### Admin:

Final projects!

### Today:

- Merkle Trees
- Merkle Puzzles
- PK crypto based on Merkle puzzles
- Constructions
  - Merkle-Damgard
  - Reccall (SHA-3)

### Readings:

- Katz & Lindell: Chapter 5
- Paar & Pelzl: Chapter 11
- Ferguson: Chapter 5
- Wikipedia: SHA-3
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":

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

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

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

Apply to: time-stamping data
authenticated whole file system

Need: CR

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

Want to create puzzle with solution known to creator that requires (on average) a fixed amount of work to solve.

Let \( h : \{0,1\}^k \rightarrow \{0,1\}^d \) be a crypto hash fn (e.g. SHA-256 with \( d = 256 \)).

The "puzzle" will be to invert \( h \), i.e. solve \( h(x) = y \) for \( x \) given \( y \).

To make this a puzzle, we restrict \( x \) to be in a known set \( S \) of possible solutions. E.g. \( S = \{0,1\}^s \) for \( s = 40 \).

To create a puzzle, pick \( x \in S \) at random, compute \( y = h(x) \).

Difficulty of solving \( 2^{s/2} \) by brute-force search.

If \( s < d \) there will be no "false solutions"—no collisions.

Can create multiple (identical) puzzles \((k, y)\) means solving \( h(k \| x) = y \) for \( x \in S \).

Puzzle spec is \((h, k, s, y)\).

Puzzle creator knows solution

Can also have puzzles where creator doesn't know solution with truncated hashes

\( h : \{0,1\}^k \rightarrow \{0,1\}^s \)

try \( x \) at random until \( h(x) = y \)

Expected work is \( 2^s \).
**Hash cash (Adam Back, 1997)**

- Anti-spam measure
- Requires sender to provide "proof of work" ("stomp")
- Email without POW or from sender on whitelisted

**POW:**

- Solve puzzle $h(k, r)$ ends in 20 zeros
  - $k = \text{sender} || \text{receiver} || \text{date} || \text{time}$
  - $r = \text{variable to be solved for}$
- Include $r$ in header as POW
- Easy for receiver to verify, hard to compute (POW)
- Takes $2^{20}$ trials to solve
- Doesn't work well against botnets

$h(k,r) = h(k||r)$
Modular puzzles

- First "public key" system (really: key agreement)

\[ \text{Alice} \quad \text{Eve} \quad \text{Bob} \]

Eve is passive eavesdropper.
How can Alice & Bob agree on a key?

Use puzzles (with restricted domain, so have unique solns)
\( n = \# \text{puzzles of difficulty } 2^{s-1} = D \) each

1. Bob chooses \( n \) values \( x_1, x_2, \ldots, x_n \) from \( S = \{0, 1\}^s \)
   
   Bob computes \( y_i = h(i || x_i) \)

   Bob sends \( \left( y_i, E_{x_i}(K_i) \right) \) to Alice for \( i = 1, 2, \ldots, n \), where \( K_i \in \{0, 1\}^{s+3} \)

2. Alice picks random \( i \) from \( \{0, 1, 2, \ldots, n\} \)
   Alice solves \( P_i \) for \( x_i \)
   * decrypts to obtain \( K_i \)
   * sends \( h(K_i) \) to Bob

3. Bob & Alice use \( K_i \) to communicate secretly from then on.

Time for good guys:
\[
\text{Bob: } O(n) + O(D) \quad \text{Alice: } O(D)
\]

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

For \( n = D = 10^9 \), "almost practical."
Hash function construction ("Merkle-Damgård" 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" $f$
  $f: \mathbb{Z}_2^{13} \times \mathbb{Z}_2^{13} \rightarrow \mathbb{Z}_2^{13}$
  [ $f$ should be OW, CR, PR, NM, TCR, … ]
- Merkle-Damgård 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
    - $10^k$ bits
    - fixed-length representation of length of input
    so result is a multiple of $b$ bits in length:
  $$M = M_1 \cdot M_2 \cdots M_n \ldots (n \ b\text{-bit blocks})$$
\[ h(m) = c_n \text{ truncated to } d \text{ bits} \]

**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 6-bit key and 6-bit input/output blocks.

(Davies-Meyer construction)

\( E \) based on "ARX" instruction:

- \( A = \text{addition} \quad x \leftarrow x + y \)
- \( R = \text{rotation by fixed amt} \quad x \leftarrow x \lll 5 \)
- \( X = \text{xor} \quad x \leftarrow x \oplus y \)

(Also "AND" and "OR")

Constant time (no side-channel attacks)
Typical compression function (MD5):

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

Kiyumi Wong discovered how to make collision for MD4, MD5... "Differential cryptanalysis"

\[
g_k(y, z) = \begin{cases} 
xy 0 & y = 0, z = 1 \\
xz 0 & y = 1, z = 0 \\
xz 0 & y = 0, z = 1 \\
x0 0 & y = 1, z = 0 \\
\end{cases}
\] depending on round
Keccak = SHA-3

Keccak-SHA3-256

Also: SHA3-224 = 384 - 512

SHA-3 "sponge construction"

"squeeze"
Thm: Can find SHA-3 collision in time $2^{c/2}$

Pf: Find 2 inputs that collide on capacity $c$ bits

(time = $2^{c/2}$ by birthday paradox)

Use ability to arbitrarily tweak $1 \leq r$ bits to make collision.