Computer Security

Reagan Shirk

October 6, 2020

Cryptographic Hash Functions

Hash Functions

- Takes a message of arbitrary length and turns it into a fixed-length short message
- This is also known as:
  - message digest
  - one-way transformation
  - one-way function
  - hash
- it's one way, you can generate a value of a hash message and can't use it to recover the original message
- Output is normally 128 or 160 bits
- Length of H(m) is much shorter than m

Desirable Properties

- Performance: it's easy to compute H(m)
- One-way: Given H(m) but not m, it's infeasible to find m
- Weak collision resistance (free): Given H(m), it's infeasible to find m' such that H(m) = H(m')

Length of Hash Image

- Why do we have 128 or 160 bits in the output? Why not 50, 60, 80, 200?
  - Tradeoff - making it smaller is too easy, making it larger is too complex

Birthday Paradox

- What is the smallest group size k such that the probability that at least two people in the group have the same birthday is greater than 0.5?
  - Assume 365 days in the year, all birthdays are equally likely
  - k = 23
  - $Q(365, k) = \frac{365!}{(365-k)!365^k}$
  - $P(365, k) = 1 - Q(365, k)$
- Sharing a birthday = collision. More amount of people = greater chance of collision.
- Generalization of this paradox:
  - Given:
    - a random integer with uniform distribution between 1 and n, and a selection of k instances of the random variables,
  - What is the least value of k such that there will be at least one duplicate with a probability ≥ 0.5?
  - $P(n, k) = 1 - \frac{n!}{(n-k)!n^k} \approx 1 - e^{-k \times \frac{k-1}{2n}}$
  - For a very large k, we have $k = \sqrt{n}$
- Implication for Hash Functions

- For a hash function of length m, the hash value of an arbitrary input message is randomly distributed between 1 and 2^m
- What is the least value of k such that if we hash k messages, the probability that at least two of them have the same hash is larger than 0.5?
  - $k = 2^{\frac{m}{2}}$
  - For the birthday attack, this would be $m \geq 128$

Hash Function Applications

File Authentication

- We want to detect if a file has been changed after it was stored
- Method:
  - compute a hash $H(F)$ for a file F
  - Store $H(F)$ separately from F
  - Can tell at any time if F has been changed by computing $H(F')$ and comparing it to $H(F)$
    - Will work unless there is a collision, but since hash function has strong collision resistance you'll probs be good
  - Why not just keep a duplicate copy of the file?
    - Hash is smaller, takes more memory to save a copy of the file
    - Hash = compression but compression $\neq$ hash

User Authentication

- Alice wants to authenticate herself to Bob, assuming they already share a secret key K
- Method:
  - Alice will choose a random number R and compute the hash with the secret key and R - $Y = H(R|K)$
  - Bob is given Y and he verifies that $Y = H(R|K)$
- Why not just send K in plaintext or $H(K)$? What's the purpose of R?

Commitment Protocols

- Alice and Bob wish to play the game of odd or even over a network
  - Alice picks a number X
  - Bob picks another number Y
  - Alice and Bob simultaneously exchange X and Y
  - Alice wins if $X + Y$ is odd, otherwise Bob wins
  - There's an issue, super easy for cheating
- Try this:
  - Alice must commit to X before Bob will send Y
  - Alice picks X and computes $Z = H(X)$
  - Bob gets Z and picks Y
  - I'll come back later he changed slides
  - What problems are there if the set of possible values for X is small?
    - Collision
    - You can manually increase the space (salt?) to decrease collisions

Message Encryptions

- Assume Alice and Bob share a secret key K but don't want to just use encryption of the message K
  - Don't want to spend all of the memory/computational power
- Alice sends B a random number R_1
- Bob sends Alice a random number R_2
  - Both numbers are encrypted
- You have a one time pad of the hashes chaining into each other, XOR with plaintext to get ciphertext

Message Integrity