摘要课题介绍

## 课题背景

## 研究意义

## 研究内容

# 消息认证基础

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、消息认证的基础、Hash函数、以及典型的Hash算法SHA和MD5

## 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。

例子：RSA算法是基于大整数因数分解问题提出的一种算法。分解的难度保证了RSA算法的可靠性，简单来说就是密钥长度决定了RSA算法的安全性。1999年，RSA-155（512bits）成功被破解。2009年RSA-768（1024bits）成功被破解。现在的RSA算法长度大多约为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。

ElGamal算法

椭圆曲线离散对数问题（diffie-hellman）

格问题

## 认证的基本概念

定义

消息认证是指通过对消息或者消息有关的信息进行加密或签名变换进行的认证，目的是为了防止传输和存储的消息被有意无意的篡改，包括消息内容认证、消息的源和宿认证、及消息的序号和操作时间认证等。

认证和加密的区别在于：

加密用以确保数据的保密性，阻止对手的被动攻击，如截取，窃听等；

而认证用以确保报文发送者和接收者的真实性以及报文的完整性，阻止对手的主动攻击，如冒充、篡改、重播等。

认证系统常用的参数有：

口令、标识符、密钥、信物、智能卡、指纹、视网纹等。

对消息内容的攻击方式：

伪造消息

篡改消息内容

改变消息顺序

消息重放和延迟

### 认证系统的模型

Hash函数

MD5

RSA

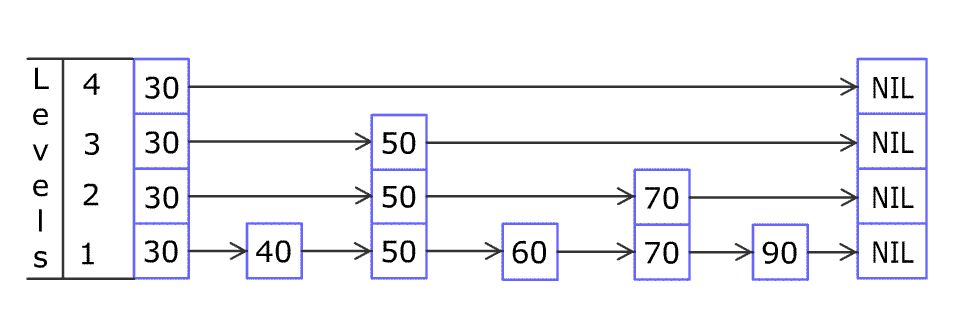
三 云存储中的数据认证

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



Hash 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

云计算中典型的数据认证算法

四 跳表

跳表产生的背景

平衡树可以用于表示抽象的数据类型如字典和有序链表，它通过树旋转（Tree Rotation）操作强制使树结构保持平衡来保证节点搜索的效率。在数据为随机插入的情况下，平衡树性能表现良好；但数据为顺序插入或者需要删除节点的情况下，平衡树的性能就会有些糟糕。

跳表可以作为平衡树的一种替代选择。它使用随机的平衡策略取代平衡树严格的强制的树平衡策略。因此它具有更简单有效的插入/删除方法以及更快的搜索速度。

跳表的是由链表构成

认证采用跳表这个数据结构是考虑到了很多因素，

1. 节点重排，以及顺序性的访问方式
2. 灵活的增删节点
3. 链表查询的效率低下

链表满足以上的前两的点，但由于查询的效率问题，从而对链表进行了改进。

单向链表越靠后末端的节点查询速度会越慢，双向越靠近中间会越慢。由于我们要在云存储上存储的文件越多，认证的结果就越有效率问题。

由于链表的查询效率的问题，对链表进行改进，从而衍生出了跳表。

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）

定义左边界为head 右边界为 tail

令startNode = head  
1. 当 startNode.value < node.value 时 startNode.forward

2. 直到 startNode.value>=node.value ,startNode.down  
3. 重复1，2，直到startNode.down==null

4 return startNode;

public SkipListNode find(Integer value) {

int i = level-1;

SkipListNode node = head;

while(i!=-1) {

while(node.getForword()[i].getValue()<=value) {

node = node.getForword()[i];

}

i--;

}

return node;

}

删除节点

1通过查询 判断节点是否存在

2 存在 则把currentNode的表示前一个节点的值，赋给下一个节点

**public** **void** delete\_NodeByValue(Integer value) {

**if**(!isExsit(value)) {

**return**;

}

SkipListNode node = find(value);

**for**(**int** i =0;i<3;i++) {

SkipListNode pre = node.getBack()[i];

SkipListNode next = node.getForword()[i];

pre.getForword()[i] = next;

next.getBack()[i] = pre;

}

}

增加节点

**private** **void** add\_Node(Integer value) {

**if**(isExsit(value)) {

**return**;

}

SkipListNode addNode = **new** SkipListNode(value,level);

SkipListNode node = find(value);

**for**(**int** i = 0;i<=level-1;i++) {

SkipListNode next = node.getForword()[i];

SkipListNode pre = next.getBack()[i];

addNode.getForword()[i] = next;

addNode.getBack()[i] = pre;

}

**for**(**int** i = 0;i<=level-1;i++) {

SkipListNode next = node.getForword()[i];

SkipListNode pre = next.getBack()[i];

**double** num = rand.nextDouble();

**if**(i==0) {

pre.getForword()[i] = addNode;

next.getBack()[i] = addNode;

}

**else** **if**(i>0&&num>0.5) {

pre.getForword()[i] = addNode;

next.getBack()[i] = addNode;

}

**else** {

**break**;

}

}

}

修改节点

**public** **void** change\_Value(Integer value1,Integer value2) {

**if**(!isExsit(value1)) {

**return**;

}

SkipListNode node = find(value1);

node.setValue(value2);

}

跳表的性能

跳表的特点：

(1) 每个跳表由很多层结构组成；

(2) 每一层都是一个有序链表，且第一个节点是头节点；

(3) 最底层的有序链表包含所有节点；

(4) 每个节点可能有多个指针，这与节点所包含的层数有关；

(5) 跳跃表的查找、插入、删除的时间复杂度均为O(log N)。

跳表的实际应用：

应用场景：  
节点增加和更新比较少，查询频次较多的情况。

1、Lucene, elasticSearch

2、Redis：

Redis sorted set的内部使用HashMap和跳跃表(SkipList)来保证数据的存储和有序，HashMap里放的是成员到score的映射，而跳跃表里存放的 是所有的成员，排序依据是HashMap里存的score,使用跳跃表的结构可以获得比较高的查找效率，并且在实现上比较简单。

认证跳表