数据结构与算法（Python）-08/第9周

北京大学 陈斌

2021.05.04

线下课堂

- 本周内容小结：排序查找（下）
- 问题解答
- MD5算法
- 完美散列函数的应用
- 【K08】课堂练习

W08：查找与排序（下）

- 508 什么是散列 7m21s
- 509 完美散列函数 15m02s
- 510 区块链技术 17m20s
- 511 散列函数设计 8m47s
- 512 冲突解决方案 11m59s
- 513 映射抽象数据类型及Python实现 14m58s
- 514 排序与查找小结 9m45s

508 什么是散列

散列：Hashing

- 能够使得查找的次数降低到**常数**级别，我们对数据项所处的位置就必须有更多的**先验知识**。
- 如果我们**事先**能知道要找的数据项**应该**出现在数据集中的什么**位置**，就可以直接到那个位置看看数据项是否存在即可。
- 由数据项的**值**来确定其存放位置，如何能做到这一点呢?

$$key = hash(data)$$

$$D[key] = data$$

```
if D[hash(data)]==data:
    return True
else:
    return False
```

509 完美散列函数

- 给定一组数据项，如果一个散列函数能把每个数据项映射到不同的槽中，那么这个散列函数就可以称为“**完美散列函数**”

  对于固定的一组数据，总是能想办法设计出完美散列函数

- 但如果数据项经常性的变动，很难有一个系统性的方法来设计对应的完美散列函数

- 退而求其次，好的散列函数需要具备特性

  冲突最少（近似完美）、计算难度低（额外开销小）、充分分散数据项（节约空间）

MD5(128bits), SHA系列

SHA-0/SHA-1输出散列值160位（20字节），
SHA-256/SHA-224分别输出256位、224位，
SHA-512/SHA-384分别输出512位和384位

完美散列函数用于数据一致性校验

密码加密、文件校验、防篡改

510 区块链技术

- **区块链（block chain）是一种分布式数据库**
  去中心化（decentralized）
  需要解决：不需要信任和权威，也可以防止篡改和破坏的问题
- **两个主要机制**
  用散列技术和链表技术实现了“牵一发动全身”的抗修改区块链
  用“工作量证明”的奖励机制实现了“互相竞争、互相监督”的共识体系
- **最大的区块链——比特币（bitcoin）**
- **应用越来越广泛，成为信息社会的基础设施**
  数字人民币

511 散列函数设计

- **散列函数设计原则**

  压缩度高、分散度高、计算量小

- **几个典型的算法**

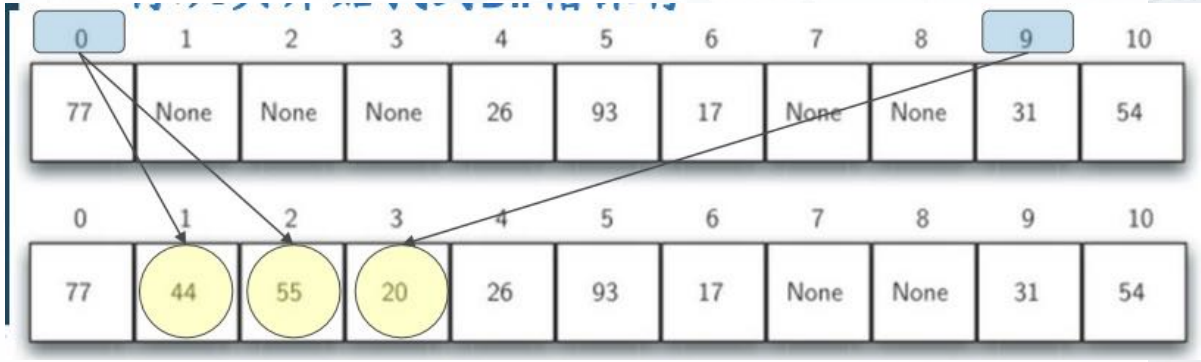
  折叠法、平方取中法、非数值编码转换法等等

512 冲突解决方案

- 如果两个数据项被散列映射到同一个槽，需要一个系统化的方法在散列表中保存第二个数据项，这个过程称为“解决冲突”
- 解决散列的一种方法就是为冲突的数据项再找一个开放的空槽来保存

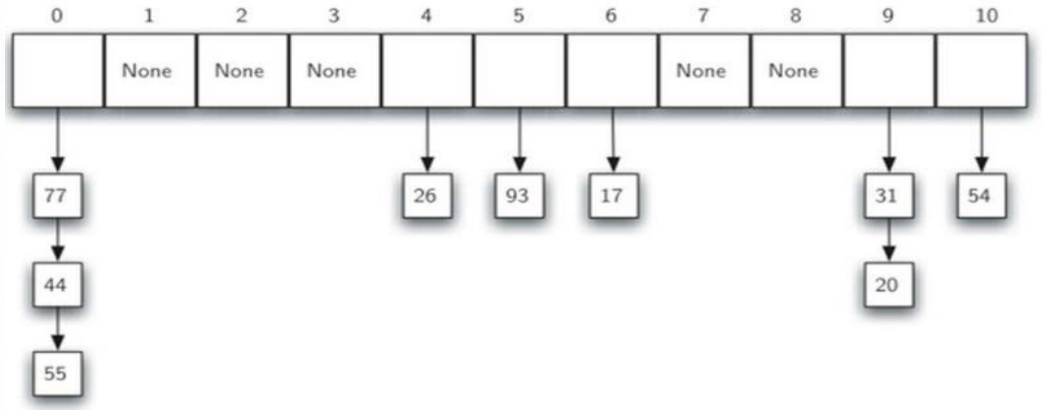
  最简单的就是从冲突的槽开始往后扫描，直到碰到一个空槽

  如果到散列表尾部还未找到，则从首部接着扫描
- 这种寻找空槽的技术称为“开放定址open addressing”
- 向后逐个槽寻找的方法则是开放定址技术中的“线性探测linear probing”



512 冲突解决方案

- 如果两个数据项被散列映射到同一个槽，需要一个系统化的方法在散列表中保存第二个数据项，这个过程称为“解决冲突”
- 除了寻找空槽的开放定址技术之外，另一种解决散列冲突的方案是将容纳单个数据项的槽扩展为容纳数据项集合（或者对数据项链表的引用）



513 映射抽象数据类型及Python实现

- **ADT Map定义的操作如下：**

  Map()：创建一个空映射，返回空映射对象；

  put(key，val)：将key-val关联对加入映射中，如果key已存在，将val替换旧关联值；

  get(key)：给定key，返回关联的数据值，如不存在，则返回None；

  del：通过del map[key]的语句形式删除key-val关联；

  len()：返回映射中key-val关联的数目；

  in：通过key in map的语句形式，返回key是否存在于关联中，布尔值

可以用各种技术来实现ADT Map
** 无序表：put/顺序查找的get
** 有序表：put/二分查找的get
** 散列表：put/内容关联的get
** 后续的二叉排序树……

514 排序与查找小结

算法的选择：一封同学来信

- 所以排序算法有时候并不存在绝对的优劣，尤其是时间复杂度相同的算法们
- 要在特定的应用场合取得最高排序性能的话，还需要对**数据本身**进行分析，针对数据的特性来选择相应排序算法
- 另外，除了时间复杂度，有时候空间复杂度也是需要考虑的关键因素

  归并排序时间复杂度O(nlog n)，但需要额外一倍的存储空间
- 算法选择不是一个绝对的优劣判断，需要综合考虑各方面的因素

  包括运行环境要求、处理数据对象的特性

问题解答

- 既然下标增加的二次探测法有oj这样的bug（即可能存在无法插入的情况），为什么还认为这是比线性探测更好的方法呢？
- 会出现周期，但能够到达1/2较为分散的槽。
- 主要的相对优势在于避免局部集中，而引起连锁的冲突。

```
NO, **2, 5, 17
00, 000, 0, 00
01, 001, 1, 01
02, 004, 4, 04
03, 009, 4, 09
04, 016, 1, 16
05, 025, 0, 08
06, 036, 1, 02
07, 049, 4, 15
08, 064, 4, 13
09, 081, 1, 13
10, 100, 0, 15
11, 121, 1, 02
```

问题解答

› **如何根据负载因子估计平均查找次数呢？例如mooc上面的选择题判断：“如果采用线性探测的开放定址法来解决冲突，负载因子0.8，成功的查找，平均需要比对次数约为3”，这是怎么得到的呢？**

散列算法分析

见视频513的最后，Canvas第9周下载论文PDF

- **如果采用线性探测的开放定址法来解决冲突（λ在0~1之间）**

  成功的查找，平均需要比对次数为：$\frac{1}{2}\left(1+\frac{1}{1-\lambda}\right)$

  不成功的查找，平均比对次数为：$\frac{1}{2}\left(1+\left(\frac{1}{1-\lambda}\right)^2\right)$

- **如果采用数据链来解决冲突（λ可大于1）**

  成功的查找，平均需要比对次数为：$1+\lambda/2$

  不成功的查找，平均比对次数为：λ

查找过程的信息

- **存取（access）：由位置存储/获取值**
- **查找（search）：由值确定其存储位置**
- **数据存储位置与其值无关：无序表**

  只有相等或不相等：顺序查找，时间复杂度O(n)
- **数据存储的相对位置与其值相关：有序表**

  如果还可以进行大小比较：按照大小排序后，二分查找，时间复杂度O(log n)
- **数据存储的绝对位置与其值相关：散列表**

  值和存储位置有函数关系：通过散列函数直接映射到位置，时间复杂度O(1)

排序过程中的比较信息

- **基于比较，一阶信息**

  ai, aj之间的大小比较，如ai<aj

  冒泡、选择、插入、谢尔排序

  时间复杂度在$O(n^2)$级别

- **基于比较，二阶信息**

  ai, aj以及ai, ak之间大小的信息，利用ai, ak之间的隐含大小

  快速排序、归并排序

  时间复杂度$O(n \log n)$

- **基于先验信息，非比较**

  预先知道各数字符号之间的大小关系

  基数排序、桶排序；时间复杂度$O(n)$

MD5算法

- MD5的全称是Message-Digest Algorithm
- MD5由美国密码学家罗纳德·李维斯特（Ronald Linn Rivest）设计，于1992年公开，用以取代MD4算法。
  RFC-1321给出了标准规范的MD5算法
- MD5是输入不定长度信息，输出固定长度128-bits的算法。
- 经过程序流程，生成四个32位数据，最后联合起来成为一个128-bits散列。
  基本方式为，求余、取余、调整长度、与链接变量进行循环运算。
- https://zh.wikipedia.org/wiki/MD5

MD5算法

- MD5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组
- 经过了一系列的处理后，算法的输出由四个32位分组组成，将这四个32位分组级联后将生成一个128位散列值。
- 填充：将10...加到明文数据后面，让长度凑到L%512=448，再添加64bits表示明文长度，这样就能512bits分组

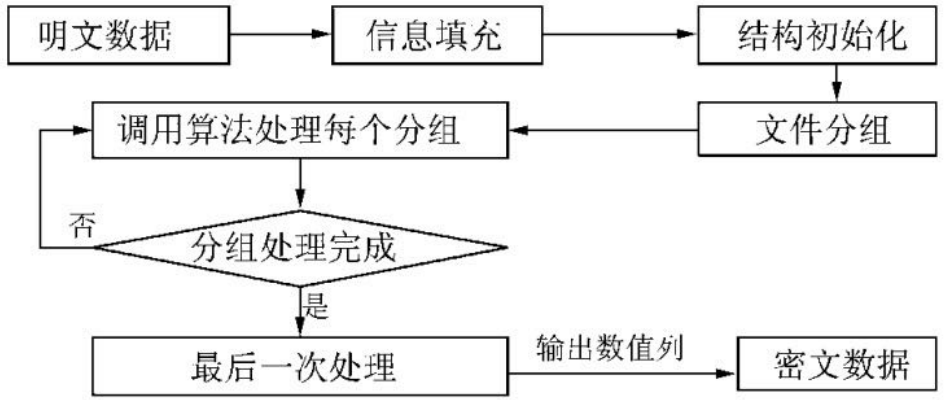


图 1　MD5 算法流程图

MD5算法伪代码


```
//Note: All variables are unsigned 32 bits and wrap modulo 2^32 when calculating
var int[64] r, k

//r specifies the per-round shift amounts
r[ 0..15]: = {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17, 22}
r[16..31]: = {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14, 20}
r[32..47]: = {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16, 23}
r[48..63]: = {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15, 21}

//Use binary integer part of the sines of integers as constants:
for i from 0 to 63
    k[i] := floor(abs(sin(i + 1)) × 2^32)

//Initialize variables:
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476

//Pre-processing:
append "1" bit to message
append "0" bits until message length in bits ≡ 448 (mod 512)
append bit length of message as 64-bit little-endian integer to message
```

```
//Process the message in successive 512-bit chunks:
for each 512-bit chunk of message
    break chunk into sixteen 32-bit little-endian words w[i], 0 ≤ i ≤ 15

    //Initialize hash value for this chunk:
    var int a := h0
    var int b := h1
    var int c := h2
    var int d := h3

    //Main loop:
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            f := (b and c) or ((not b) and d)
            g := i
        else if 16 ≤ i ≤ 31
            f := (d and b) or ((not d) and c)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f := b xor c xor d
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f := c xor (b or (not d))
            g := (7×i) mod 16

        temp := d
        d := c
        c := b
        b := leftrotate((a + f + k[i] + w[g]),r[i
        a := temp
    Next i
    //Add this chunk's hash to result so far:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d
End ForEach
var int digest := h0 append h1 append h2 append h3 //(expressed as little-endian)
```

```
//Main loop:
for i from 0 to 63
    if 0 ≤ i ≤ 15 then
        f := (b and c) or ((not b) and d)
        g := i
    else if 16 ≤ i ≤ 31
        f := (d and b) or ((not d) and c)
        g := (5×i + 1) mod 16
    else if 32 ≤ i ≤ 47
        f := b xor c xor d
        g := (3×i + 5) mod 16
    else if 48 ≤ i ≤ 63
        f := c xor (b or (not d))
        g := (7×i) mod 16

    temp := d
    d := c
    c := b
    b := leftrotate((a + f + k[i] + w[g]),r[i]) + b
    a := temp
Next i
```

计算一个文件的MD5值


```
1  import hashlib
2
3  blk_size = 2048   # 每次读取的字节数
4  m = hashlib.md5() # 创建一个散列函数对象
5  with open("2004-199.pdf", "rb") as f:
6      while True:
7          buf = f.read(blk_size)
8          if not buf:
9              break
10         m.update(buf)  # 更新
11
12 print(m.hexdigest())  # 打印散列值的16进制文本串
```

```
>>> %Run hashtest.py
 7667d184375a8d968e9e107217f7e8ea
```

MD散列冲突构造

Collisions for Hash Functions

MD4, MD5, HAVAL-128 and RIPEMD

Xiaoyun Wang[1], Dengguo Feng[2], Xuejia Lai[3], Hongbo Yu[1]

The School of Mathematics and System Science, Shandong University, Jinan250100, China[1]

Institute of Software, Chinese Academy of Sciences, Beijing100080, China[2]

Dept. of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai, China[3]

xywang@sdu.edu.cn[1]

revised on August 17, 2004

1 Collisions for MD5

MD5 is the hash function designed by Ron Rivest [9] as a strengthened version of MD4 [8]. In 1993 Bert den Boer and Antoon Bosselaers [1] found pseudo-collision for MD5 which is made of the same message with two different sets of initial value. H. Dobbertin[3] found a free-start collision which consists of two different 512-bit messages with a chosen initial value IV_0'.

IV_0': $A_0' = 0x12AC2375$, $B_0' = 0x3B341042$, $C_0' = 0x5F62B97C$, $D_0' = 0x4BA763ED$

Our attack can find many real collisions which are composed of two 1024-bit messages with the original initial value IV_0 of MD5:

Canvas第9周下载论文PDF

https://blog.csdn.net/SysProgram/article/details/73753354

SHA家族

2008 AMD CPU

SHA函数对比

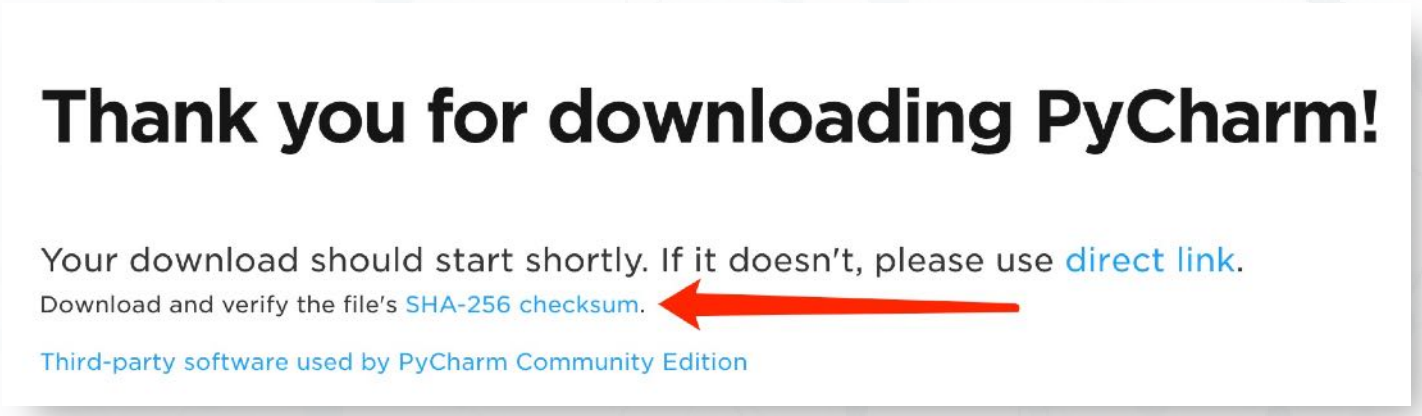
算法和变体		输出散列值长度 (bits)	中继散列值长度 (bits)	数据区块长度 (bits)	最大输入消息长度 (bits)	循环次数	使用到的运算符	碰撞攻击 (bits)	性能示例[3] (MiB/s)
MD5（作为参考）		128	128 (4 × 32)	512	无限[4]	64	And, Xor, Rot, Add (mod 2^{32}), Or	≤18 (发现碰撞)	335
SHA-0		160	160 (5 × 32)	512	$2^{64}-1$	80	And, Xor, Rot, Add (mod 2^{32}), Or	<34 (发现碰撞)	–
SHA-1		160	160 (5 × 32)	512	$2^{64}-1$	80		<63[5] (发现碰撞[6])	192
SHA-2	*SHA-224*	224	256 (8 × 32)	512	$2^{64}-1$	64	And, Xor, Rot, Add (mod 2^{32}), Or, Shr	112	139
	SHA-256	256						128	
	SHA-384	384	512 (8 × 64)	1024	$2^{128}-1$	80	And, Xor, Rot, Add (mod 2^{64}), Or, Shr	192	154
	SHA-512	512						256	
	SHA-512/224	224						112	
	SHA-512/256	256						128	
SHA-3	*SHA3-224*	224	1600 (5 × 5 × 64)	1152	无限[7]	24[8]	And, Xor, Rot, Not	112	–
	SHA3-256	256		1088				128	
	SHA3-384	384		832				192	
	SHA3-512	512		576				256	
	SHAKE128	*d* (arbitrary)		1344				min(*d*/2, 128)	–
	SHAKE256	*d* (arbitrary)		1088				min(*d*/2, 256)	

完美散列函数的应用

- **作为校验码防止出错（或恶意替换）**

  文件F，和Hash(F)，如果F的内容有任何错误变为F'

  那么Hash(F')就与Hash(F)有显著区别




```
248a02f4ea28bdc0b61b8d34e362a1d7fd24af88db02bae34f42070c2968b13f
*pycharm-community-2021.1.1-aarch64.dmg
```

```
[chenbin@chenbindeAir Downloads % shasum -a 256 pycharm-community-2021.1.1-aarch
248a02f4ea28bdc0b61b8d34e362a1d7fd24af88db02bae34f42070c2968b13f  pycharm-commu
```

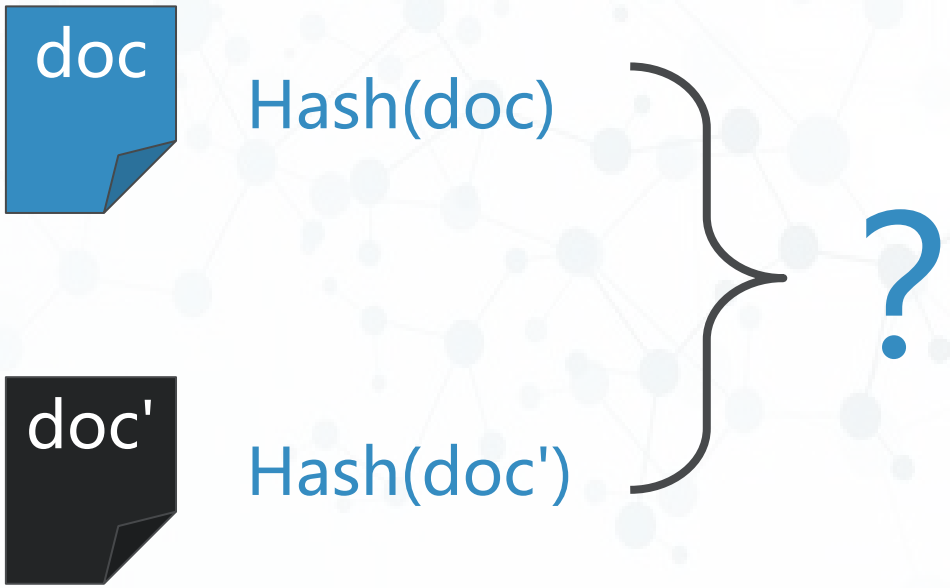
完美散列函数的应用

作为防止篡改的手段

Alice写了一个电子借据doc，和Hash(doc)一起发送给Bob……

且慢！如果Alice和Bob偷偷改了借据，

都声称自己的那个doc是原件怎么办？



完美散列函数的应用：非对称加密

基于数学难题的公开密钥加密

例如：两个特别大的素数P1和P2，其乘积PU=P1*P2

用PU加密的信息，仅能用P1（或P2）解密，反之亦然

PU称为“公开密钥”——公钥——公诸于众

P1（或P2）称为“私有密钥”——私钥——要藏好勿泄漏

仅知道公钥PU，是无法在合理的时间内得到私钥P1的

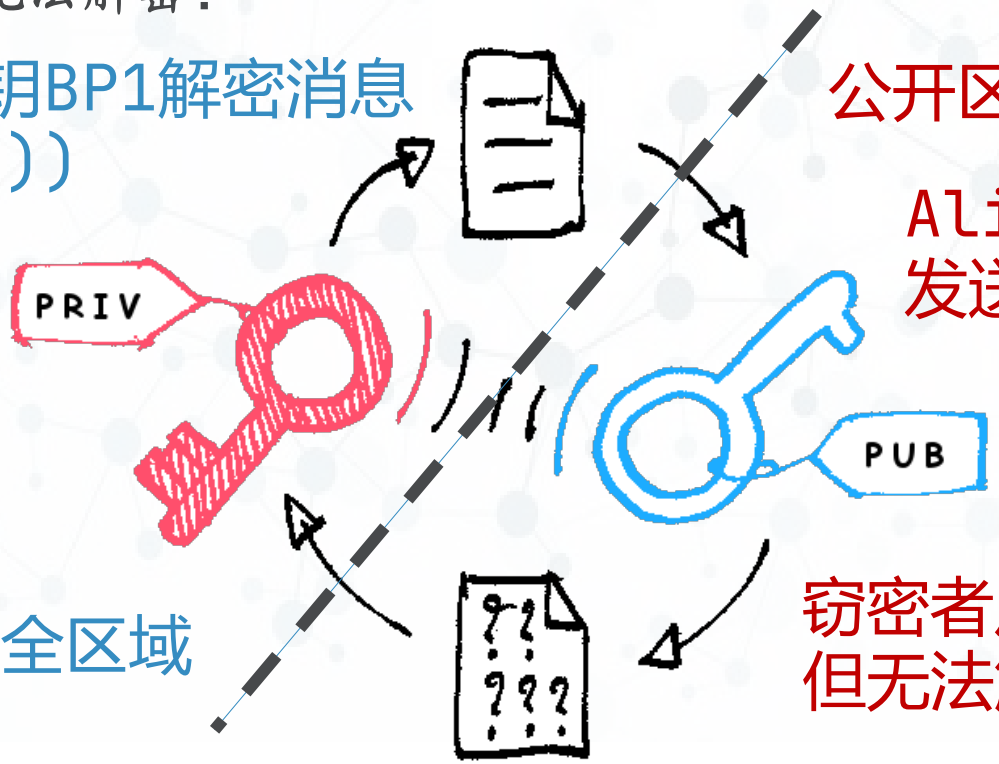
非对称加密通信：允许不可靠信道

- **假设传输信息的途径能够被任何人截取**

  例如凯撒加密（对称加密），加密和解密的密钥是同一个

  A-(+3)-D，这个3就是密钥，如果被人截取，则加密被破解

  如何让窃密者无法解密？



Bob用自己的私钥BP1解密消息
BP1(BPU(text))
得到明文text

公开区域

Alice用Bob的公钥BPU加密消息
发送BPU(text)

安全区域

窃密者从网络截取了密文BPU(text)
但无法解密

数字签名：防止篡改/抵赖

- Alice借了Bob的钱，写下一个电子借据doc
- Alice将Hash(doc)，用自己的私钥AP1加密为AP1(Hash(doc))
- 把doc和AP1(Hash(doc))一起发送给Bob
- 这样，Bob和所有人都可以用Alice的公钥APU来检查doc是不是原文

  Hash(doc) == APU(AP1(Hash(doc)))

  （用私钥加密的信息只能用公钥解密，Alice的公钥APU是众所周知的）

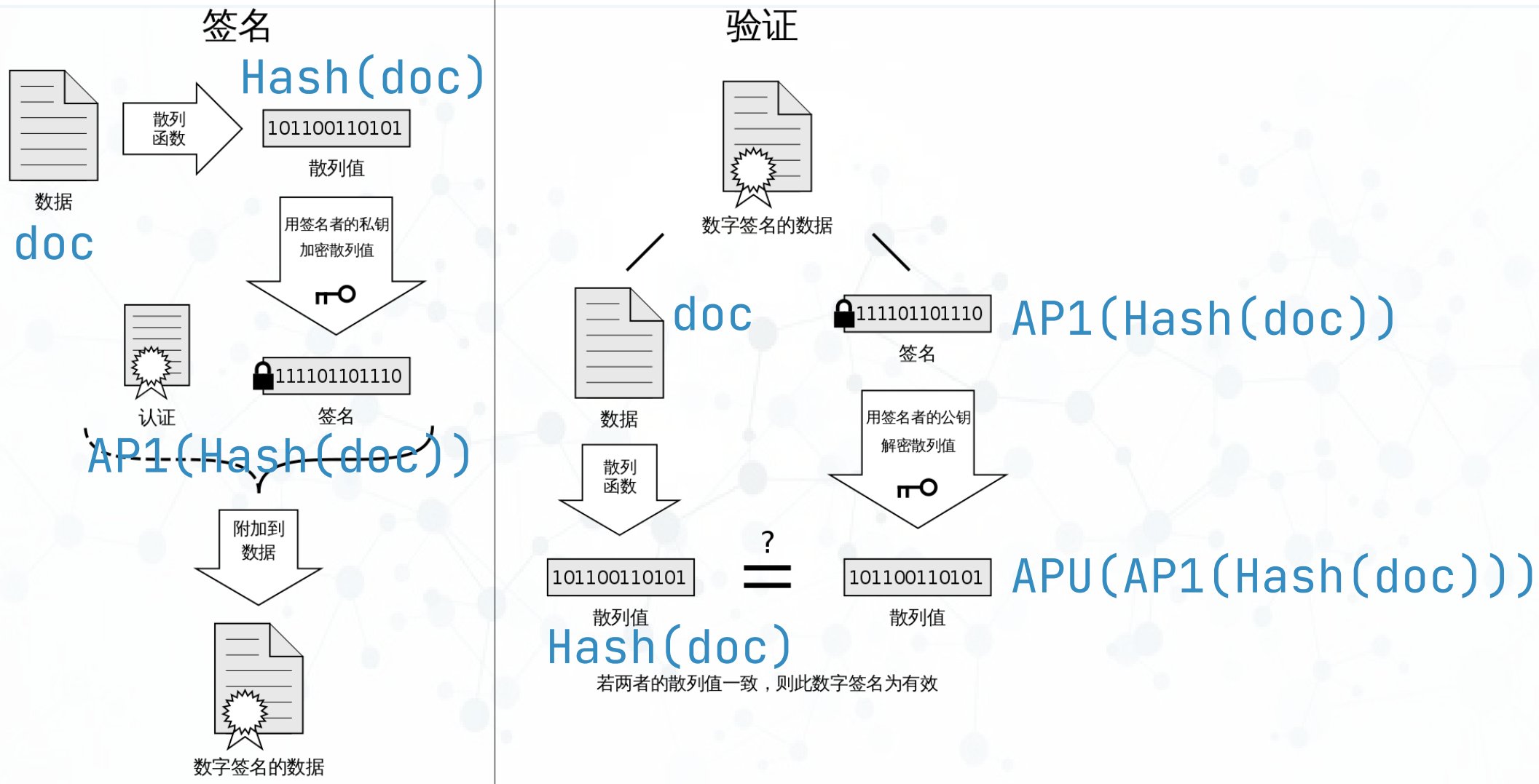
- Alice也无法抵赖说，从来没写过借据doc

  因为AP1(Hash(doc))在Bob和大家的手里，

  没有私钥AP1的人是无法算出这个签名的

  而且有Hash(doc)在大家手里，Alice当然也无法篡改借据doc的内容

数字签名和验证：Alice的签名和验证



最后一个问题：公钥的有效性

- **某天，Alice说你们拿到的我的公钥APU，是假的！**

  有人冒名顶替我向大众发布了APU

  这样就可以抵赖掉借据了？

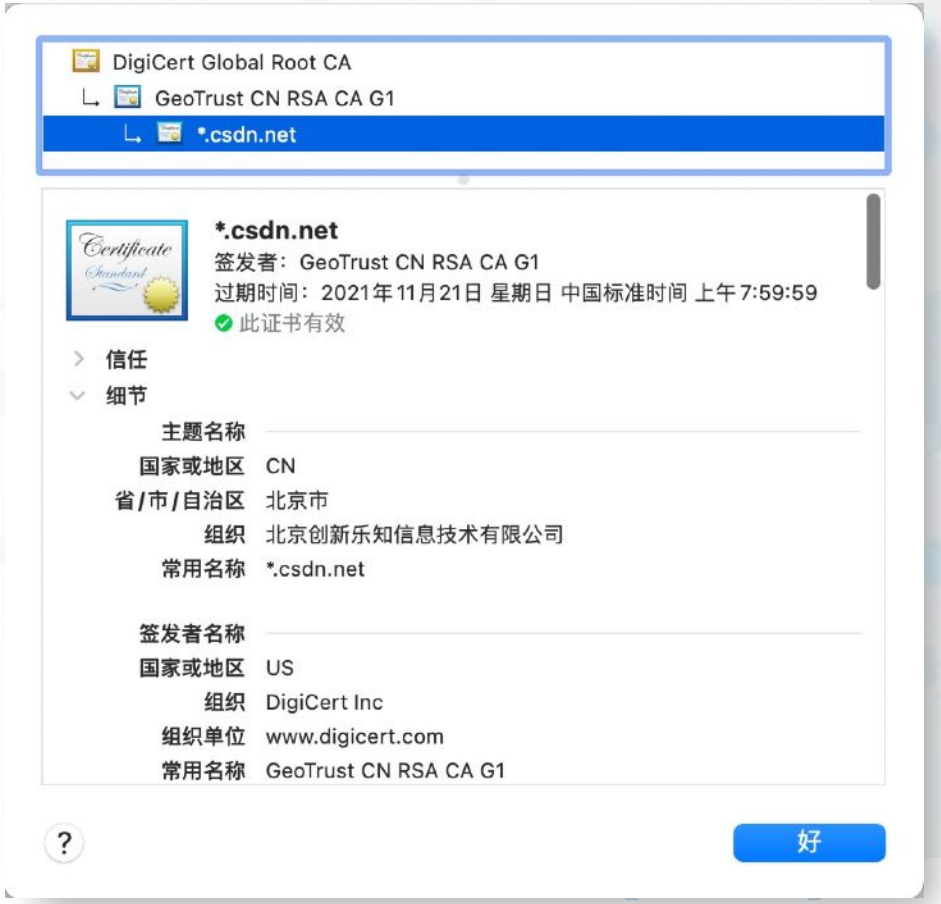
- **技术之外的解决方案：权威机构发行数字证书**

  大家信任权威机构（如证书中心、银行等）

  每个人通过线下等身份认证途径，向权威机构申请证书

  权威机构用自己的私钥AuthP1来加密Alice提交的个人信息（包括公钥APU），这个AuthP1(AliceInfo+APU)，就是权威机构签发的数字证书

  可以用来证明持证者的身份



【K08】散列的课堂练习

- 请设计算法重新实现散列表的put方法，使得散列表可以在负载因子达到一个预设值时自动调整大小。
- **提示**：算法需要处理现有在散列表中的数据项，保证在扩充散列表之后，能正常访问现有的数据项。
- 可以采用伪代码说明算法。

下课！