Admin:

Pset #1 due, & pset #2 out, on Monday 2/19.

Lectures by TA's next week (secret sharing & bitcoin).

Submit passwords (not real ones) to TA's, for pset #2, today.

Project idea:

"Format-Transforming Encryption."

Shrimpton 2014 Real-Word Crypto talk slides in our "top-secret" folder.

Also see https://ftp.proxy.org

Today:

Crypto hash funs: applications & constructions

Applications:
- Signatures
- Commitments
- Merkle trees
- Password
- Hash-cush

Construction:
- Merkle-Damgard
- Sponge function
Digital signatures ("hash & sign")

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

Sign: $\sigma = \text{sign}(SK_A, M)$ [Alice's sign on M]

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

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 $\sigma$.

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 $x$")
- Alice 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...

\[ C(x) = h(r|x) \quad r \in \mathbb{R}^{32} \]

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 \( x_1, x_2, \ldots, x_n \) & compute authenticator node as fn of values at children...

This is a "Merkle tree":

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

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

Need: CR
Hash-cash (by Adam Back)

- "Proof of work" by email sender
- Intent: reduce spam by making email "expensive" (computational)
- Sender must solve puzzle:

  \[ h(\text{sender, recipient, date, time, r}) \]

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 botnets 😞
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)
  [Must have c > d; better if c > 2d ...]
- Choose "message block size" b (e.g., b = 512 bits)
- Design "compression function" f
  \[ f : \{0,1\}^c \times \{0,1\}^b \rightarrow \{0,1\}^c \]
  [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 IV, c₀
    [Note that c₀ 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:
  \[ M = M₁, M₂, \ldots, Mₙ \]

```
\[ m \quad \quad 1000...0[\text{l}m] \]
```
Then:

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

**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)
Keccak

Keccak Sponge Construction

d: output hash size in bits ∈ \{224, 256, 384, 512\}
c = 2d bits
r = 2c
r ≥ d (so hash can be found in 1 block of r)

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

f has 24 rounds (for \(w = 64\)), not quite identical (round constant)
f is public, efficient, invertible function from \(E_0, \ldots, E_{15}\) to \(E_0, \ldots, E_{15}\)