Admin:

Pset #1 due Mon 2/22 [Submit separate pdf for each problem.]
Pset #2 out Mon 2/22.

Today:

Cryptographic Hash Functions

Definition
Random Oracle Model
Properties: OW, CR, TCR, PR, NM
Applications

Readings:

Katz/Lindell Chapter 5
Paar/Pelzl Chapter 11
Ferguson/Schneier/Kohno Chapter 5
C-.
An ideal hash function: a "Random Oracle" (RO)

- Theoretical model - good intuitive guidance, but not achievable in practice

- Oracle ("in the sky")
  - receives input $x$ and returns $h(x)$
    for any $x \in \{0,1\}^*$, $|h(x)| = d$ bits.
  - On input $x$:
    - if $x$ not in book:
      - flip coin $d$ times to determine $h(x)$
      - record pair $(x, h(x))$ in book
    - else return $y$ where $(x, y)$ in book

- Gives random answers, but use of book ensures consistency.
**Random Oracle Model**

Many crypto schemes proved secure in ROM ("Random Oracle Model") which assumes existence of RO.

Then RO is replaced by hash function (e.g. SHA-256) in practice, which is hopefully "pseudorandom enough" that adversary can’t exploit any flaws in SHA-256.
Hash function desirable properties:

1. "One-way" (pre-image resistance)
   "Infeasible", given $y$, to find any $x'$ s.t. $h(x') = y$ ($x'$ is a "pre-image" of $y$)

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

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

In ROM, requires trying about $2^{d/2}$ $x$'s ($x_1, x_2, ..., x_{2^{d/2}}$) before a pair $x_i, x_j$ colliding is found. (This is the "birthday paradox".)
Note that collisions are unavoidable since
\[ |E_{0,18^*}| = \infty \]
\[ |E_{0,18^d}| = 2^d \]

Birthday paradox detail:

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

\[
E(\# \text{collisions}) = \sum_{i \neq j} \Pr( h(x_i) = h(x_j))
\]

\[ = \binom{n}{2} \cdot 2^{-d} \quad [ \text{if } h \text{ "uniform"} ]
\]

\[ \approx \frac{n^2 \cdot 2^d}{2} \]

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

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.
TCR

3. "Weak collision resistance" (target collision resistance and pre-image resistance)

"Infeasible" given $x \in \mathbb{Z}_p^*$, 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 far OOW, 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 \( \iff \) \( h \) is CR 
(neither implication holds)
But if \( h \) "compresses", then \( \text{CR} \Rightarrow \text{OW} \).

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 offline 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")

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

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

**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.)
4. Commitments

- Alice has value x (e.g., auction bid).
- Alice computes C(x) ("commitment to x") & submits C(x) as her "sealed bid".
- When bidding has closed, Alice should be able to "open" C(x) to reveal x.
- Binding property: Alice should not be able to open C(x) in more than one way!
  (She is committed to just one x.)
- Secrecy (hiding): Auctioneer (or anyone else) seeing C(x) should not learn anything about x.
- Non-malleability: Given C(x), it shouldn't be possible to produce C(x+1), say.

How:

\[ C(x) = h(r \| x) \quad r \in \mathbb{F}_2^{256} \]

To open: reveal r & x.

- Note that this method is **randomized** (as it must be for secrecy).

- Need: OW, CR, NM

  (really need more, for secrecy, as C(x) should not reveal partial information about x, even.)