TOPIC: 6.857 - Network & Computer Security

DATE: 3/7/12

FILE UNDER:

PAGE: L9.1

Admin:

Pset #2 due Friday; email to 6857-staff@mit.edu

Project proposal presentations on Monday!

Today:

- Hash function properties
- Hash function applications

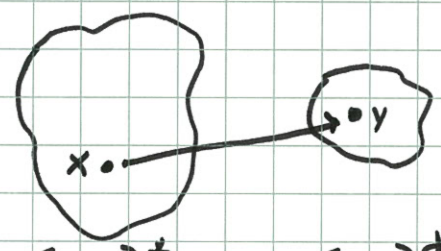
Hash function desirable properties:

OW

① "One-way" (pre-image resistance)

"Infeasible", given $y \in_R \{0,1\}^d$ to find

any x s.t. $h(x) = y$ (x is a "pre-image" of y)



$h: \{0,1\}^* \longrightarrow \{0,1\}^d$

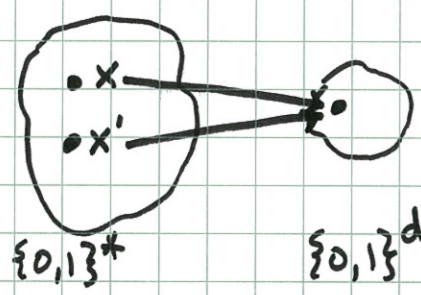
(Note that a "brute-force" approach of trying x's at random requires $\Theta(2^d)$ trials (in ROM).)

CR

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 $(x_1, x_2, \ldots)$ before a pair x_i, x_j colliding is found. (This is the "birthday paradox".))

Note that collisions are unavoidable since

$$|\{0,1\}^*| = \infty$$
$$|\{0,1\}^d| = 2^d$$

<u>Birthday paradox detail:</u>

If we hash $x_1, x_2, \ldots, x_n$ (distinct strings)

then

$$E(\#\text{collisions}) = \sum_{i \neq j} \Pr(h(x_i) = h(x_j))$$
$$= \binom{n}{2} \cdot 2^{-d} \qquad [\text{if } h \text{ "uniform"}]$$
$$\approx \frac{n^2 \cdot 2^{-d}}{2}$$

This is ≥ 1 when $n \geq 2^{(d+1)/2} \approx 2^{d/2}$

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.

TOPIC:

DATE:

FILE UNDER:

TCR

③ "Weak collision resistance" (target collision resistance, 2nd pre-image resistance)

"Infeasible", given $x \in \{0,1\}^*$, 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 to find x').)

PRF

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...)

NM

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 $\not\Longleftrightarrow$ 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.

TOPIC:

DATE:

FILE UNDER:

③ Digital signatures ("hash & sign")

PK_A = Alice's public key (for signature verification)

SK_A = Alice's secret key (for signing)

<u>Signing:</u> $\sigma = \text{sign}(SK_A, M)$ [Alice's sig on M]

<u>Verify:</u> $\text{Verify}(M, \sigma, PK_A) \in \{\text{True}, \text{False}\}$

Adversary wants to forge a signature that verifies.

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

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

  Verifier recomputes h(M) from M, then verifies σ.

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

- Need <u>CR</u> (Else Alice gets Bob to sign x, where $h(x) = h(x')$, then claims Bob really signed x', not x.)

- Don't need <u>OW</u> (e.g. h = identity is OK here.)