Lec4_Cryptography III

Hash Function

y=H(x)

- unlimited input size, and fixed output size
- public-known algorithm - no secret, same input → same output

computationally infeasible:

- large input space → cannot find input x, if space small, not hold

Properties

- pre-image resistance
  - given hash y is **computationally infeasible** to find x: $h(x)=y$
  - **one-way**: cannot reverse hash
- collision resistance
  1. either x(y) already chosen, find x′ (weak collision resistance)
     - computationally infeasible to find any value $x \neq x'$ $h(x) = h(x')$
     - cannot alter a document and have its hash stay the same
  2. both x and x'(and y) can be chosen (strong collision resistance)
     - computationally infeasible to find any two x and x'
     - cannot create 2 document which have same hash
- output looks random
- avalanche effect

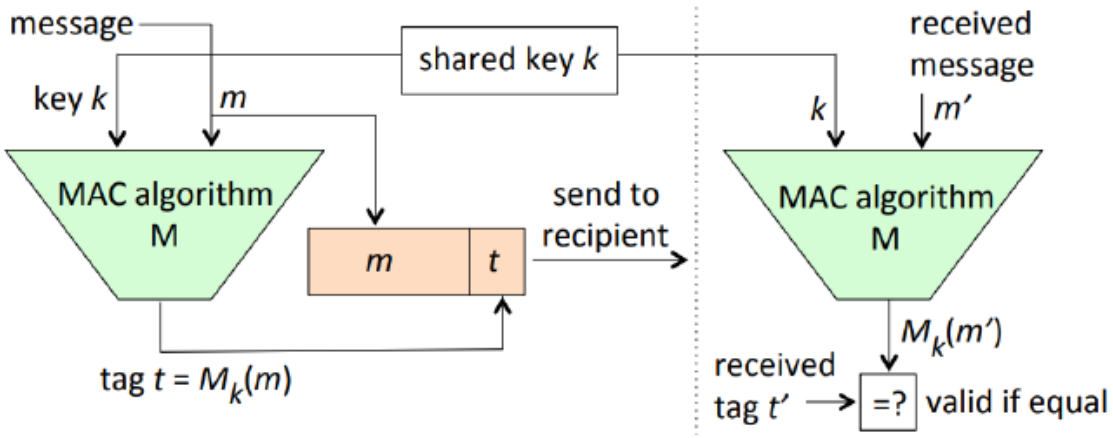
```
def find_near_collision(n):
    prefix_length = n
    bucket_size = 2 ** n  # Number of buckets based on prefix le

    buckets = {}  # Dictionary to store the buckets
    message = get_random_message()  # Generate a random message

    while True:
        hash_value = hashlib.sha256(message.encode()).hexdigest(

        prefix = hash_value[:prefix_length]  # Extract the prefi

        if prefix in buckets:
            # Near collision found!
            full_hash_1 = buckets[prefix]
            full_hash_2 = hash_value
            return full_hash_1, full_hash_2

        buckets[prefix] = hash_value  # Store the full hash in t
```

Message Authentication Codes(MAC)

- encryption cannot guarantee integrity

Assuring **integrity of data**, **identity of the party** that originated the data

Process:



1. hash(input msg + shared key) = tag
2. send (msg + tag) to recipient
3. hash(msg′ + shared key) = tag′, check valid if equal

Formula

HMAC = $h((k' \oplus \text{outer-pad}) \,||\, h((k' \oplus \text{inner-pad}) || m)))$

k' = h(k)

Properties

- adversary don't know key:
  - cannot get correct tag for msg
  - cannot generate new pair of matching msg & tag
- lack of non-repudiation
  - data origin authentication

Usage

- authenticated encryption
- key derivation from password
- SHA3 not vulnerable to hash extension

Authenticated encryption(AE)

MAC-then-Encrypt(MtE)

- HMAC plaintext, encrypt (plaintext, HMAC), send Enc(plaintext, HMAC)

Encrypt-then-MAC (EtM)

- encrypt plaintext, HMAC ciphertext, send (ciphertext, HMAC)

encryption ensure **confidentiality**

- CBC mode based on adjacent bit

Property:

- produce ciphertext + authentication tag
- encryption + MAC, all-in-one
- AE provide **data origin** authentication
- detect unauthenticated ciphertext manipulation

Formula

- (cipher, tag) = AE(k, IV, **m**)
- m = AD(k, IV, **c, t'**) if t = t'

AEAD - with associated data

(c, a, t) = AEAD(k, IV, **m, a**)

m = ADAD(k, IV, c, a, t') if t = t' and a and c are authenticated

CMM mode: counter CBC-MAC(auth) + CTR mode (stream cipher)