Admin:
- Problem Set #1 due in Lecture 6
- Problem Set #2 out Lecture 6
- Next lecture by TA (secret sharing and bitcoin)
- Submit passwords (not real ones) for problem set #2

Project Ideas:
- “Format-Transforming Encryption”
- Shrimpton 2014 Real-World Crypto talk
- Also see https://fteproxy.org/

Today:
- Crypto hash functions: applications and constructions
  - Applications:
    o Signatures
    o Commitments
    o Merkle trees
    o Payword
    o Hash-cash
  - Construction:
    o Merkle-Damgard
    o Sponge function
Digital signatures ("hash & sign")

PKₐ = Alice's public key (for signature verification)
SKₐ = Alice's secret key (for signing)

Signing: \( \sigma = \text{sign} (SKₐ, M) \) [Alice's sign on M]
Verify: \( \text{Verify} (M, \sigma, PKₐ) \in \{ \text{True}, \text{False} \} \)

Adversary wants to forge a signature that verifies.
- For large M, easier to sign \( h(M) \):
  \[ \sigma = \text{sign} (SKₐ, h(M)) \] ["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 DW (e.g. \( h = \text{identity} \) is OK here.)
4) Commitments

- Alice has value $x$ (e.g. auction bid)
- Alice computes $C(x)$ ("commitment to $x$")
- $R$ 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:

$$C(x) = h(r||x) \quad r \in \mathbb{R}^{256}$$

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

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

\[
\text{root} = h(\text{value at } x) = h(h(\text{value at } y) || \text{value at } z)
\]

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

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

Need: CR
Hash-cash (by Adam Back)

- "Proof of work" by email sender

- Intent: reduce spam by making email "expensive" (computation)

- Sender must solve puzzle:
  
  find r s.t.

  $$h(sender, recipient, date, time, r)$$

  ends in 20 zeros

- include r in header as "proof of work/payment"

- each for recipient to verify

- takes about $$2^{20}$$ trials to solve for r

- doesn't work against bot-nets 😞
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)
  \[\text{Must have } c > d; \text{ better if } c > 2 \cdot d \ldots\]
- 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 should be OW, CR, PR, NM, TCR, ...]}
- Merkle-Damgard is essentially a "mode of operation"
  allowing for variable-length inputs:
  * Choose a c-bit initialization vector \(I_0\), \(c_0\)
    \[\text{[Note that } c_0 \text{ is fixed & public.]}\]
  * [Padding] Given message, append
    - \(10^b\) bits
    - fixed-length representation of length \(m\) of input
  so result is a multiple of \(b\) bits in length:
  \[M = M_1, M_2, \ldots, M_n \quad (n \text{ b-bit blocks})\]
Then:

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

**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 \( f \):**

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

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

(Davies-Meyer construction)
Keccak Sponge Construction

\[ d = \text{output hash size in bits} \in \{224, 256, 384, 512\} \]

\[ C = 2d \text{ bits} \]

State size = \(25w\) where \(w = \text{word size (e.g. } w = 64)\)

\[ C + r = 25w \]

\[ r \geq d \text{ (so hash can be first } d \text{ bits of } Z_0) \]

Input padded with \(10^d\) until length is multiple of \(r\)

\(f\) has 24 rounds (for \(w=64\)), not quite identical (round constant)

\(f\) is public, efficient, invertible function from \(\{0,1\}^{25w}\) to \(\{0,1\}^d\)

e.g.

\[ d = 256 \]

\[ C = 512 \]

\[ r = 1088 \]

\[ w = 64 \]