SJSU SAN JOSÉ STATE UNIVERSITY

Lesson 8 – Hash Functions

Yan Chen

CS166 Fall 2024

- Public key crypto provides integrity and non-repudiation when used for digital signature
  - Since only signee knows his/her private key
  - No one else can “encrypt” message use signee’s private key
  - If signed using public key crypto, signee cannot deny after!
- Always remember public key is public!
  - A message can be signed and sent by different people!
  - A message can be signed and encrypted by different people!
  - Signee, encrypter, sender can all be different!

PKI

- Public Key Infrastructure (PKI): the stuff needed to securely use public key crypto
  - Generate and manage the keys
  - Include certificate authority (CA), a trusted 3rd party (TTP) to create and sign digital certificate for users
- Digital certificate: contains user's name and public key
- Verify CA's signature to verify integrity & identity of owner of corresponding private key
- PKI can use different "trust models"
  - Monopoly, oligarchy, anarchy

- A signed message may be costly to compute & send
  - Suppose Alice send Bob a signed message $S = [M]_{Alice}$
  - Alice also needs to send Bob M so Bob can verify $M = \{S\}_{Alice}$
  - If M is big, it means Alice needs to send 2 big messages!
- Hash function: "map" big M to smaller "fingerprint" of M
  - Notation: h(M), also called hash, or digest
  - Then Alice signs h(M) instead and sends M and $S = [h(M)]_{Alice}$
  - Bob verifies that $h(M) = \{S\}_{Alice}$
- Collision: different M map to same h(M)
  - Must happen...since input space is larger than output space

- Use birthday problem to understand collision
- "Lighter" version of birthday problem: Suppose N people in a room, how large must N be before the probability "at least one person's birthday is on X" is ≥ 50%?
  - For example, X can be Sep. 19
  - Ignore the leap years and all dates are equally likely (1 / 365)
  - Approach: get P(A') first, then P(A) = 1 – P(A')

    A': no one's birthday is on X, and $P(A') = \left(1 - \frac{1}{365}\right)^N = \left(\frac{364}{465}\right)^N$

  - Solve $P(A) = 1 - \left(\frac{364}{365}\right)^N = 50\%$, get N ≈ 253

- Birthday Problem: how many people must be in a room before probability is ≥ 50% that at least 2 people have the same birthday?
  - Similar approach: find A' – all people have different birthday
  - Solve $P(A) = 1 - P(A') = 1 - \frac{365}{365} * \frac{364}{365} * \cdots * \frac{365-N+1}{365} = 50\%$
  - Get N ≈ 23
  - Or different approach, since we are comparing all pairs x and y

    ${}_N C_2 = \frac{N!}{2!(N-2)!} = \frac{N(N-1)}{2} = 365$, get N ≈ 19
  - i.e., if have x outputs, compute sqrt(x) inputs to find "collision"

- Apply to hash function...suppose h(M) has n bits
  - Then there are 2^n possible hashes (outputs)
  - So, if hash about sqrt(2^n) = $2^{n/2}$ inputs, will find 1 collision
  - How many inputs need be hashed to find m collisions?
- If find a collision, the hash is broken
  - If Trudy finds M' ≠ M such that h(M) = h(M'), then she can replace (M, S) with (M' and S), and Bob can't detect that
  - So, need $2^{n/2}$ tries to break a n-bit h(M) by "brute-force"
  - Suppose h(M) has n bits, how may bits should a symmetric key have to achieve same level of security?

- A secure crypto hash function h(x) must be...
- Deterministic
  - Same x always result in same h(x)
- Compressive
  - The output length is small
- Efficient
  - h(x) is easy to compute for any x
- One-way
  - Easy: given x, compute h(x)
  - Hard: given h(x), find x

- A secure crypto hash function h(x) must have...
- Avalanche effect
  - Small change in x (1 bit) result in huge change in h(x)
- Weak collision resistance ("lighter" birthday problem)
  - Given x and h(x), hard to find $y \neq x$ such that $h(y) = h(x)$
  - That is, hard to generate a message that result in same h(x)
- Strong collision resistance (birthday problem)
  - Hard to find any x and y, with $x \neq y$ such that $h(x) = h(y)$
  - Lots of collisions exist, but hard to find any
  - If h(x) has n bits, how many collisions if we hash x inputs?

- Data $X = (X_1, X_2, X_3, \ldots, X_n)$, each X_i is a byte (8 bits)
  - Define $h(X) = (X_1 + X_2 + X_3 + \ldots + X_n) \bmod 256$
- Is this a secure cryptographic hash?
  - No! Easy to find collisions
  - Example:

    X = (10101010, 00001111), h(X) = 10111001

    Y = (00001111, 10101010), h(Y) = 10111001

    That is, $X \neq Y$, but $h(X) = h(Y)$
  - Other examples? h(x)

- Data $X = (X_1, X_2, X_3, \ldots, X_n)$, each X_i is a byte (8 bits)
  - Define $h(X) = [nX_1 + (n-1)X_2 + (n-2)X_3 + \ldots + X_n] \bmod 256$
- Is this a secure cryptographic hash?
  - No! Not easy as example 1, but still not so hard...
  - Example:

    $X = (00000001, 00001111)$, $h(X) = 00010001$

    $Y = (00000000, 00010001)$, $h(Y) = 00010001$

    That is, $X \neq Y$, but $h(X) = h(Y)$
  - But this hash is used in a non-crypto application rsync

- Cyclic Redundancy Check (CRC)
  - Essentially, CRC is the remainder in a long division calculation
  - Good for detecting burst errors
  - Such random errors unlikely to yield a collision
- But can't use in crypto!
  - Since it's easy to construct collisions (strong collisions)
- CRC has been mistakenly used where crypto integrity check is required (e.g., WEP)

- Recall: crypto hash function must be secure and fast
  - The message is hashed in blocks
  - The hash function consists of some number of rounds
  - Secure: “avalanche effect” after few rounds
  - Fast: simple rounds
  - Analogous to design of block ciphers
- Will briefly introduce the following crypto hash functions
  - Popular ones: MD5 & SHA-1
  - “Fast and strong”: Tiger hash

- MD5 & SHA-1 are popular hash functions
  - Both broken now, so not used for encryption any more
  - But still widely used for integrity or other non-crypto apps
- MD5: Message-Digest algorithm (version 5)
  - Invented by Rivest (who also invented RSA, RC4, RC6, etc.)
  - 128-bit output
- SHA-1: Secure Hash Algorithm (version 1)
  - Was U.S. government standard, inner workings similar to MD5
  - 160-bit output
  - Replaced by SHA-3 now

- Tiger Hash
  - Designed by Ross Anderson and Eli Biham
  - Fast and strong
  - Optimized for 64-bit processors
  - Can be replacement for MD5 or SHA-1
- A high-level view of the algorithm
  - Used lots of ideas from block ciphers
  - Input divided into 512-bit blocks (same as MD5 and SHA-1)
  - 24 inner rounds
  - Intermediate vales & final output are all 192 bits

- A high-level view of inner rounds
  - 192 bits breaks into three 64-bits words, a, b, and c
  - 4 S-boxes: each maps 8 bits to 64 bits

    Claimed that each input bit affects a, b and c after 3 rounds
  - Key schedule: small change in message affects many bits of intermediate hash values
  - Multiply: ensure that input to S-box in one round mixed into many S-boxes in next
  - S-boxes, key schedule and multiply together ensures the avalanche effect

- HMAC: hashed MAC, used for integrity
  - Recall: MAC (Message Authentication Code) is computed as the last block of CBC, used for integrity
  - A "keyed" hash –a key is required
- HMAC algorithm
  - Let B be the block length of hash, in bytes
  - ipad= 0x36 repeated B times
  - opad= 0x5C repeated B times
  - HMAC(M,K) = h(K $\oplus$ opad, h(K $\oplus$ ipad, M))

- Suppose Alice, Bob, and Charlie are bidders
  - Alice plans to bid A, Bob bids B, and Charlie bids C
  - If bids revealed, Trudy can place max(A, B, C) + 0.1
- A possible solution?
  - Alice, Bob, Charlie submit hashes h(A), h(B), h(C)
  - All hashes received and posted online
  - Then bids A, B, and C submitted and revealed
  - Hashes don't reveal bids (one way)
  - Can't change bid after hash sent (collision)
  - Any flaw?

- Reduce spam email by requesting sender to prove they did some "work" ("proof-of-work") before accepting email
  - Here, "work" == CPU cycles
  - This approach will NOT eliminate spam
  - Instead, make spam more costly to send
  - Then the amount of email can be sent is limited
- A high-level view of the algorithm
  - Let M = email message, R = value to be determined, T = current time
  - Sender must determine R so that h(M, R, T) = (00…0, X)

- A high-level view of the algorithm (continued)
  - That is, initial N bits of hash value are all zero
  - The calculation is done by "brute-force"
  - Sender: sends (M, R, T)
  - Recipient: verify that h(M, R, T) begins with N zeros
  - "Hard" work for sender: on average 2^N hashes
  - "Easy" work for recipient: always 1 hash
- Choose N so that
  - Work acceptable for normal amounts of email
  - Work is too high for spammers

- Software
  - Buffer overflow
  - Incomplete mediation
  - Race conditions

- Birthday problem
- Crypto hash function properties
  - Deterministic, compressive, efficient, one-way
  - Avalanche effect, weak/strong collision resistance
- Crypto hash function examples
  - MD5, SHA-1, Tiger hash
- Crypto hash function usages
  - HMAC, online bids, spam reduction

- Suppose that a secure cryptographic hash function generates hash values that are n bits in length. Explain how a brute force attack could be implemented. What is the expected work factor?
- Consider a CRC that uses the divisor 10011. Find two collisions with 10101011, that is, find two other data values that produce the same CRC checksum as 10101011.
- Does a MAC work as an HMAC? That is, does a MAC satisfy the same properties that an HMAC satisfies?

References

- Stamp, Mark, “Information Security, Principles and Practice, 2nd ed.,” Wiley, New Jersey, USA, 2011