Admin:
Pset #1 due today
Pset #2 out today

Today:
Cryptographic Hash Functions II ("Merkle Damgård")
  • Merkle trees
  • Puzzles & brute-force search
  • PH crypto based on puzzles (Merkle puzzles)
  • Hash function construction methods
    • Merkle-Damgård
    • Keccak

Readings:
Katz/Lindell:  Chapter 5
Paar/Peltz:  Chapter 11
Ferguson:  Chapter 5

News:
Lenovo "Superfish"
Citizenfour wins Oscar for best documentary (HBO tonight at 9 pm)

Project idea:
Do security analysis of "OpenWrt" router software
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 fn 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

Needs: CR

Used in bitcoin...
Puzzles & Brute-Force Search

\[ h : \{0,1\}^* \rightarrow \{0,1\} \]

If \( h \) is well-modeled as a random oracle, inverting \( h \) requires \( 2^d \) steps on average:

Given \( y \in \{0,1\}^d \), adversary can do no better than trying \( x_1, x_2, \ldots, \) until he finds \( x_i \) s.t. \( h(x) = y \). Probability that \( h(x_i) = y \) is \( 2^{-d} \) (by ROM), so expected trials needed is \( 2^d \). Brute-Force

To make a "puzzle," choose \( d \) to be "not too large."

E.g. \( h(x) = \text{sha256}(x) \mod 2^d \)

where \( d = 40 \)

Takes \( 2^{40} \) steps to solve, on average.

Note: special-purpose chips & boards can do \( \approx 2^{40} \) hashes/second, so this is maybe a "one-second puzzle" for such a device.

Puzzle difficulty is controllable (by choosing \( d \))

Easy to create many puzzles: \( h_k(x) = h(k || x) \)

so one puzzle for each parameter \( k \).

Puzzle spec = \((k, d, y)\) want \( x \) s.t. \( h_k(x) = y \)

Puzzle creator knows solution (computes \( y \), given \( x \))
Hash cash (Adam Back, 1997)

- Anti-spam measure
- Requires sender to provide "proof of work" ("stemp")
- Email without POW or from sender on whitelist is discarded.

POW:

solve puzzle $h(k, r)$ ends in 20 zeros
where $k = \text{sender} || \text{receiver} || \text{date} || \text{time}$

$r = \text{variable to be solved for}$

- Include $r$ in header as POW
- Easy for receiver to verify payment (POW)
- Takes $x 2^{20}$ trials to solve
- Doesn't work well against botnets 😞
Merkle Puzzles (1974)

- First "public key" system. (Really: key agreement)

Alice <--- Eve <--- Bob

How can Alice & Bob agree on a key k over channel, while Eve is eavesdropping?

Parameters:
- \( n \) = # of puzzles
- \( D = 2^d \) = puzzle difficulty

1. Bob makes \( n \) puzzles of difficulty \( D \)

\[ P_1, P_2, \ldots, P_n \]

& sends them all to Alice (& Eve)

2. Alice picks random \( i \) (\( 1 \leq i \leq n \)) & solves \( P_i \)

(saves D for Alice)

3. Alice lets Bob know (but not Eve) which one she has solved, e.g., by sending \( h(K_i) \)

4. Further communication protected with session key \( K_i \).

Time for good guys = \( O(n) + O(D) \)

Bob

Alice

Time for Eve = \( O(n \cdot D) \)

For \( n = D = 10^9 \), "almost practical"!
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" $f$
  $$ f : \{0,1\}^c \times \{0,1\}^b \rightarrow \{0,1\}^c $$
  [If 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 $IV$, $c_0$
  [Note that $c_0$ is fixed & public.]
  * [Padding] Given message, append
    - $10^* \text{ bits}$
    - fixed-length representation of length of input
  so result is a multiple of $b$ bits in length:
    $$ M = M_1 \, M_2 \, \ldots \, M_n \quad (n \, b\text{-bit blocks}) $$
Then: $h \{ \begin{array}{c}
I \leftrightarrow f \leftrightarrow c_1 \\
M_1 \leftrightarrow f \leftrightarrow c_2 \\
M_2 \leftrightarrow f \leftrightarrow c_3 \\
M_3 \leftrightarrow f \leftrightarrow \ldots \\
M_n \leftrightarrow f \leftrightarrow \text{c}\text{n}
\end{array} \}

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. \( \Box \)

**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)
Typical compression function (MD5):

- Chaining variable & output are 128 bits = 4 x 32
- IV = fixed value
- 64 rounds; each modifies state (in reversible way) based on selected message word
- Message block b = 512 bits considered as 16 32-bit words
- Uses end-around XOR too around entire compression fn (as above)

\[ g(x, y, z) = \begin{cases} 
  x, & \text{if round is even} \\
  x \oplus y \oplus z, & \text{if round is odd}
\end{cases} \]

Xiaoyun Wang discovered how to make collision for MD4, MD5... ("Differential Cryptanalysis")
Keccak = SHA-3

d = 256
c = 512
r = 1088
w = 64

Keccak Sponge Construction

C = \text{output hash size in bits} \in \{224, 256, 384, 512\}

r = \text{state size in bits} = 25w

C + r \geq d \text{ (so hash can be first d bits of E0)}

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

f \text{ is public, efficient, invertible function from } \{0, 1\}^{25} \text{ to } \{0, 1\}^{25}