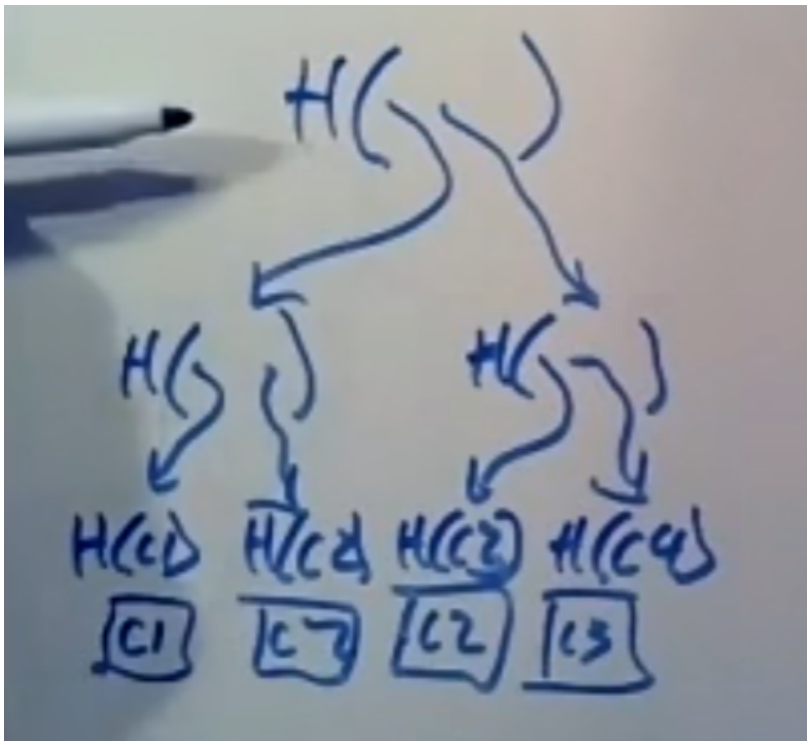
LECURE 18 FORK CERTFICATE TRANSPARENCY EQUIVOCATION

1. 背景

- Raft/Paxos 等共识算法都是在相互信任的结点之间进行部署。而有些系统中，结点可以自由地进出一个分布式系统，这些结点相互不信任（同时没有权威机构），因此需要在这些相互不信任的结点之上构建分布式系统。
- 关键问题：确保各方看到的关于证书的信息是相同的
- 证书透明度可以用于非加密货币
- **中间人攻击（Man-In-The-Middle）**：中间人可能截取DNS报文，伪造服务器，该服务器提供和目标网页一样的页面，而后截取相关信息。
- Certificate Authority：可以抵抗中间人攻击
    - 网站服务器有私钥
    - 网站向CA申请证书，证书内容包括
        - 网站公钥
        - CA签名
        - 服务器名称
    - 认证流程：
        1. 客户端访问服务器
        2. 服务器向客户端发送证书
        3. 客户端要求服务器用私钥签名
        4. 服务器返回签名
    - 如何抵抗中间人攻击：中间人没有服务器私钥，无法进行签名。
    - CA模式的缺点：
        - CA太多，需要存储的CA的公钥太多

    - CA本身不是完全可信的，有可能被贿赂，**故意**给中间人下发证书
    - 小型网站很难获得CA的证书
    - CA下发证书/ 网站向谁申请证书 较为自由
    - CA很难确认申请证书的网站是否的确拥有那个域名，因此可能**无意**下发证书给中间人
    - **可以通过建立统一的证书数据库来消除这些问题，但是在现实世界中显然不可能**
- Merkle Tree



哈希函数应该为密码哈希函数，具有抗碰撞性

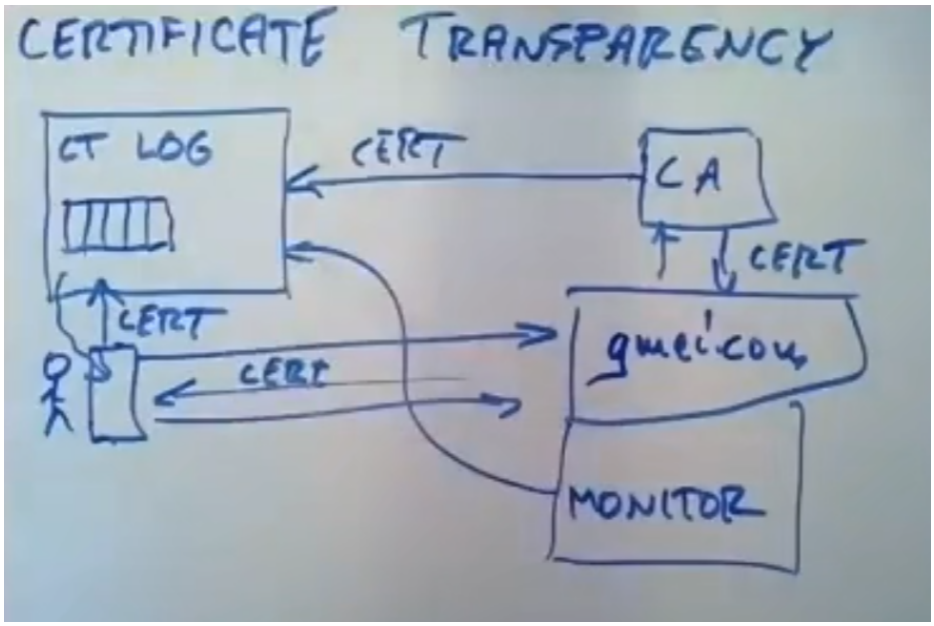
    - 多叉树，通常为二叉树
    - 叶子结点为证书/其他数据的哈希值
    - 非叶子结点为其所有子结点的哈希值
    - 根节点被称为Merkle **根/STH（signed tree head)**

2. Certificate Transparency

根本思路是**审计**（audit）

- 中间人仍然可以发布虚假证书，但是这些证书会被公开，可以进行检查。

2.1. 基本结构：



- 客户端访问服务器的基本流程：
  1. 客户端提出访问请求
  2. 服务器发送向客户端发送证书
  3. 客户端将证书发送给Certificate Transparency Log Server，询问证书是否在其Log中
  4. CT服务器向客户端回复 yes/no
- 服务器申请证书流程
  1. 向CA申请证书
  2. CA向服务器下发证书
  3. CA将证书发送给CT Server，CT Server将其插入到自己的log中
- Monitor：
  - 大型公司或者一些独立机构会建立monitor
  - monitor会定期向CT Server索要新插入的log entry
  - 大公司会检查新证书是不是盗用了自己的域名向CA申请证书，在这种情况下，公司会向客户通知

- 现实世界中存在多个CT Server。monitor也不会因为一两次发现某个CT Server出错就将其从可信列表中删除。但是需要CT Server提供者给出合适的理由。

2.2. Log Server设计

2.2.1. Log **应该具有的特性：**

- Append-only：append-only可以确保如下情况不会发生：CT 先向客户端发送假证书，而后删除该证书，如此monitor就无法发现这个假证书
- no forks：Log server不能保留两套Log。如果允许server保留两套log，则sever可能将一套给客户端看，另一套给monitor看
- untrusted

2.2.2. Merkle Tree

仅考虑了叶子结点为2的整数次幂的情况！！！！

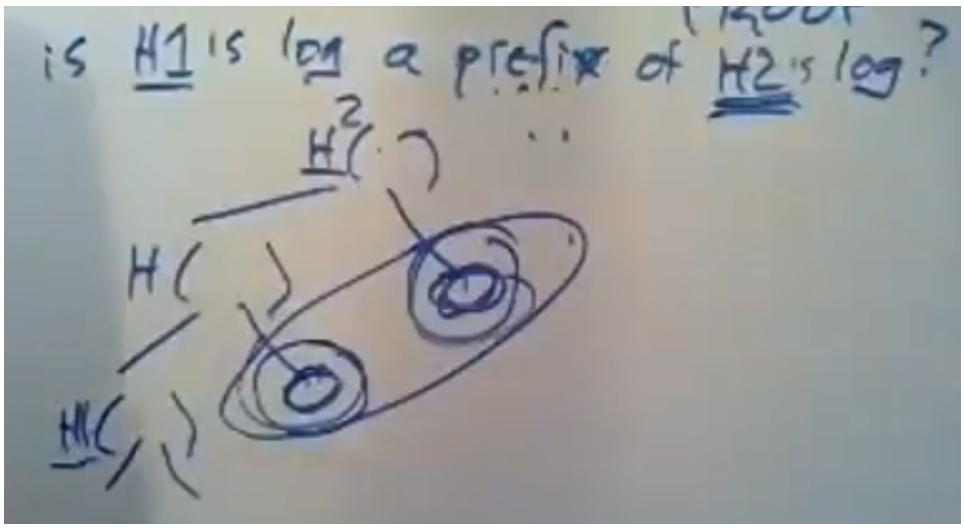
- 使用Merkle Tree来强制Log Server证明关于日志的信息。
  - 包含（inclusion）证明：用于证明Log 中在**特定位置** 的确有 **特定证书。**防止恶意的CT server 错误地返回yes/no。如果CT Servers说谎，则以后可以忽略不使用该server。
  - 步骤：
    1. 客户端从Log Server处获得了一个STH
    2. Log Server向客户端返回特定证书在log中的位置，以及其他哈希值。（猜测在Merkle Tree为二叉树的情况下，可能需要返回当前证书所有祖先节点的一个儿子结点的哈希值，这个儿子结点不应该是当前证书的某个祖先节点或该证书本身，见下图）

在证书数目为n的情况下，Log Server所需要返回的哈希值数目大约是$O(logn)$级别的。

3. 客户端按照返回的数据重新计算哈希值即可。

2.2.3. Fork Attack-Equivocation

- 防止Log Server准备了两套Log，将含有bogus certificate的Log对应的STH返回给客户端。
- 机制：**Gossip。**所有的客户端随机将自己所见的STH放入一个STH池中，随机从池中抽取STH来检查是否存在不一致的STH。
- Merkle Log Consistency Proof：
  - 在STH池中，不同的STH不一定代表产生了fork，也可能是在STH1下，产生了新的certificate，进而生成了新的STH2，需要检验两个不同的STH是否存在前缀关系。
  - 检验方法：



只需要使用H1和其他的结点的哈希值，计算出对应的STH值H2'，而后判断H2与H2'是否相等即可。

2.3. Fork Consistency

客户端不会允许切换到另一个与当前STH不兼容的STH（或者认为是log 分支）上，仅允许在当前分支上进行拓展。

3.