Pset #3 due Monday (3/19)
Form your project groups! Choose a topic...

- Hash fn applications (continued)
- Hash fn construction (Merkle-Damgård)
- "Merkle-Damgård Revisited" (Coron, Dodis, Melinaud, Puniya)
- Floyd's "Two-Finger Algorithm" for finding collisions.
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:**

\[
C(x) = h(r \| x) \quad r \in_R \{0,1\}^{256}
\]

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.)
5. To authenticate a collection of \( n \) objects:

Build a tree with \( n \) leaves \( x_1, x_2, \ldots, x_n \) and compute authenticator node as \( f_n \) of values at children... This is a "Merkle tree":

Root is authenticator for all \( n \) values \( x_1, x_2, \ldots, x_n \)

To authenticate \( x_i \), give sibling of \( x_i \) & sibling of all his ancestors up to root.

Apply to: time-stamping data

Authenticating whole file system

\[
\text{Needs: CR}
\]
Hash function construction ("Merkle-Damgard" style)

- Choose output size \( d \) (e.g. \( d = 256 \) bits)
- Choose "chaining variable" size \( c \) (e.g. \( c = 512 \) bits)
  \[ \text{Must have } c \geq d; \text{ better if } c > 2 \cdot d \ldots \]  
- Choose "message block size" \( b \) (e.g. \( b = 512 \) bits)
- Design "compression function" \( f \)
  \[ f : \{0,1\}^c \times \{0,1\}^b \rightarrow \{0,1\}^c \]
  \[ \text{[} f \text{ should be OW, CR, PR, NM, TCR, \ldots ]} \]
- Merkle-Damgard is essentially a "mode of operation"
  allowing for variable-length inputs:

* Choose a \( c \)-bit initialization vector \( IV, c_0 \)
  \[ \text{[Note that } c_0 \text{ is fixed & public.]} \]
* [Padding] Given message, append
  - 10* bits
  - fixed-length representation of length of input

so result is a multiple of \( b \) bits in length:

\[ M = M_1, M_2, \ldots, M_n \ldots \text{ (n b-bit blocks)} \]
Then:
\[ h \left( \begin{array}{c} m_1 \\ c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{array} \right) \]

\[ h(m) = c_n \] truncated to d bits

**Theorem:** If f is CR, then so is h.

**Proof:** Given collision for h, can find one for f by working backwards through chain. \( \square \)

**Thm:** Similarly for OW.

**Common design pattern for f:**

\[ f(C_{i-1}, M_i) = C_{i-1} \oplus E(M_i, C_{i-1}) \]

where \( E(K, M) \) is an encryption function (block cipher) with b-bit key and c-bit input/output blocks.

(Davies-Meyer construction)
"Merkle-Damgard Revisited" ( Coron, Dodis, Malinaud, Puniya)

Is MD a "good" method?

What does this mean?

Suppose that \( f \) is a random oracle (fixed input length)

\[
f : \{0,1\}^b \rightarrow \{0,1\}^c
\]

Then is \( MD^f \) indistinguishable from a \( VIL \ RO \)?

(\( VIL = "\text{variable input length}" \))

Adversary has access to:

A. \( MD^f \) and also to \( f \) (\( f \) is \( \text{FIL} \))

B. \( RO \ h \) and also to \( g \) (\( h \) is \( VIL \& g \text{ FIL} \))

where \( g \) is constructed to bear same relation to \( h \) as \( f \) does to \( MD^f \) ("simulator")

Note: \( g \) may call \( h \), but doesn't see \( \text{Adv's} \) calls to \( h \).
Standard construction $\text{MD}^f$ fails (for $c=d$):

Can't build simulator $g$ to bear right relation to $h$
(i.e. so that $h$ appears to be $\text{MD}^g$)

Example of problem (message extension): (sketch)

$h$ & $g$ should satisfy

$$\text{MD}^g(m, \| m_2) = h(m, \| m_2) = g(g(IV, m_1), m_2)$$

$\text{Adv}$:

$$\begin{cases} \text{computes } u = h(m_1) \\ \text{computes } v = g(u, m_2) \\ \text{computes } w = h(m_1, m_2) \\
\begin{cases} \text{if } v = w : \text{answer "A world"} \\ \text{else: answer "B world"} \end{cases} \end{cases}$$

$\text{Adv}$ always right in A world, and almost always right in B world, since simulator $g$ doesn't know how to answer query $(\#)$. \[ \text{[It didn't see query for } u, \text{ so even though it can access } h, \text{ it doesn't have ability to figure out } m_1, \text{ and so reply to } (\#) \text{ in way that makes it consistent with } h(m_1, m_2).] \]
But, it is not hard to fix MD construction so it becomes "indistinguishable from RO" (given FÎL RO
f)

[technically this is called "indifferentiability"].

Four methods: (any work to fix MD)

1. Encode m to be "prefix-free" before applying MD:
   e.g. 0||m, ||0||m2, ||0||m3, ... ||1||mn
   \[= \] length of message in bits
   mn padded with 10*

2. Drop output bits:
   Let \( d = c/2 \). Drop \( c/2 \) bits of output.

3. NMAC construction:
   \[ g(MD^f(m)) \] \[ \text{[g indep. function} \]
   \[ \text{from } \{0,1\}^c \text{ to } \{0,1\}^d \]

4. HMAC construction:
   \[ MD^f(MD^f(m)) \]

* With such methods, it is then "safe" to treat (modified) MD^f as a RO (assuming f is indistinguishable from a FIL RO.)
Floyd's "Two-Finger" algorithm for finding collisions

Let $f : \{0,1\}^n \rightarrow \{0,1\}^n$

Pick random $x_0$.

Let $x_{i+1} = f(x_i)$ for $i = 1, 2, \ldots$, eventually loops:

$$x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_t \rightarrow x_{t+1} \rightarrow \ldots \rightarrow x_d \rightarrow \ldots \rightarrow x_{t+c-1}$$

Tail of length $c$

Cycle of length $c$

If $f$ is "random":

$$E(t) = E(c) = \Theta(\sqrt{n}) = \Theta(2^{n/2}) \quad \text{B.P.}$$

Two Finger alg:

1. Finger 1: $x_0, x_1, x_2, \ldots$ (single speed)
2. Finger 2: $x_0, x_2, x_4, \ldots$ (double speed)

Until $x_d = x_{2d}$

[Lemma: $d \equiv 0 \pmod{c}$]

Then:

- Finger 1 starts at $x_d, x_{d+1}, \ldots$ (both single speed)
- Finger 2: $\ldots, x_0, x_1, \ldots$

Until they collide at $x_t$

Previous step gives $f(x_{t-1}) = f(x_{t+c-1})$ collision!