须提供一个可靠的数据完整性验证的方法。必须满足以下几点：

1. 能公开认证，用户不能实时在线，需要第三方进行周期性检验
2. 可以进行无数次审
3. 防止数据泄露，因第三方参与，必须保证第三方不能知道数据信息。
4. 动态数据操作

1.3国内外研究现状

Boneh等人提出BSL签名算法。在同等安全BSL签名机制比RSA签名的位数短，支持公开认证，能节省更多的存储空间，减少计算消耗。

Wang等人利用Merkle Tree这个数据结构构造了一个数据认证算法，这个算法能保护用户的数据不被泄露。并且支持动态认证，但每次都需要返回大量的认证路径，导致计算效率不高。

在Goodrich等人提出跳表[1]的基础上提出了认证跳表，利用跳表中节点存储数据的hash值，并通过相应的哈希算法实现数据的可信认证，此算法每次只支持一个数据源的动态操作。

姚等人对此进行改进，提出来LBT算法。此算法弥补了认证跳表不支持批量操作的缺点，但对于大数据操作时，有可能导致服务器超负荷运转。

1.4本文的主要工作

第一章中：主要介绍了数据认证的意义，以及在云存储中应用，同时谈到企业对于云存储的担忧。并介绍了国内外对云存储数据认证的贡献，以及他们提出算法的不足。

第二章中：主要介绍了认证的一些相关知识。首先介绍了密码学基础，重点介绍了认证模型和Hash算法。

第三章中：主要介绍了云存储中数据认证的两种协议PDP和POR，同时给出了动态数据更新和公开验证的关键技术。

第四章中：主要介绍了云存储中的两种常用的数据结构跳表和Merkle Tree，并给出了它们的主要操作。

第五章中：主要介绍了基于跳表和Merkle Tree的数据认证原理，以认证跳表为例给出其Java实现代码。

总结本文的主要成果，以及今后的规划。

二、相关基础知识

随着网络的发展，数据传输的速度也越来越快。密码学的重要性也越来越重要，消息认证是

密码学的一个重要内容。本章详细的对密码学级密码学中消息认证的基础知识做出详细的介绍。具体包括：困难问题、椭圆曲线、认证、Hash函数、以及典型的Hash算法SHA和MD5

2.1困难问题

困难问题是密码学的基础，所有的加密算法和认证算法必须依附于困难问题，即没有困难问题就没有密码学。想构造一个密码学算法，必须找到一个与之相对的困难问题。

2.1.1大整数因数分解问题

给定两个素数p，q，计算比较容易；

给定大整数n，求n的素因数p，q使非常困难[10]。

虽然说大整数因数分解是一个困难问题，但随着计算机的硬件发展，这个困难问题也越来越容易破解。依据这个问题构造的RSA算法现在512bits和1024bits相继与1999年和2009年被破解。现在要保证RSA算法的安全n大约要去2048bits。

2.1.2离散对数问题

已知有限循环群及其生成元g和阶。

给定整数a，计算元素很容易；

给定元素h，计算整数x，0≤x≤n,使得非常困难。

目前，有限域上的离散对数问题仍没有找到一种方法能有效破解。它的难易程度和生成元的取值无关，只和群中的元素数量有关。一般来说当k取值很小的时候，我们可以利用计算机穷举得到结果，现在的k一般取为1024bits。

2.1.3椭圆曲线

椭圆曲线是在射影平面上满足方程

（其中a、b、c、d、e为常数）上所有点的集合。

在诸多困难问题中，椭圆曲线上的离散对数问题要比其他问题难解的多，据研究，160位ECC加密安全性相当于1024位RSA加密，210位ECC加密安全性相当于2048位RSA加密[3]。

椭圆曲线密码体制是比较安全且密钥长度较短的公钥密码体制

2.2认证

认=检查、确认。证=身份、证明。认证其实就是为了确认身份、检查权限。

认证包括身份认证和消息认证

身份认证是指验证信息的发送者是否为真，包括信源、信宿等。

消息认证是指验证信息的完整信，验证数据在存储或传输过程中未被篡改、重放等

2.2.1常见的攻击方式：

1. 重放攻击：偷听者记录当前的消息，在适当的时间再次发送。

对策：增加时间戳

1. 字典攻击：利用常用的口令（例如123456、password等）对消息进行破解。

对策：在口令里面增加额外信息

2.2.2Kerberos认证系统

认证系统是防止消息被篡改、重放等攻击的一种有效方式，确保接受者确认消息的真伪。

Kerberos是身份验证协议、是Windows 2000的默认身份验证服务，是基于tcp/ip网络协议设计的可信协议。

步骤如下:

1. client(客户端)发送请求到验证服务器（AS），验证服务器创建一个“session”（根据你的请求），并产生一个代表服务器的随机值。
2. client将这个”session”发送给授权服务器（TGS），TGS会确认”session”，并返回一个消息，并打上时间戳。
3. client向消息服务器发送TGS返回的消息，选择接受或拒绝，并执行相应的服务。

2.3Hash函数

Hash函数，即散列函数，音译为哈希函数，目的是将任意长的消息压缩为一个固定长度的摘要。

Hash的特点：

1. 输入域中的元素是无穷的
2. 输出域中的元素是有限的
3. 当输入发生微小变化时，输出会发生巨大变化
4. 对于任意输入x计算H(x)是比较容易的，知道H(x)，寻找x是比较困难的

Hash构造方法

1. 直接寻址法

2. 数字分析法

3. 平方取中法

4. 折叠法

5. 随机数法

6. 除留余数法

2.3.1MD5

MD5 (message-digest algorithm 5)，经由MD2、MD3和MD4发展而来。主要用途：1防止信息被篡改、2防止直接看到明文、3防止抵赖。

MD5原理

**1、数据填充**

对消息进行数据填充，设消息长度为a，a满足(a+x) mod 512=448。可以算出需要填充长度x。最后使得消息长度为512的整数倍[11]。

**2、数据初始化**

准备需要用到的数据：

4个常数： A = 0x67452301, B = 0x0EFCDAB89, C = 0x98BADCFE, D = 0x10325476;

4个函数：F(X,Y,Z)=(X & Y) | ((~X) & Z); G(X,Y,Z)=(X & Z) | (Y & (~Z));  H(X,Y,Z)=X ^ Y ^ Z; I(X,Y,Z)=Y ^ (X | (~Z));

四种常用操作（)）

1. 其中a=b+((a+F(b,c,d)++)<<<s)
2. 其中a=b+((a+G(b,c,d)++)<<<s)
3. 其中a=b+((a+H(b,c,d)++)<<<s)
4. 其中a=b+((a+I(b,c,d)++)<<<s)

**3、数据分组**

把最终消息以512位为一组，分成若干组，把这512位分成16快32位

每种操作进行16次，得到最后一组a，b，c，d。

每组操作完成之后A=a+A，B=b+B，C=c+C，D=d+D，进行下组计算，最后将ABCD级联

输出的结果就是MD5码。

2.3.2SHA-1

SHA-1是由美国标准技术局（NIST）颁布的国家标准，这个算法也是有MD4发展而来。它和MD5的原理基本一样，这里我就只说MD5和SHA的区别。

SHA-1 与 MD5 的比较

l 对强行攻击的安全性：SHA-1得到的结果比MD5码长32 位。使用强行技术， MD5是

量级的操作，而对SHA-1则是量级的操作。这样，SHA-1能保证更大的安全强度。

2 速度：在相同的硬件上，因为SHA-1的循环次数多且缓存区大（160bits：128bits），所以SHA-1 的运行速度比 MD5 慢

2.4可证明安全

一个密码学协议的提出，在没有可证明安全的概念以前，只能经过时间才能得出是否安全。

可证明安全是一种规约方式。如果没有敌手能实现其目标则证明该协议是安全的。

过程为：

描述协议

描述敌手的目标（越低则方案越安全）

描述敌手的能力（越强则方案越安全）

构造安全模型（利用困难问题构造）

归约证明（归约到困难问题不可解上）

例子RSA签名

敌手的目标：

Total Break（完全攻破）

Universal Forgery（对任意签名伪造）

Selective Forgery（可选择对签名伪造）

Existential Forgery（对存在的签名伪造）

Strong Existential Forgery（对多次存在的签名伪造）

敌手的能力：

No Message（没有信息）

Known Message（知道信息）

Generic Chosen Message（知道通用信息）

Directed Chosen Message（知道定向信息）

Adaptive Chosen Message（可以选择性知道信息）

RSA加密

敌手的目标：

Total Break（完全攻破）

Decrypting a given Ciphertext（解密给定密文）

Obtaining useful information from Ciphertext（从密文中获取信息）

敌手的能力：

Key Only Attack（唯密钥攻击）

Chosen Plaintext（选择明文攻击）

Generic Chosen Ciphertext（选择密文攻击）

Adaptive Chosen Ciphertext 1（自适应的选择密文攻击）

Adaptive Chosen Ciphertext 2（自适应的选择密文攻击2）

在Adaptive Chosen Message情况下，可以抵抗Existential Forgery。说明此协议是安全的

三 云存储的完整性验证

云存储作为一种线上的存储方式，我们肯定会怀疑存储的数据是否安全，是否被别人修改，数据认证成为了云存储中的一个重点研究对象。本章简述了云存储数据完整性方案模型。

3.1数据完整性方案模型：

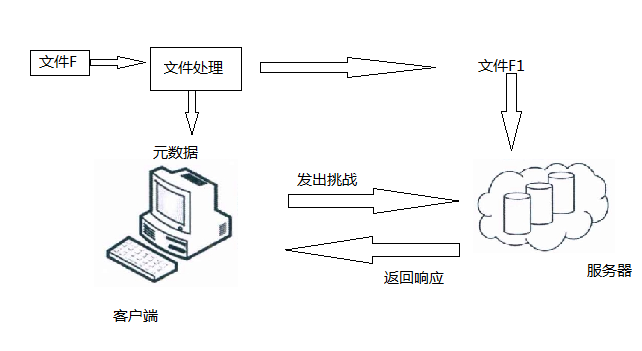
在现有的数据完整性认证模型中：PDP和POR是最典型两种模型。这两种模型的出现为以后的数据认证模型奠定了基础。

PDP模型高效，因为它不需要对数据进行加密。

POR可以恢复以损坏的文件。

3.1.1PDP模型

Ateniese等人在2007年提出了PDP模型，它是一个基于概率统计的一个模型，是采用抽样的形式，通过一个或几个子文件来判断整个文件的完整性。客户端在做预处理时，利用少量的元数据去向服务器发起挑战。



图（1）PDP算法

PDP包括以下算法

KeyGen() ->(pk,sk):客户端生成公钥pk=（N,g）和私钥sk=（e，d，v）其中e是事先说明的大数e>且d>，v，输出pk，sk。

TagBlock(pk,sk,m,i) ->:客户端利用pk和sk生成，。输出，称为元数据的标签。

GenProof(pk,F,chal,∑) –>V:服务端利用输入的pk和chal，遍历所有的元数据，得到V=(T,)

其中T=，H(g mod N),输出V；

CheckProof(pk,sk,chal,V)->{success,failure}:客户端利用pk和sk和chal生成，比较V和是否相等，若相等则输出success，否则failure。

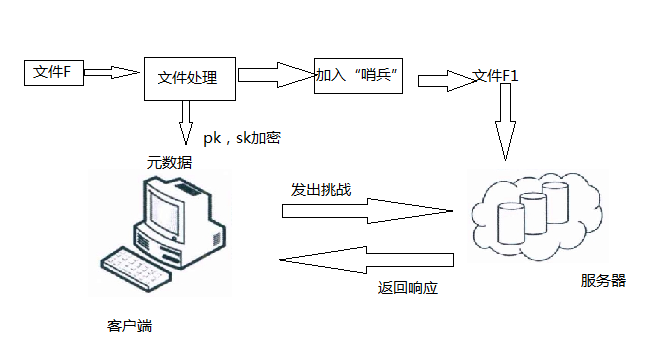
我们生成PDP协议和规范分为两步：setup和challenge。

setup：客户端运行KeyGen()生成公钥pk=（N,g）和私钥sk=（e，d，v）并在客户端上存储。将原始文件分块，再利用TagBlock(pk,sk,m,i)生成，并将原始文件和集合{}一同发送个服务器。

Challenge：客户端以n为单位按周期向服务器发起挑战，即在原始文件中随机抽取c个样本组成chal集合，并chal请求发送个服务器。服务器利用GenProof(pk,F,chal,∑)生成证明V，并返回客户端。收到V，CheckProof(pk,sk,chal,V) 来确定原始文件是否完整。

3.1.2POR模型

Juels等人于同年提出了POR模型的概念。POR和PDP一样都可以保证数据的完整性，但是POR模型设置了客户端发送挑战的次数的最大值，导致了POR模型只能支持静态数据验证和恢复。



图（2）POR算法

POR包括以下算法

KeyGen()->K

Encode(F;K,) ()->()

Extract(;K,) ()->F

Challenge(;K,) ()->c

Respond(c,)->r

Verify((r,); K,)->b

POR算法和PDP算法执行基本一致，只有

Encode(F;K,) ()->()算法中对文件进行了加密，并在文件中加入”哨兵”数据和原始数据混合生成新的数据。

Extract(;K,) ()->F算法是通过正确的验证来返回原始数据，由于”哨兵”的产生是随机的有限的，每次数据恢复就会返回一个”哨兵”，从而导致只能做有限次验证。

具体的POR算法请参考文献。

3.2公开验证

在本地存储的文件，我们一般通过访问全部数据来验证完整性，如果在云存储中也采用同样的方式进行数据完整性的校验，会耗费计算机大量的资源。即使不考虑计算机的性能问题，我们也不能每时每刻的让客户端保持连接状态。因此，我们把数据完整性校验委托给第三方来代替我们进行验证工作。

在公开验证中，引入了一个第三审计员，在客户端与服务端建立可信机制，所以又把这种验证方式称为数据审计。

公开验证的关键：

在PDP和POR协议中，只有客户端拥有原始数据的私钥sk，其他人无法进行验证。同时这两种算法的客户端计算复杂度、存储开销、通信开销都比较高。如果提高验证效率和扩展算法得以支持公开验证成了学者关注的主题。现在的方案大多都是基于同态验证标签（HVTs）来实现的。

同态验证标签具有两个基本性质：

无数据块验证：只需要提供原始数据由标签构造的证据，就能进行数据审计

同态性：已知数据块m，n它们分别对应的同态标签，。数据块f=m||n。

可以通过，来表示数据块f的同态标签。

3.3动态数据更新

实际上,当公司用到云存储来保存文件，这些文件的数量是很庞大的，并且需要随时随地的进行数更新，因此云存储应该支持动态的数据更新。当信息变化时，我们也应该保证这些数据的完整。由于网络资源有限、下载整个文件然后更新，显然是不现实的，所以必须找到一个合理的算法保证云存储服务器进行了数据更新。

在PDP协议的基础上增加三个算法

PrepareUpdate(pk,sk,data)->info:由客户端执行，输入公钥pk，私钥sk，以及要更新的信息data，输出请求信息info。（info包括增删改或查操作和数据信息）

PerformUpdate(info,F,)->{,,}:由服务器执行，输入info，文件F，元数据 ，输出新的文件,新的元数据,更新标识。

VerifyUpadte()->{accept,reject}:由客户端执行（如果是支持公开验证，可以委托给第三方代理执行），输入公钥pk和更新标志，如果验证通过则输出accept，否则更新失败输出reject。

四 跳表和Merkle Tree

跳表和Merkle Tree是两种常用的数据结构，在动态数据更新和数据认证中起着很大的作用。本章主要介绍跳表和二叉树这两种数据结构的原理，以及一些重要的节点操作。

4.1跳表

在云存储中我们经常要存储大量的数据文件。如果我们直接用二叉树或者链表用来记录文件节点，会影响我们的查询效率。

然而 在目前经常使用的平衡二叉树结构中，它是一种严格的平衡，虽然它的最差的查询时间一定是O(logn)，但它在插入方面需要旋转和overhead等操作。

跳跃表（Skip List）于1987年诞生，其查询 、插入和删除操作的预期时间复杂度为O（logn）。虽然跳表的最差查询时间是不稳定的，但跳表对插入更快，对并发的处理更友好。

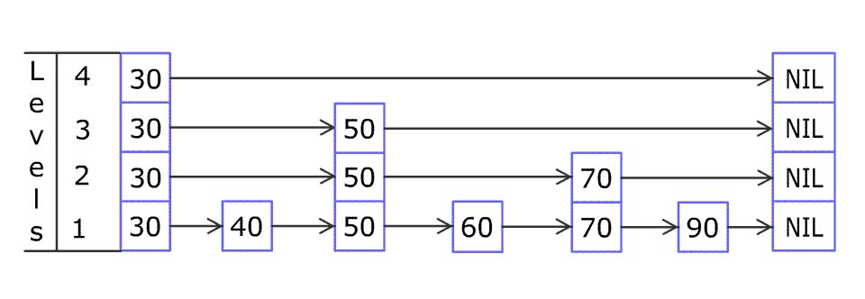
4.1.1跳表的基本原理

跳跃表由多条链构成（，， ……，），满足一下条件：

最底层链表包含所有的元素，链表中的元素升序

每个链表有两个特殊节点 -∞(起始节点)和+∞(终止节点)

S0包含所有元素，对任意的i>0， +1是的子集，且中的每一个节点，有一个下行指针，指向中与之有相同索引值的节点。

 图（三）跳表结构图

跳表的特点

1. 节点重排，以及顺序性的访问方式
2. 灵活的增删节点

4.1.2跳表的相关操作：

查询节点：（如果当前节点存在，则返回当前节点。如果不存在则返回离此节点最近的）

1. 从最上层的链（Sh）的开头开始
2. 假设当前位置为p，它向右指向的节点为q（p与q不一定相邻），且q的值为y。将y与x作比较

(1) x>y 从p向右移动到q的位置

(2) x=<y 从p向下移动一格

3. 当p在底链（S0）中且x=<y 输出当前p节点

删除节点

1. 通过查询 判断节点是否存在

2. 存在 则将该节点所在整列从表中删除

3. 将多余的“空链”删除

增加节点

高度的生成方法

（1）level = 0，

（2）产生一个0到1的随机数r，如果r小于一个常数p（），

（3）则执行方案level=level+1且重新执行（2）（3）

（4）否则，输出level。

1. 通过查询 判断节点是否存在
2. 在当前节点p后增加高度为level的节点

4.1.3跳表的性能

空间复杂度： O(n) （期望）

相关操作的时间复杂度：

查找： O(logn) （期望）

插入： O(logn) （期望）

删除： O(logn) （期望）

空间复杂度分析：

令由高度的生成方法，每个元素插入到第i层（Si）的概率为，则在第i层插入的期望元素个数为n，跳跃表的元素期望个数为 ，当的数时，次数总和小于2n。

S = 

当

S = 2n

空间复杂度为O(n)

实际测试： 共执行了次 maxLevel=3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P | 平均插入时间 | 平均高度 | 总结点数 | 平均查找时间 |
|  | 0.53976 | 1.7503845 | 17503845 | 0.28027 |
|  | 0.46651 | 1.5754064 | 15754064 | 0.25341 |
|  | 0.60256 | 1.3146518 | 13146518 | 0.31451 |
|  | 0.78464 | 1.1734221 | 11734221 | 0.35154 |

一般我们把p取，此时的平均查询时间最短，如果有要求空间的话，我们也可以退而求其次取p=。这样节点就不会占据大量空间。

4.2Merkle Tree

密码学家 Merkle提出了可信树的思想，并把这种结构以自己的名字命名。

Merkle Tree是一种二叉树。节点存储的数据的Hash值，所以又有人把它称作Hash Tree。

4.2.1Merkle Tree的相关知识

Tree的相关定义

结点：树的结点包含一个数据元素以及若干指向其子树的分支

深度：树中结点的最大层次

高度：从叶子结点开始定义，叶子结点为第一层，往上依次递增，直至根结点。

度：结点所拥有的子树数量(二叉树的度最大为2)

终端节点：度为0的结点称为叶子结点

Merkle Tree的遍历和二叉树基本一致

**先序遍历**

即遍历顺序为“根—>左->右”。

**中序遍历**

即遍历顺序为“左—>根—>右”，

**后序遍历**

即遍历顺序为“左—>右—>根”。

4.2.2Merkle Tree的相关操作：

查询（树不为空）：

1. 要查找的value与当前节点curvalue比较

2. value<curvalue:当前节点移动到左子树，并重新执行1

value>curvalue:当前节点移动到右子树，并重新执行1

value=curvalue：return 当前节点

删除

执行查询，如果不存在，则返回, 否则要删除的结点可能分下面四种情况：  
a. 要删除的结点无孩子结点：直接删除该结点。  
b. 要删除的结点只有左孩子结点：删除该结点且使被删除节点的双亲结点指向 被删除节点的左孩子结点  
c. 要删除的结点只有右孩子结点：删除该结点且使被删除节点的双亲结点指向被删除结点的右孩子结点  
d. 要删除的结点有左、右孩子结点：在它的右子树中寻找中序下的第一个结点，用它的值填补到被删除节点中

插入

1. 树若为空，直接插入
2. 树不为空，执行查找，找到终端节点
3. 插入新节点

五 基于跳表和Merkle Tree结构的认证

上章介绍跳表和Merkle Tree这两种数据结构。它们比一般的数据验证算法更适用于云存储的数据验证，它们可以有效的实现对数据的增删查改。本章介绍基于跳表和Merkle Tree的数据认证的原理，并以跳表数据结构为例给出基于跳表认证的实现。

5.1基于跳表的认证原理和算法

对算法中出现的名词的定义：

路径栈：在遍历中经过的节点，并且用栈进行保存。本文用skipNodeStack

来表示。

节点层数：跳表的当前层数。用level表示

isNextNode：查询的节点值是否是大于右节点的值

isPreNode：查询的节点值是否是大于左节点的值

Roberto提出的认证跳表算法有两步，首先获取辅助信息即节点的遍历路径和节点的Hash值返回给客户端。然后对节点的Hash值和辅助信息进行比较，验证是否通过。

遍历的路径栈具体步骤如下：

1. 如果当前节点的右节点不为空，则移动到右节点。
2. 如果当前节点处在level=0的地方则将该节点放入isNextNode集合中，否则将该节点的下层节点添加到isNextNode集合中。然后向上遍历，直至到当前节点的最高层。
3. 判断节点的最高层的右节点是否为空，若空则停止计算，否则重1开始。

节点的验证：其实就是用isNextNode和isPreNode来检验遍历的路径栈是否正确。

节点的验证过程步骤如下：

1. 如果isNextNode不为空：

如果当前节点不是最高层，则下个节点处在索引为i的level+1层

如果是最高层，则下个节点处在索引为i+的level层

1. 如果isNextNode为空：

如果isPreNode不为空，则下个节点处在索引为i的level+1层

如果isPreNode为空，则下个节点就是节点

Hash值的计算：

1. 若isPreNode不为空：

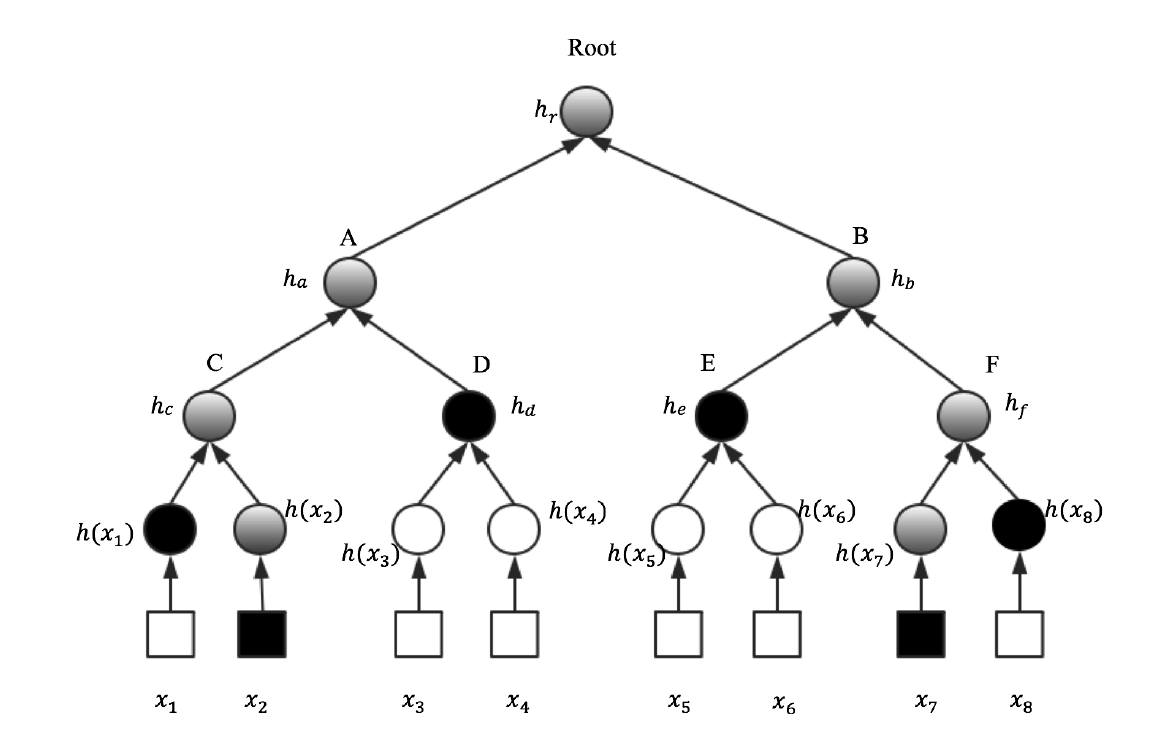
Hash(node)=Hash(currentNode)||Hash(preNode)；

2. 若isPreNode为空

Hash(node)= Hash(currentNode)

最后将Hash值和路径栈一并返回，客户端进行验证，若一致则成功，否则失败。

5.2基于Merkle Tree的认证原理



图（四） Merkle Tree图

**执行查询找到目标节点，并构造认证路径，利用叶子节点的Hsah值计算出根节点的Hash值，最后根节点与客户端的Hash值进行比较。**

**假定现在有8个数据源{}，我们现在要找3,7的。3的路径是root->a-d-3**

**7的路径是root-b-f-7**

Hash值计算将子节点的Hash值进行或运算直到root节点为止

具体的算法见[]。

5.3基于跳表的认证实现

开发环境是eclipse+jkd1.8

认证跳表的实现过程：

准备算法：

Hash函数——MD5码

定义四个初始变量：

**static** **final** **long** ***A***=0x67452301L;

**static** **final** **long** ***B***=0xefcdab89L;

**static** **final** **long** ***C***=0x98badcfeL;

**static** **final** **long** ***D***=0x10325476L;

一个初始函数:

g(i,x,y,z);其中g(0,x,y,z)=F(x,y,z)=(x&y)|((~x)&z);

g(1,x,y,z)=G(x,y,z)=(x&z)|((~z)&y);g(2,x,y,z)=H(x,y,z)=x^y^z;

g(3,x,y,z)=I(x,y,z)=y^(x|(~z));

移位数组s = {{7,12,17,22},{5,9,14,20},{4,11,16,23},{6,10,15,21}};

结果集result={A,B,C,D}

四种常用操作（)）

1. 其中a=b+((a+F(b,c,d)++)<<<s)
2. 其中a=b+((a+G(b,c,d)++)<<<s)
3. 其中a=b+((a+H(b,c,d)++)<<<s)
4. 其中a=b+((a+I(b,c,d)++)<<<s)

具体算法

数据填充fileFunction(String message)：

1：将message转化为字节数组byte，然后利用字节数组byte的长度算出需要填充的字节数x。

2：添加1和若干0直到长度等于x，计算此时的在计算message的二进制的低64位，此时再将此填充到byte数组中。

3：将处理好的byte数组按512长度分为若干组，并把每个组按64字节转化为long型，记每一组为。

数据处理dataHanding(long[] data)：

输入长度为16的,对其进行4轮操作。第一轮进行16次

第二轮进行16次…。

其中是代表第数据块的16进制数。

最后将16进制数转化为字符串进行拼接。

其中关键算法dataHanding(long[] data)的代码如下：

a = result[0], b = result[1], c = result[2], d = result[3]

for i=0:3 step 1

for j=0:15 step 1

result[0] += g(i,result[1], result[2], result[3])

+data[K[i][j]]+)

result[0] = result[1]

+ (result[0]<< ***S***[n][i] | (result[0] >>> (32 - ***S***[n][i])));

end

change(result[0],result[1], result[2] result[3])

end

result[0] += a;

result[1] += b;

result[2] += c;

result[3] += d;

测试：

分别对”hello”和”认证”两个字符串进行加密：

java程序：

hello：5D41402ABC4B2A76B9719D911017C592

认证: B7158A42CA1C2845B3C08111982B0494

在线MD5码加密（[https://md5jiami.51240.com](https://md5jiami.51240.com/)）结果：

hello：5D41402ABC4B2A76B9719D911017C592

认证: B7158A42CA1C2845B3C08111982B494

认证跳表实现

建立跳表结构：

跳表节点的结构如下

{

Long key;//节点的key

Object value;//节点原始的Hash值

SkipListNode[] forword;//用于存储前节点

SkipListNode[] back;//用于存储后节点

}

实现跳表的常用操作：算法见第四章

init();//初始化跳表

add\_Node();//增加节点

find();

delete\_Node();

change\_Node();

然后将跳表加工成认证跳表,

认证路径和Hash值的计算：

node = find(key);

initValue = node.value

while node.value – MAX !=0

node = node.next;

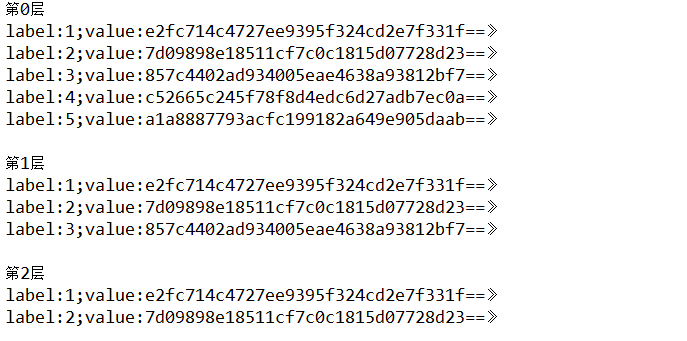
hash = culHash(initValue,node.value);//计算Hash值或运算

nodeArray.add(node);//认证路径的数组

end

return nodeArray hash

我将X:\\hash下的文件利用MD5消息摘要算出Hash值。跳表结构如下



图（5） 跳表结构的Java结果

每个节点存储的Hash值分别对应的是5个文件。任意文件的修改都会改变level=0的节点的Hash值从而影响到最后生成的Hsah。

我都修改key=3所对应的文件：

|  |  |
| --- | --- |
| key=1的Hsah值 | 修改前fffefdffffffffbf95f324cd2e7f331f |
| 修改后e7fefdffdfffffff95f324cd2e7f331f |
| key=2的Hsah值 | 修改前fdfefdf7ffffffbddb8f609c4f667f02 |
| 修改后e7fefdffdffffffdb55ae004c240ae60 |
| key=3的Hsah值 | 修改前e5feedf7ffffff9deae4638a93812bf7 |
| 修改后e5fefdffdffffffd56cc59b19ed215d2 |
| key=4的Hsah值 | 修改前e5aeedf7d7ffff9d4edc6d27adb7ec0a |
| 修改后e5aeedf7d7ffff9d4edc6d27adb7ec0a |
| key=5的Hsah值 | 修改前a1a8887793acfc199182a649e905daab |
| 修改后a1a8887793acfc199182a649e905daab |

从表格中可以看出：认证跳表结构确实能检验出数据的完整性，但是当我改变key=3所对应的文件是，key=4的认证路径没有经过被修改的文件，所以它认为此数据源没有被修改。如果想要确认整个文件的完整性，则必须从最小的节点开始，但是这要就要耗费比前一个节点更高的计算消耗。

参考文献

1. 云存储中数据完整性验证技术研究 罗昊-《电子科技大学硕士论文》-2017
2. 云数据完整性验证的关键技术研究    邱佳惠-《电子科技大学硕士论文》-2017
3. 椭圆曲线密码算法的能量分析    周文锦-《电子科技大学硕士论文》-2006
4. 分组密码算法实现效率研究 黄橙-《四川大学硕士论文》-2005
5. 椭圆曲线密码算法的能量分析    周文锦-《电子科技大学硕士论文》-2006
6. 密码算法的研究综述 贾宁-《现代电子技术》-2007
7. 数字签名技术及其在社会保险网络信息系统中的应用研究    赖永东-《广西大学硕士论文》-2008
8. 密码杂凑函数及其安全性分析    关振胜-《信息网络安全》-2012
9. 可撤销匿名凭证系统的研究与实现    -《互联网论文库》-2015
10. ECC多门限动态签密方案的研究与设计 范畅-《贵州大学硕士论文》-2007
11. 杂凑函数的攻击方法研究 陈士伟-《解放军信息工程大学博士论文》-2012

致谢

四年大学时光匆匆已过，我的学校生活也即将结束。我要感谢我的导师刘光军，他不仅在我的毕业设计帮助我，同时在指导我在大学数学建模，并在全国大学生数学建模取得优异的成绩。刘光军老师有丰富的专业知识，在密码学这门课中我了解了当前信息保密的重要性。

其次，我还要感谢我大学的其他老师，包括张力宁老师、海小娟老师等，他们不仅在学习上给予我帮助，不管在什么时候只要我有问题，他们总能在第一时间给出答案。

最后，还要感谢和我共同度过的同学和舍友，他们和我一同学习，一同进步。因为他们的陪伴，让我的大学生活不再枯燥。虽然在四年中，我在家的时间没有学校长，但是父母总是能在第一时间关系我，是他们给了我大学生活的经济来源，让我全心全意在学习上。

最后，再次感谢全体关心我的人。