RFC6962（证书透明度）

一.文档简介

1.目的

主要目的是限制CA的权利。在文档颁发以前，CA可以对任何网站颁发证书，无论这个网站是否申请，CA的颁发证书的权利并不受限。CA可以利用这个权利：对想要攻击的网站颁发证书，而后利用证书实现中间人攻击。而本文档颁发的主要目的就是防止这种攻击的出现。

2.关于merkle树

证书透明度旨在通过提供所有已颁发证书的可公开审核、仅附加且不受信任的日志来缓解证书颁发错误的问题。日志是可公开审核的，因此任何人都可以验证每个日志的正确性，并在向其中添加新证书时进行监视。日志本身并不能防止错误发布，但它们可以确保相关方（尤其是证书中指定的方）能够检测到此类错误错误发行。每个日志由证书链组成，任何人都可以提交。公共CA将把其所有新颁发的证书贡献给一个或多个日志；此外，预计证书持有人将贡献自己的证书链。那些担心错误发布的人可以监视日志，定期询问日志中的所有新条目，从而可以检查他们负责的域是否已颁发了他们意想不到的证书。

其中，由于日志存在仅附加属性（只能在末尾添加，而不能删除或修改），因此日志使用merkle树来实现（一种使用哈希函数连接父节点与子节点的二叉树）。

二.Merkle树实现

1.初始化

(1)思路

在RFC6962文档中，哈希算法使用SHA256，因此在本次实验中，也同样选用SHA256进行哈希。merkle树的输入为一个列表D[n] = { D[0]...D[n-1] }，其中D[m]代表第m+1个元素的取值。merkle树的输出为merkle树的根，记做MTH(D[n])。

其中，MTH(D[n])可以递归定义为

$$MTH(D[n]) = \begin{cases} None & n = 0 \\ SHA\_256(0x00 || D[0]) & n = 1 \\ SHA\_256(0x01 || MTH(D[0:k]) || MTH(D[k:n])) & else \end{cases}$$

$$其中 k < n <= 2k; D[0:k]为左子树; D[k:n]为右子树$$

注意，我们不要求输入列表的长度为2的幂。由此产生的Merkle树可能不平衡；然而，它的形状是由叶子的数量决定的。

(2)代码

定义一个类，名为MerkleTree。其中包含了merkle树初始化，存在性证明，不存在性证明等方法。

计算MTH的python代码如下。为了后序计算方便，这里MTH返回的是十六进制的整型数据。

```
 def MTH(self,leaf_ls):
        n = len(leaf_ls)
        if n==0:
            return
        elif n==1:
            return
int(SHA256.new((str(0x0|leaf_ls[0])).encode()).hexdigest(),base=16)
        else:
            k = math.ceil(n/2)
          return  int(SHA256.new((str(0x1|self.MTH(leaf_ls[0:k])|

self.MTH(leaf_ls[k:n]))).encode()).hexdigest(),base=16)
```

初始化的python代码如下。注意，为了方便以后的不存在性证明，在初始化的时候已经将元素取值列表排好序了。

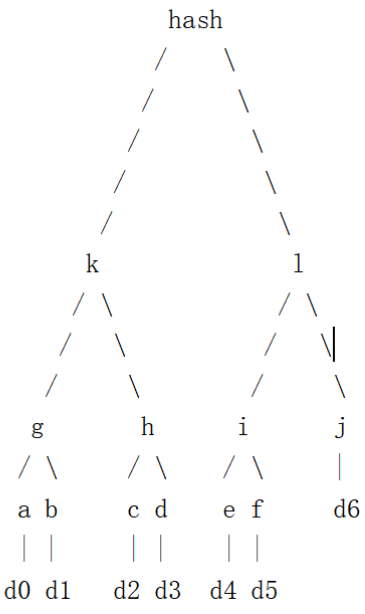
```
    def __init__(self,leaf_ls):
        '''
        function: 初始化函数。按leaf_ls生成merkle树
        :param leaf_ls: int类型-叶子取值列表
        '''
        self.leaf_ls = leaf_ls                #元素取值列表
        self.leaf_ls.sort()                   #对列表排序
        self.leaf_num = len(leaf_ls)          #叶节点的数量
        self.root = self.MTH(self.leaf_ls)  #根节点取值
```

2.存在性证明

(1)思路

Merkle树中叶节点的审计路径是由叶节点计算出merkle树的根节点所需的额外节点的最短列表。如果通过该叶节点和其对应的审计路径计算出的根节点与真实的根节点相同，那么该叶节点的存在性证明通过。

eg.



则d0的审计路径是[b, h, l]；d3的审计路径 是[c, g, l]；d4的审计路径是 [f, j, k]；d6 的审计路径是 [i, k]。

我们可以将审计路径的计算过程递归的定义为(Path(m,D[n]):计算D[m]的审计路径)

$$Path(m,D[n]) = \begin{cases} None & m=0, n=1 \\ PATH(m, D[0:k]) : MTH(D[k:n]) & m < k \\ PATH(m, D[n]) = PATH(m-k, D[k:n]) : MTH(D[0:k]) & m >= k \end{cases}$$
$$其中 k < n <= 2k$$

(2)代码

首先要计算审计路径，按照上述公式递归计算即可

```
    def Path(self,m,leaf_ls):
        '''
        function: 对leaf_ls[m]节点求审计路径
        :param m: int型
        :leaf_ls: 列表型
        :return: list型，返回leaf_ls[m]的审计路径(有序)
        '''
        if m>=self.leaf_num:
            return
        n = len(leaf_ls)
        if n==0 or n==1:
            return
        else:
            k = math.ceil(n/2)
            if m<k:
                result = []
                tmp = self.Path(m, leaf_ls[0:k])
                if tmp!= None and len(tmp)!=0:
                    for i in tmp:
                        result.append(i)
                result.append(self.MTH(leaf_ls[k:n]))
                return result
            else:
                result = []
                tmp = self.Path(m-k, leaf_ls[k:n])
                if tmp != None and len(tmp) != 0:
                    for i in tmp:
                        result.append(i)
                result.append(self.MTH(leaf_ls[0:k]))
                return result
```

对于元素取值为n的节点进行存在性证明。首先获取取值为n节点的下标，如果无法获取，则存在性证明失败。而后根据下标计算节点所对应的审计路径。得到审计路径之后，首先计算对节点取值进行哈希，得到merkle树中的叶节点。而后利用叶节点和审计路径求出跟，记做result。最后用result和merkle树的根做对比，如果相等，则存在性证明成功；如果不等，则失败。

```
    def inclusion_proof(self,n):
        '''
        funtion: 对n进行存在性证明
```

```
        :param n: int型，n为元素的值
        :return: bool值，true代表n存在于merkle树中，false代表算法失败
        '''
        try:
            index = self.leaf_ls.index(n)
        except:
            return False
        audit_path = self.Path(index,self.leaf_ls)
        hash_leaf = int(SHA256.new((str(0x0|n)).encode()).hexdigest(),base=16)
        result = hash_leaf
        if audit_path!= None:
            for i in audit_path:
                result =
int(SHA256.new((str(0x1|result|i)).encode()).hexdigest(),base=16)


        return (result==self.root)
```

3.不存在性证明

(1)思路

想要对n进行不存在性证明，首先要保证元素取值列表有序。而后在元素取值列表中取出与n相邻的元素(如2.5，相邻元素为2和3)，对这两个相邻的元素进行存在性证明。如果两个元素都在merkle树中，那么证明两个元素在元素取值列表中也相邻（中间没有其他元素）。若上述证明全部通过，则不存在性证明完成。

(2)代码

排序的工作在init函数中已经完成，因此这里不用再排序。首先在元素取值列表中找到与n相邻的元素，比n小的记做b，比n大的记做a，如果找不到，说明n不在merkle树中，证明完成。若可以找到，则分别对a，b进行存在性证明。在元素取值列表中求出a，b所对应的下标，如果二者相减的绝对值为1，说明a，b在merkle树中相邻。不存在性证明完成。

注意在不存在证明中，函数返回True说明不存在，返回False说明存在。与存在性证明的返回值含义恰好相反。

```
 def exclusion_proof(self,n):
        '''
        funtion:对n进行不存在证明
        :param n: float类型，n为元素的值
        :return: bool值，true代表n不存在于merkle树中，false代表算法失败
        '''
        #找到两个相邻元素
        try:
            below = [i for i in self.leaf_ls if i<n]
            above = [j for j in self.leaf_ls if j>n]
            b = below[-1]
            a = above[0]
        except:
            return not self.inclusion_proof(n)
```

```
        #对这两个相邻元素进行存在性证明
        if(self.inclusion_proof(b)==True and self.inclusion_proof(a)==True):
        #证明这两个相邻元素相邻
            if(abs(self.leaf_ls.index(b)-self.leaf_ls.index(a))==1):
                return True

        return False
```

三.结果展示

编写一个简单的测试函数，其中size为merkle树的元素个数，定义为10^5。对merkle分别进行初始化，存在性证明和不存在性证明三个操作，并记录时间。

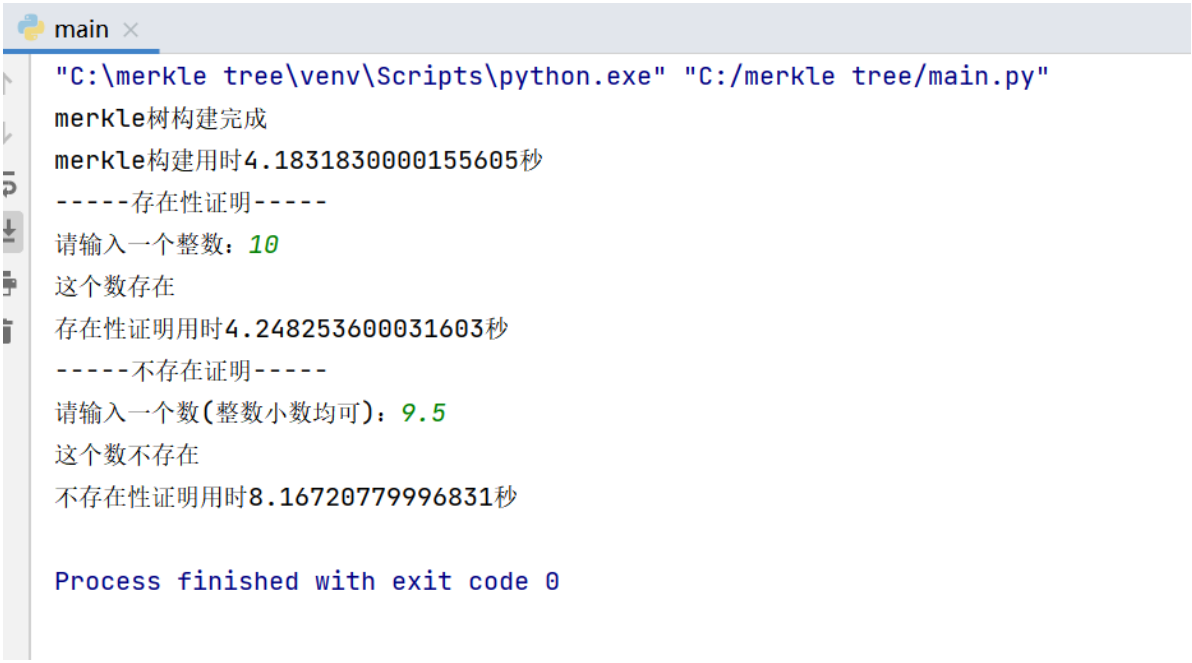
```
if __name__ =="__main__":
    start1 = time.perf_counter()

    size = 100000
    leaf_ls = []
    for i in range(size):
        leaf_ls.append(i)
    test = MerkleTree(leaf_ls)
    print("merkle树构建完成")
    end1 = time.perf_counter()
    print(f"merkle构建用时{end1 - start1}秒")

    print("-----存在性证明-----")
    example1 = int(input("请输入一个整数："))
    start2 = time.perf_counter()
    if(test.inclusion_proof(example1)==True):
        print("这个数存在")
    else:
        print("这个数不存在")
    end2 = time.perf_counter()
    print(f"存在性证明用时{end2 - start2}秒")

    print("-----不存在证明-----")
    example2 = float(input("请输入一个数(整数小数均可)："))
    start3 = time.perf_counter()
    if(test.exclusion_proof(example2)==True):
        print("这个数不存在")
    else:
        print("这个数存在")
    end3 = time.perf_counter()
    print(f"不存在性证明用时{end3-start3}秒")
```

运行代码，结果如下图。


main
"C:\merkle tree\venv\Scripts\python.exe" "C:/merkle tree/main.py"
merkle树构建完成
merkle构建用时4.1831830000155605秒
-----存在性证明-----
请输入一个整数：10
这个数存在
存在性证明用时4.248253600031603秒
-----不存在证明-----
请输入一个数(整数小数均可)：9.5
这个数不存在
不存在性证明用时8.16720779996831秒
Process finished with exit code 0