Admin:

Today:

Pset #2 due Friday; email to 6.857-staff@mit.edu

Project proposal presentations on Monday!

* Hash function properties
* Hash function applications
Hash function desirable properties:

1. **"One-way"** (pre-image resistance)
   "Infeasible", given \( y \in \mathbb{Z}_d^{13} \) to find any \( x \) s.t. \( h(x) = y \) (\( x \) is a "pre-image" of \( y \))

   \[
   h : \{0,1\}^* \rightarrow \{0,1\}^{13}
   \]

   (Note that a "brute-force" approach of trying \( x \)'s at random requires \( \Theta(2^d) \) trials (in ROM),)

2. **"Collision-resistance"** (strong collision resistance)
   "Infeasible" to find \( x, x' \) s.t. \( x \neq x' \) and \( h(x) = h(x') \) (a "collision")

   \[
   h(x) = h(x')
   \]

   In ROM, requires trying about \( 2^{d/2} \) \( x \)'s \((x_1,x_2,...)\) before a pair \( x_i,x_j \) colliding is found. (This is the "birthday paradox",)}
Note that collisions are unavoidable since
\[ |E_{0,1}^{\mathbb{Z}^d}| = \infty \]
\[ |E_{0,1}^{\mathbb{Z}^d}| = 2^d \]

**Birthday paradox detail:**

If we hash \( x_1, x_2, \ldots, x_n \) (distinct strings)
then
\[
E(\# \text{ collisions}) = \sum_{i \neq j} \Pr(h(x_i) = h(x_j))
\]
\[
= \binom{n}{2} \cdot 2^{-d} \quad \text{[if h "uniform"]}
\]
\[
= \frac{n^2 \cdot 2^{-d}}{2}
\]

This is \( > 1 \) when \( n \gg 2^{(d+1)/2} \approx 2^{d/2} \)

The birthday paradox is the reason why hash function outputs are generally twice as big as you might naively expect; you only get 80 bits of security (w.r.t. CR) for a 160-bit output.

With some tricks, memory requirements can be dramatically reduced.
(3) "Weak collision resistance" (target collision resistance, 2nd pre-image resistance)

"Infeasible" given $x \in \mathcal{E}_0, \mathcal{E}_1^*$, to find $x' \neq x$ s.t. $h(x) = h(x')$.

Like CR, but one pre-image given & fixed.

(In ROM, can find $x'$ in time $\Theta(2^d)$ (as for OW, since knowing $x$ doesn't help in ROM).

(4) Pseudo-randomness

"$h$ is indistinguishable under black-box access from a random oracle"

(To make this notion workable, really need a family of hash functions, one of which is chosen at random. A single, fixed, public hash function is easy to identify...)

(5) Non-malleability

"Infeasible", given $h(x)$, to produce $h(x')$ where $x$ and $x'$ are "related" (e.g. $x' = x + 1$).

These are informal definitions...
Theorem: If \( h \) is CR, then \( h \) is TCR.
(But Converse doesn't hold.)

Theorem: \( h \) is OW \( \iff \) \( h \) is CR
(neither implication holds)
But if \( h \) "compresses", then CR \( \Rightarrow \) OW.

Hash function applications

1. Password storage (for login)
   - Store \( h(PW) \), not PW, on computer
   - When user logs in, check hash of his PW against table.
   - Disclosure of \( h(PW) \) should not reveal PW (or any equivalent pre-image)
   - Need OW

2. File modification detector
   - For each file \( F \), store \( h(F) \) securely
     (e.g. on off-line DVD)
   - Can check if \( F \) has been modified by recomputing \( h(F) \)
   - Need WCR (aka TCR)
     (Adversary wants to change \( F \) but not \( h(F) \).)
   - Hashes of downloadable software = equivalent problem.
3. Digital signatures ("hash & sign")

\( PK_A = \text{Alice's public key (for signature verification)} \)

\( SK_A = \text{Alice's secret key (for signing)} \)

**Signing:** \( \sigma = \text{sign} \left( SK_A, M \right) \) [Alice's sign on \( M \)]

**Verify:** \( \text{Verify} \left( M, \sigma, PK_A \right) \in \{ \text{True, False} \} \)

Adversary wants to forge a signature that verifies.

- For large \( M \), easier to sign \( h(M) \):

\( \sigma = \text{sign} \left( SK_A, h(M) \right) \) ["hash & sign"]

Verifier recomputes \( h(M) \) from \( M \), then verifies \( \sigma \).

In essence, \( h(M) \) is a "proxy" for \( M \).

- **Need CR** (Else Alice gets Bob to sign \( x \),

where \( h(x) = h(x') \), then claims

Bob really signed \( x' \), not \( x \).

- **Don't need OW** (e.g. \( h = \text{identity} \) is OK here.)