摘要课题介绍

## 课题背景

## 研究意义

## 研究内容

# 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

## 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。

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。

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。

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。

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。

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

## 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

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

### Kerberos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。

## Hash函数

  Hash，一般翻译做“散列”，也有直接音译为“哈希”的，就是把任意长度的输入，通过散列算法，变换成固定长度的输出，该输出就是散列值。  
 散列表（Hash table，也叫哈希表），是根据关键码值(Key value)而直接进行访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。

Hash构造方法

1. 直接寻址法：取关键字或关键字的某个线性函数值为散列地址。即H(key)=key或H(key) = a\*key + b，其中a和b为常数（这种散列函数叫做自身函数）

　　2. 数字分析法：分析一组数据，比如一组员工的出生年月日，这时我们发现出生年月日的前几位数字大体相同，这样的话，出现冲突的几率就会很大，但是我们发现年月日的后几位表示月份和具体日期的数字差别很大，如果用后面的数字来构成散列地址，则冲突的几率会明显降低。因此数字分析法就是找出数字的规律，尽可能利用这些数据来构造冲突几率较低的散列地址。

　　3. 平方取中法：取关键字平方后的中间几位作为散列地址。

　　4. 折叠法：将关键字分割成位数相同的几部分，最后一部分位数可以不同，然后取这几部分的叠加和（去除进位）作为散列地址。

　　5. 随机数法：选择一随机函数，取关键字的随机值作为散列地址，通常用于关键字长度不同的场合。

　　6. 除留余数法：取关键字被某个不大于散列表表长m的数p除后所得的余数为散列地址。即 H(key) = key MOD p, p<=m。不仅可以对关键字直接取模，也可在折叠、平方取中等运算之后取模。对p的选择很重要，一般取素数或m，若p选的不好，容易产生同义词。

MD5

MD5是message-digest algorithm 5（信息-摘要算法）的缩写，被广泛用于加密和解密技术上，它可以说是文件的“数字指纹”。任何一个文件，无论是可执行程序、图像文件、临时文件或者其他任何类型的文件，也不管它体积多大，都有且只有一个独一无二的MD5信息值，并且如果这个文件被修改过，它的MD5值也将随之改变。因此，我们可以通过对比同一文件的MD5值，来校验这个文件是否被“篡改”过

**1、数据填充**

对消息进行数据填充，使消息的长度对512取模得448，设消息长度为X，即满足X mod 512=448。根据此公式得出需要填充的数据长度。

填充方法：在消息后面进行填充，填充第一位为1，其余为0。

**2、添加消息长度**

在第一步结果之后再填充上原消息的长度，可用来进行的存储长度为64位。如果消息长度大于264，则只使用其低64位的值，即（消息长度 对 264取模）。

在此步骤进行完毕后，最终消息长度就是512的整数倍。

**3、数据处理**

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));

把消息分以512位为一分组进行处理，每一个分组进行4轮变换，以上面所说4个常数为起始变量进行计算，重新输出4个变量，以这4个变量再进行下一分组的运算，如果已经是最后一个分组，则这4个变量为最后的结果，即MD5值。

SHA1

SHA1该算法的思想是接收一段明文，然后以一种不可逆的方式将它转换成一段（通常更小）密文，也可以简单的理解为取一串输入码（称为预映射或信息），并把它们转化为长度较短、位数固定的输出序列即散列值（也称为信息摘要或信息认证代码）的过程。散列函数值可以说是对明文的一种“指纹”或是“摘要”，所以对散列值的数字签名就可以视为对此明文的数字签名。

安全散列算法SHA（Secure Hash Algorithm，SHA)是美国国家标准技术研究所发布的国家标准FIPS PUB 180，最新的标准已经于2008年更新到FIPS PUB 180-3。其中规定了SHA-1，SHA-224，SHA-256，SHA-384，和SHA-512这几种单向散列算法。SHA-1，SHA-224和SHA-256适用于长度不超过2^64二进制位的消息。SHA-384和SHA-512适用于长度不超过2^128二进制位的消息。

编辑

在SHA1算法中，我们必须把原始消息（字符串，文件等）转换成位字符串。

**数据填充**

对消息进行数据填充，使消息的长度对512取模得448，设消息长度为X，即满足X mod 512=448。根据此公式得出需要填充的数据长度。

填充方法：在消息后面进行填充，填充第一位为1，其余为0。

三 云存储中的数据认证

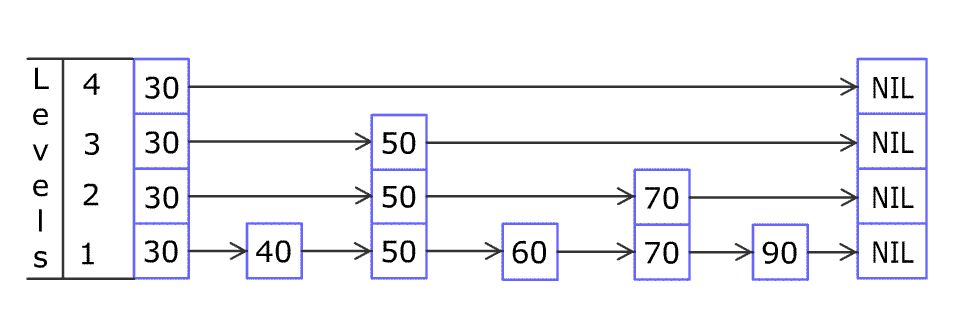
云存储作为一种线上的存储方式，我们肯定会怀疑存储的数据是否安全，是否被别人修改，数据认证成为了云存储中的一个重点研究对象。本章简述了云存储中数据认证的俩个重要的数据结构，以及典型的数据认证算法。

云存储中常用的两种数据结构

跳表

跳表全称跳跃表，是一个分层的链表结构，在链表原有的基础上增加了多层索引。一般来说，我们把最顶层最左边的节点称为起始节点。

跳表结构



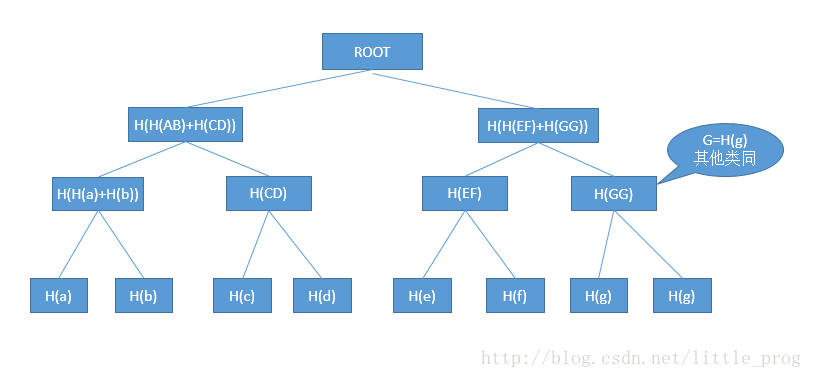
Merkle Tree

Hash Tree是基于Hash函数为基础构建的一个数据结构。是一种典型的以空间换取时间的算法之一

Merkle Tree是一种树，大多数是二叉树，也可以多叉树，无论是几叉树，它都具有树结构的所有特点；

Merkle Tree的叶子节点的value是数据集合的单元数据或者单元数据HASH。

非叶子节点的value是根据它下面所有的叶子节点值，然后按照Hash算法计算而得出的。



认证的设计模型

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 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。 [2-5] 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 。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。 1 2 (, , , )n S ee e = Λ ，其中 i e 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”； 930 电 子 科 技 大 学 学 报 第 43 卷 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。

云存储中典型的数据认证算法

1、基于MHT的数据完整性验证

默克尔哈希树是一种树型的数据结构，其叶子结点是可独立验证的数据块（或者间接数据）的Hash，任一内部结点是其孩子结点数据联接后计算得到的 hash 值。适用于快速验证某一数据块或少数数据块的完整性。通过节点i的值和snop算得根节点的hash值（snop：节点i与MHT根节点之间路径上每个节点的兄弟节点。在图中h3的snop为h4、n3、n2）则根节点hash值为root‘=hash(n2|hash(n3|hash(h3|h4)))，比较root’与原来的root节点的hash值即可。相对于传统的完整性检查，Merkel哈希树的时间复杂度大为降低，从0(n)降到为0(logn)。

2.基于跳表的数据完整性验证

跳表是从二叉树扩展出的一种数据结构，它是一种有序的多层次链表，如下图所示

在跳表中查找元素x，首先从左上角的头结点s开始，头结点s即为跳表中最左上角的节点，设v为查找到的当前节点，初始时，v=s，查找过程中只需要用到两个方式，右移和下移（跳表中的元素都是有序的），右移为在本行中向右查找直到找到第一个小于或者等于x的最大元素，下移也是同样的道理，那么在该图中查找元素14的过程如下：

本图中有S0-S4五个链，其中我们定义停滞元素为存在于Si-1而不存在与Si中的元素，高塔元素为即存在于Si-1又存在于Si中的元素，另外定义elem(v)为存在在节点v的元素，down(v)Si-1中节点v下面的结点；right(v)为Si中v结点右边的结点，跳表上的验证过程设计如下;

标签值f(v)的计算方法如下： 定义w=right(v),u=down(v),当right(v)=null时，定义f(v)=0。那么：

当u=null时，即v在So上：

（a）当w为高塔节点时，f(v) = h(elem(v),elem(w));

(b) 当 w 停滞节点时，f(v) = h(elem(v), f(w))。

当u不等于null时：

（a）当w为高塔节点， f(v) = f(u)；

（b）当w为停滞节点时，f(v) = h(f(u),f(w))

四 跳表

在云存储中，跳表和默克尔树是其中比较重要的数据结构，在数据认证中起着很大的作用。本章详细介绍了跳表这种数据结构，包括跳表的增删改查算法及其实现，同时还介绍了跳表的复杂度，更能直观的反应出跳表的优缺点。最后介绍了跳表的实际应用场景。

### 为什么选择跳表？

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。

## 跳表的基本原理

跳跃表由多条链构成（S0，S1，S2 ……，Sh），满足一下条件：

最底层链表S0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。

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

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

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。

跳表的实际应用：

应用场景：

不考虑空间，但要求查询时间快，和节点增加更新不是很频繁。

1. Redis

Redis能存储五种数据类型string、list、set、zset、Hash。其中zset结构使用跳跃表提高查询效率(以score为顺序的跳跃表)

认证跳表