Fingerprints and Trees
Additional Applications for Hash Functions

Foundations of Cryptography
Computer Science Department
Wellesley College

Fall 2016

Table of contents

Introduction

Efficient storage

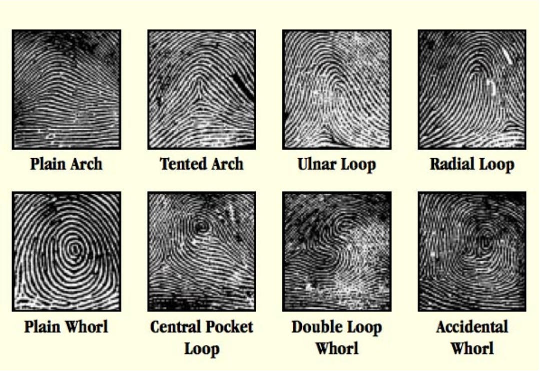
Merkle trees

Password Hashing

Commitment Schemes

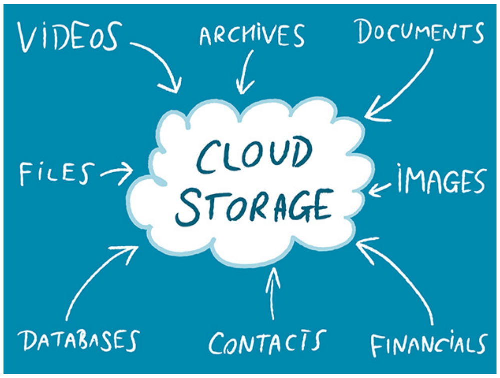
Fingerprinting

- Virus scanners identify viruses by storing a database containing hashes of known viruses.
- When an application or email attachment is downloaded, the scanner looks up the its hash in this database.
- Overhead is feasible since only a very short string needs to be recorded (and/or distributed) for each virus.



Deduplication

- When multiple users store the same file in the cloud, only one copy is needed.
- Duplication is avoided by having the user first upload a hash of the file they want to store.
- If the hash is already in the cloud, the cloud-storage provider simply adds a pointer to the existing file to indicate the this specific user also stored the file.



Distributed databases

- Consider building a database distributed over a large number of peers that support indexing and searching.
- Peers query the database by supplying a key and is given in return the pair (key, value) that matches the key.
- Peers may also insert (key, value) pairs into the database. How best to do this?

My brother’s approach

1. Randomly scatter the (key, value) pairs across the peers and ...
2. ... have each peer maintain a list of the IP addresses of all participating peers.
3. The querying peer looks into every bag, box, and drawer.

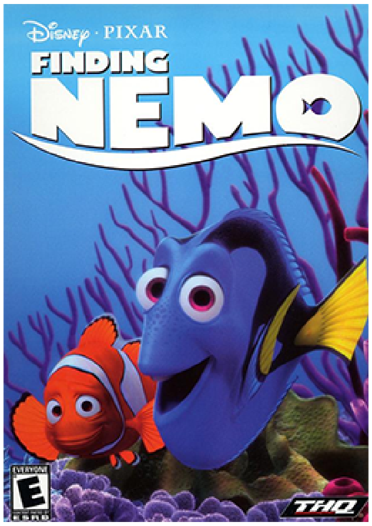
Distributed hash table

- Assign a unique integer identifier in the range $[0, 2^n - 1]$ for some fixed n to each peer.
- Using a publicly available hash function, assign integers in the same range to each key.
- Assign each (key, value) to the peer whose identifier is "closest*" to the hashed key.

*Say, the immediate successor of the key.

For example, ...

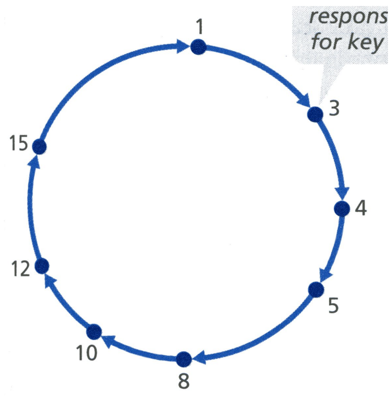
- Suppose $n = 4$, so keys are in the range $[0, 2^4 - 1]$ and there are currently eight peers: $1, 3, 4, 5, 8, 10, 12, 15$.
- Where should the pair (11, *Topper*) go and how does Alice find it?
- Suppose Alice wants to insert $(key, value)$ into the DHT. How does she determine the peer closest to key?*



*Well, she could keep track of all the peers in the system?

Circular distributed hash tables

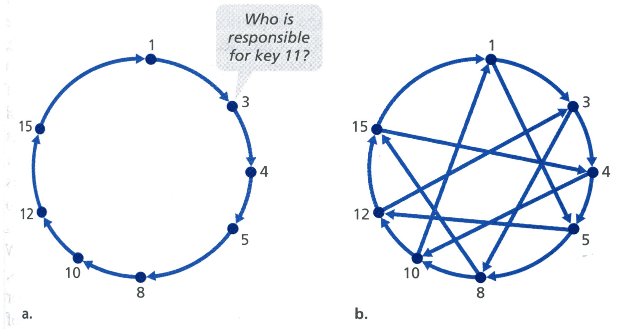
- Peers are organized in a circle. Each peer keeps track of its immediate predecessor and successor modulo 2^n.
- Peer 3 finds the record with key 11 by first passing a message to peer 4, who ...
- When the message reaches the responsible party, direct communication takes place.



*These are not physical links, the peers form an abstract overlay network above the"underlay" computer network.

Adding shortcuts

- Circular DHT is an elegant solution for reducing overlay information, but comes at a cost.



- An improvement is to add shortcuts, but there is a trade-off between the number of neighbors each peer has to track and the number of message that the DHT needs to send to resolve a single query.
- When a peer receives a message that is querying for a key, it forwards the message to the neighbor which is closest the key.

Fingerprinting files

- A client retrieves her previously uploaded file, x, from the cloud and wants to make sure it has not been altered in the meantime.
- She could maintain a copy and compare.*
- An obvious solution is to use the "fingerprinting" technique to store a short digest $h := H(x)$ and check that $H(x') \overset{?}{=} h$ for the returned file, x'

*But that would defeat the purpose of the cloud.

What happens when there are multiple files?

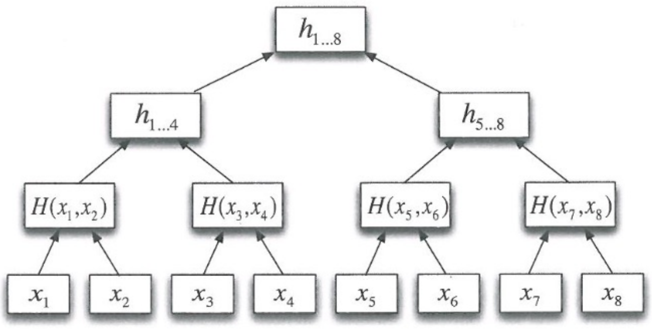
- We could simple hash each file, $x_1, \ldots, x_t$ independently, store the digests $h_1, \ldots, h_t$, then verify retrieved files as before.*
- Alternative, we could hash that lot, $h := H(x_1, \ldots, x_t$ and store only h**.
- Merkle trees give a tradeoff between these two extremes.

*But then client's storage grows linearly in t.

**But there's a downside here too.

Merkle trees

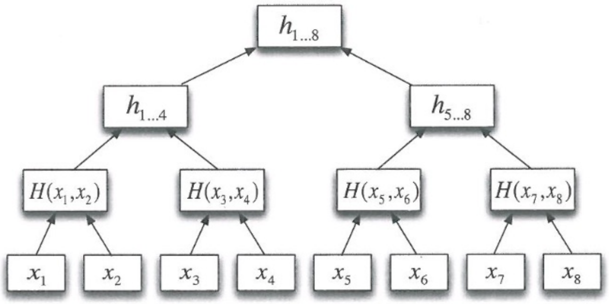
A Merkle tree computed over inputs $x_1, \ldots, x_t$ is a binary tree of depth $\log_2 t$ whose leaves are the inputs, and each internal node is the hash of the values of its two children.



Theorem 5.11. Let (sf Gen_H, H) be a collision resistant, then (sf $\mathsf{Gen}_H, \mathcal{MT}_t$)) is also collision resistant for any fixed t, where $\mathcal{MT}_t$ denotes the function that takes t input values $x_1, \ldots, x_t$, computes the resulting Merkle tree, and outputs its root value.

A solution to our problem using $\mathcal{O}(\log_2 t)$ *communication*

- The client computes $h := \mathcal{MT}_t(x_1, \ldots, x_t)$, uploads files $x_1, \ldots, x_t$ to the server, and stores h and t.
- At retrieval time, the server sends x_i along with a proof π_i of correctness consisting of the values of nodes in the Merkle tree adjacent to the path from x_i to the root.



- The client uses these to recompute the path from x_i to the root and verify that it equals h.

Password Hashing

- When a user types her password before she can use her laptop.
- To authenticate the user, some form of the user's password must be stored on the laptop. If stored in the clear, an adversary who steals the laptop can read the user's password off the hard drive and login as her.
- This risk can be mitigated by storing a hash, hpw = H(pw), of the password instead. When a user enters pw, the operating system checks whether $H(pk) \overset{?}{=} hpw$ before granting access.

*Our Linux system does just that. Hashes used to be stored openly in /etc/passwd, but off-line attacks made that a very bad idea.

Attempts to bound the attacker's chances

- Suppose the password is chosen from a relatively small space D, for example words from an English dictionary ($|D| \approx 80,000$).
- An attacker can then enumerate all possible passwords $pw_1, pw_2, \ldots \in D$ and check whether $H(pw_i) = hpw$.
- We would like to claim this is the best they can do.

Randon oracle model to the rescue

- If H is modeled as a random oracle, we can formally prove that recovering pw from hpw requires $|D|/2$ evaluations on H, on average*.
- However, preprocessing can be used to generate large tables that enable inversion faster than exhaustive search.
- Even if passwords rare chosen as a random combination of 8 alphanumeric characters (giving a password space of size $N = 62^8 \approx 2^{47.6}$, there is an attack using time and space $N^{2/3}$ that will be highly effective.

*Assuming pw is chosen uniformly from D.

Mitigating the threat of password cracking

- We could use "slow" hash functions or slow existing ones down using multiple iterations by computing $H^{(I)}(pw)$ for large I.
- Alternatively, we could introduce *salt*.
- When a user enters their password, a long random value s is generated unique to the user and value $(s, hpw) = H(r, pw)$ stored instead of $H(pw)$.

*Of course this comes at a cost to legitimate users.

Commitment schemes

A *commitment scheme* allows one party to "commit" to a message m by sending a commitment com such that:

- *Hiding:* the commitment reveals nothing about m.
- *Binding:* it is infeasible for the committer to output a commitment that can later be "opened" as two different message m, m'.

Two experiments

The commitment hiding experiment $\mathsf{Hiding}_{\mathcal{A},\mathsf{Com}}(n)$*:*

1. Parameters params $\leftarrow \mathsf{Gen}(1^n)$ are generated.
2. The adversary $\mathcal{A}$ is given input params, and outputs a pair of message $m_0, m_1 \in \{0,1\}^n$.
3. A uniform b $\{0,1\}$ is chosen and com $\leftarrow \mathsf{Com}(\mathsf{params}, m_b; r)$ is computed.
4. The adversary $\mathcal{A}$ is given com and outputs a bit b'.
5. The output of the experiment is 1 if and only if $b' = b$.

The commitment binding experiment $\mathsf{Binding}_{\mathcal{A},\mathsf{Com}}(n)$*:*

1. Parameters params $\leftarrow \mathsf{Gen}(1^n)$ are generated.
2. The adversary $\mathcal{A}$ is given input params, and outputs $(\mathsf{com}, m, r, m', r')$
3. The output of the experiment is 1 if and only if $m \neq m'$ and $\mathsf{Com}(\mathsf{params}), m; r) = \mathsf{com} = \mathsf{Com}(\mathsf{params}), m'; r')$.

Secure commitment schemes

Definition 5.13. A commitment scheme is *secure* if for all PPT adversaries $\mathcal{A}$ there is a negligible function negl such that

$$\Pr[\mathsf{Hiding}_{\mathcal{A},\mathsf{Com}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$$

and

$$\Pr[\mathsf{Binding}_{\mathcal{A},\mathsf{Com}}(n) = 1] \leq \mathsf{negl}(n).$$

Remark. It is possible to construct a secure commitment scheme from a random oracle H. How?

Remark. Commitment schemes can be constructed without random oracles using one-way functions. But that is story for another day.