Today:

Cryptographic Hash Fns II (aka "Merkle Day")
- Merkle Trees
- Merkle Puzzles
- PK crypto based on Merkle puzzles
- Constructions:
  - Merkle-Damgard (e.g. for MD5)
  - Keccak (SHA-3)

Readings:
- Katz & Lindell: Ch. 5
- Paar & Pelzl: Chapter 11
- Ferguson: Chapter 5

Admin:

(nothing)
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":

![Merkle tree diagram]

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

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

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

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

Let \( h : \{0,1\}^* \rightarrow \{0,1\}^d \) 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 \( x \) is \( 1/2^s \) by brute-force search.

If \( s \ll d \) there will be no "false solutions" - no collisions.

Can create multiple (keyed) puzzles \((k, y)\) means solving \( h(k \mid x) = y \) 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.

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

Try \( x \) at random until \( h(x) = y \)
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 $k = \text{sender} \| \text{receiver} \| \text{date} \| \text{time}$

$r$ = variable to be solved for

- Include $r$ in header of POW
- Easy for receiver to verify payment (POW)
- Takes $x \cdot 2^{20}$ trials to solve
- Doesn't work well against botnets 😞
**Magic puzzles**

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

  Alice \[\rightarrow\] Eve \[\leftarrow\] Bob

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

Use puzzles (with restricted domain, so have unique solns)

\[n = \# \text{puzzles of difficulty } 2^{s-1} = D\]

1. Bob chooses \(n\) values \(x_1, x_2, \ldots, x_n\) from \(S = \{0,1\}^s\)
   - Bob computes \(y_i = h(c \| x_i)\)
   - Bob sends \((y_i, E_{x_i}(K_i))\) to Alice for \(1 \leq i \leq n\), where \(K_i \in \{0,1\}^{256}\)

2. Alice picks random \(i\) from \([n]\) = \(\{1, 2, \ldots, n\}\)
   - 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.

   \[\text{Time for good guys} = \frac{\Theta(n)}{\text{Bob}} + \frac{\Theta(D)}{\text{Alice}}\]

   \[\text{Time for Eve} = \Theta(n \cdot D)\]

   For \(n = D = 10^4\), "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 $c < 2d$; 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, $c_0$
    [Note that $c_0$ 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:
    $$M = M_1 M_2 \ldots M_n \ldots (n b$-bit blocks)$$

```plaintext
| m | 1000...01m1 |
```
\[ h \left( \begin{array}{c}
\text{IV} \\
\text{c}_0
\end{array} \right) \rightarrow f \rightarrow \text{c}_1 \rightarrow f \rightarrow \text{c}_2 \rightarrow f \rightarrow \cdots \rightarrow f \rightarrow \text{c}_n
\]

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 \( f \):**

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

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 block word
- Message block $b = 512$ bits considered as 16 32-bit words
- Uses end-around XOR too around entire compression fn (as above)

Xiaoyun Wang discovered how to make collision for MD4, MD5 ("Differential cryptanalysis")

$g(x, z) = \begin{cases} x \oplus y \oplus z \\ x \oplus z \oplus y \\ x \oplus y \oplus z \\ y \oplus x \oplus z \end{cases}$ depending on round
Keeccak Sponge Construction

- $d =$ output hash size in bits $\in \{224, 256, 384, 512\}$
- $c = 2d$, bits
- state size $= 25w$ where $w =$ word size (e.g., $w = 64$)
- $c + r = 25w$
- $r \geq d$ (so hash can be first $d$ bits of $E_0$)

Input padded with $10^r$ 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 $\{0,1\}^{25w}$ to $\{0,1\}^{25w}$

Keeccak = SHA-3