# Cryptographic Hashing Functions

Yusuf Ziya Uzun - CMP5121

Understanding Cryptography
by
Christof Paar and Jan Pelzl Chapter 11

# Hashing Functions Revisited

- Basic Properties:
  - Generates fixed length output from given arbitrary data.
  - For given input, always produces same output
  - Produces well separated hash space
- Helps to:
  - Reduces time complexity of input elements

- Hashing functions have very broad usage areas in Computer Science.

# Hashing Functions: Collision

- A collision happens when given two different input produces same output by the Hashing function.
- Collision:
  - For x, y e Input and H(x), H(y) e Output => x!=y & H(x)=H(y)

- Collisions are inevitable if Hash function is not injective.
- Most of times Hash functions are not injective, so there is always a probability to collision happens.

# Hashing Functions: Example

- Lets consider hashing IPv4 addresses with this function:
  - [x].[y].[z].[t] → ([x] << 8) + [y]
- For example, 1.2.3.4 and 1.2.100.100 both addresses produces same hash outputs:
  - Hash outputs are between 1 and 2^16
  - (1 << 8) + 2 = 258
  - All IP addresses starts with 1.2 block will be mapped to 258 hash output.
- [x].[y].[z].[t] → ([x] << 24) + ([y] << 16) + ([z] << 8) + [t]
  - This function produces a perfect hash output(injective) and it has no collision.
  - Hash outputs between 1 and 2^32

# Cryptographic Hashing Functions

- Basically it is such a hash function that gives us these properties:
- Computationally Efficient
  - If it is not fast enough to compute hash, we cannot use it everywhere.
  - Time - complexity trade-off must be in understandable level.
- Collision Resistant
  - It must not produce same hash outputs (theoretically impossible)
- Unpredictable
  - Infeasible
    - To generate message from hash
    - To modify message without modifying hash (one-wayness)
    - To find two different message with same hash
  - Randomness
    - Almost same messages must produce two very different looking hashes.

# Preimage Resistance

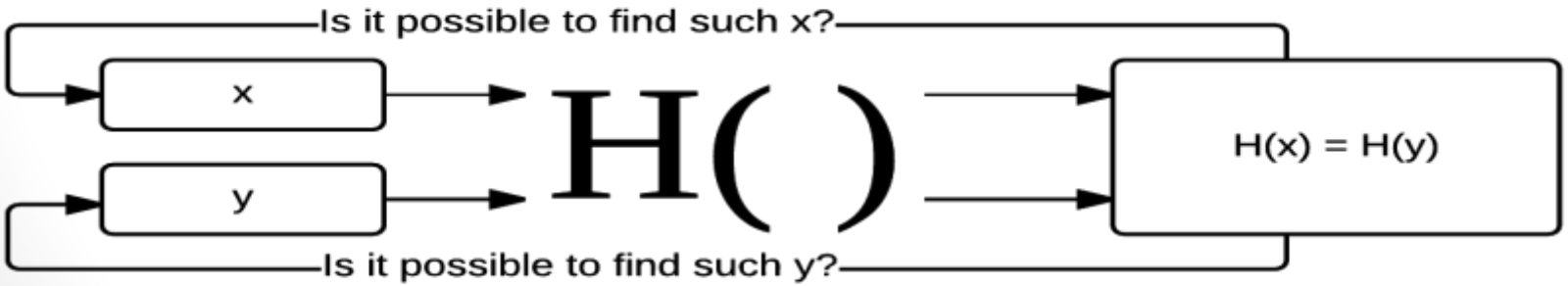
- Another name: One-wayness
- Given a hash output z it must be computationally infeasible to find an input message x such that z = H(x).
- Why important?
  - It only verifies the message, cannot parse.

# Second Preimage Resistance

- AKA: Weak Collision Resistance
- Given x, and thus H(x), it is computationally infeasible to find any y such that H(x) = H(y).
- Why important:
  - It is very obvious x and y are two different message but their digest(hash) are same output!
  - Think you send correct message x through channel and it replaced by somebody else to message y.

# Collision Resistance

- It is computationally infeasible to find any pairs x != y such that H(x) = H(y)
- Why important:
  - Birthday attack shows us we need bigger hash space to reduce collision attack probabilities.



# Applications

- Digital Signing
- Message Authentication
- CSPRNG
- Password Security
- Encryption

# Some Algorithms

- MD4 Family (Message Digest)
  - MD4 (128 bit)
  - MD5 (128 bits)
  - MD6 (up to 512 bits)
- SHA Family (Secure Hashing)
  - SHA-1 (160 bits)
  - SHA-2 (variants: SHA-256 and SHA-512)
  - SHA-3 (KECCAK: arbitrary, variants: up to 512 bits)

- crypt, bcrypt, scrypt, PBKDF2(details)

# Pigeons and Butterflies!

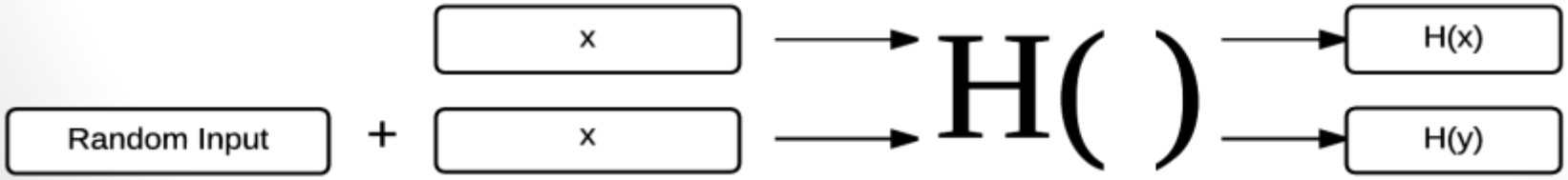
- Pigeonhole Rule (Collision Resistance Problem)
  - Input space: Infinite space
  - Digest(output) space: Finite space
  - Always smaller Cardinality than input space
  - Cannot find any injective function
  - More powerful: Birthday attack! (Collision Attack)
- Avalanche Effect (Infeasibility Problem)
  - Butterfly effect
  - Proof of Unpredictability
  - Block ciphers Diffusion Method

# Some Attacks

- Collision
  - x != y and H(x) == H(y)
- Birthday
  - Randomly tried collision attack over probabilities.
- Brute-force
- Dictionary
- Rainbow tables

# Defense: Salting the Hash

- Since the hashes one way, there is no way back to produce message from hash.
- So hash attacks mostly brute-force and it's derivatives.
- Attacker can pre-compute many hash outputs.
- This causes to crack hashes with lookup tables very quickly.
- As a solution, we can add some random input to message before it is hashed.
- This way we can prevent rainbow table and brute-force attacks.



# Demo Code

- Hashing with salt
- Iterating hash function # of times (key stretching)

# Thank you!

- References:
  - Understanding Cryptography by Christof Paar and Jan Pelzl, Chapter 11
  - http://en.wikipedia.org/wiki/Cryptographic_hash_function
  - https://crackstation.net/hashing-security.htm
  - http://www.dijksterhuis.org/creating-salted-hash-values-in-c/
  - Other Internet Materials