ECE568 笔记汇总

- Cryptography - Block Ciphers
- Cryptography - Ciphers
- Cryptography - Hashes, MACs, and Digital-Signitures
- Cryptography - Public-Key Cryptography
- Cryptography - Stream Ciphers

Table of Contents

Integrity & Authentication

Encryption

- Solves confidentiality
  - Concerned about who can read the message
- Does NOT guarantee...
  - **Integrity** - message has not been tampered with
  - **Authentication** - message arrived from the intended source

Even without knowing encryption key, attacker can

- insert random data (integrity)
- replace entire message with random data (integrity)
- reorder blocks if using ECB, replay attack (sending back transaction request multiple times, solved by ID and timestamp), flip bits

Cryptographic Hashes

- Solves integrity & authentication - detect if the message is changed in transit
- Used as part of
  - MDC (Modification Detection Codes) to provide integrity
  - MAC (Message Authentication Codes) integrity and authentication
  - Digital signatures to provide integrity, authentication and non-repudiation
- A **hash function** converts a large input into a smaller (typically fixed size) output, **H(m) = h**
  - **m** is the **data pre-image**
  - **h** is the **hash value/message digest**
  - **H()** is a lossy compression function
- **Cryptography hash function** needs
  - **Pre-image Resistance** given a hash value, hard to find a preimage that will yield the hash value (hard to reverse hash function)
  - **2nd Preimage Resistance** given preimage, hard to find another preimage that hashes to the same hash value

- **Collision Resistance** hard to find collisions (2 preimage values coincidentally hash to the same hash value)

Example SHA

- **shasum** takes any input and produces fixed-length hash value
  - very small changes in the preimage produces a very different hash value

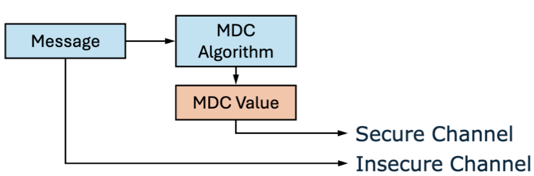
```shell
$ echo "Cryptographic hash values are like fingerprints" | shasum
d05b4ffc0677f1c5811dae6d7b914c2b60578d48

$ echo "cryptographic hash values are like fingerprints" | shasum
62e18bbb87c8e894dc3c73cc62ae6006d73bbe06
```

Hash Length

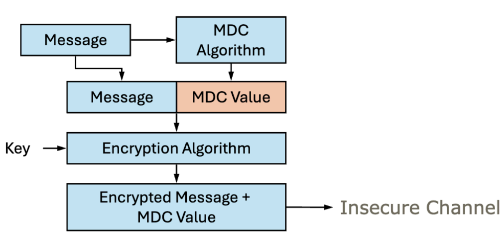
- When a hash function has 3 properties, it is ideal hash
- Security depends entirely on the length of the hash output
- If length of hash output has **n** bits, then
  - **2nd preimage resistance** expected number of guesses to find another preimage that hashes to a given hash value is 2^{n-1}
  - **Collision resistance** expected number of tries to find any 2 preimage hashing to same value is $2^{n/2}$ (birthday attack #question/ece568)

MDC (Modification Detection Codes)

- use hashes to provide **integrity**
  - alongside file downloads
  - AKA Message Integrity Code (MIC)
- taking a has of a message and sending the hash and the message separately allows the receiver to detect if the message has been modified in transit
- MDC with a secure channel provides integrity
  - allows receiver to verify the integrity of the message, does not protect message confidentiality
  - if confidentiality required, message should be encrypted separately



- MDC with encryption provides confidentiality, integrity and authentication
  - doesn't require secure channel for distribution
  - after decryption, receiver can verify both the source and integrity of message by checking MDC value matches message

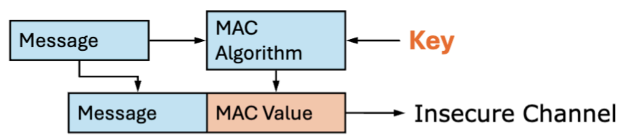


Commonly Used MDC

- MD5 - Ron Rivest at RSA, 128 bit hash value from arbitrary large input; broken
- SHA1 - NIST with help from NSA, 160bit hash, weakness
- SHA256 - produces 256bit hash, strong

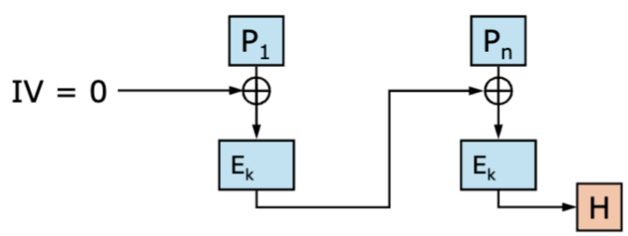
Message Authentication Code (MAC)

- uses hash to provide **integrity** and **authentication** (no **confidentiality**, msg not encrypted)
  - MAC is constructed as $h = H(k, M)$ where **k** is the secret key and **M** is the message
  - Receiver knows that whoever generated the MAC must also know the key, thus authenticating the message source #question/ece568
    - Sender & Receiver must both have secret key in the first place, receiver can verify the source
- Purpose: Detect unauthorized alteration of message & digest
  - Only whoever has SK can create acceptable digest
  - **integrity** & **authentication** (only other party has secret key)



MAC using Symmetric Ciphers

- MAC is often constructed from symmetric ciphers; CBC-MAC
  - similar to CBC, but single hash value is produced at the end
  - hash output size identical to block size of block cipher
  - if using same cipher for encrypting and MAC, MAC key must be different from encryption key
    - #question/ece568 Is each of the E_k blocks one of the construction above?

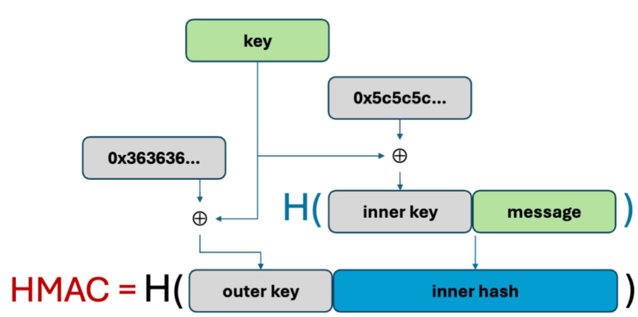


HMAC (Hashed MAC)

- simply concatenating key + message H(K + M) is *not secure*
  - MDCs are iterated functions, a single non-nested has may allow an adversary to add arbitrary info at the end of the message and compute new forged MAC
- HMAC applies the has 2x for security

$$HMAC = H((K \oplus opad) + H((K \oplus ipad) + M))$$

- A MAC can also be constructed by concatenating the Secrete Key with Musing a hash, creating HMAC
  - In HMAC, opad (outer padding) and ipad (inner padding) are those constant values shown in the diagram



HMAC = H[(K ⊕ opad) + H((K ⊕ ipad) + M)]

- "+" denotes string concatenation, "⊕" denotes logical XOR
- **M** is the arbitrary-length message
- Assume hash block size = **n** bits (*e.g.*, 512 bits for SHA1)
- **K** is the key, padded with 0's on right side to **n** bits
- **opad** = 0x3636... (or `00110110`) repeated to **n** bits
- **ipad** = 0x5c5c... (or `01011100`) repeated to **n** bits

Effectively:

$$HMAC = H(key_1 + H(key_2 + message))$$

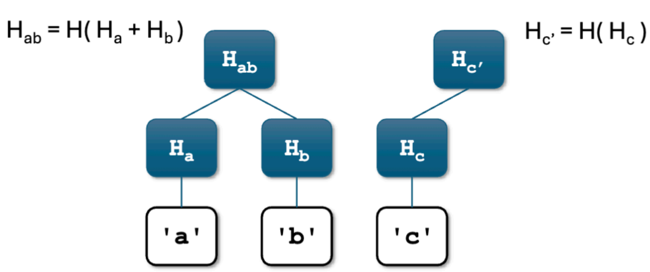
- The inner and outer padding are chosen to minimize number of common bits in key_1 and key_2

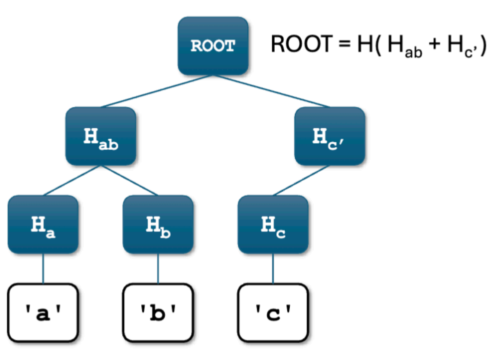
Hash-based Data Structures

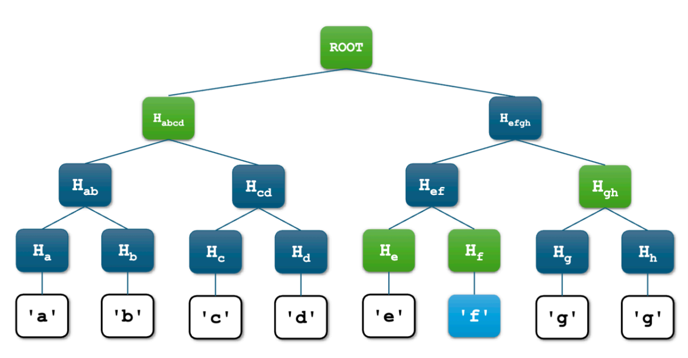
- useful to verify integrity of a **set** of things instead of single object
  - concatenating obj into long string and computing hash over string *DOES NOT WORK*
  - For example, if we're hashing directory paths
    - "/home" + "/etc"
    - "/hom" + "e/etc"
    - These would produce the same hash despite representing different directory structures. This undermines the integrity verification purpose of the hash
    - Example
- DS exist can hold data + ensure data hasn't changed
  - **hash tree** - data integrity, allowing for easy updates
  - **block chain** - allows journal of events to be created, integrity + authentication

Merkle Tree

- A **Merkle Tree Proof** is the set of hashes that allows you to prove to someone else that you both have the same tree, containing the specified element
  - **proof('a') = [H_a, H_b, H_c, ROOT]**







The purpose of a Merkle proof is to prove that a specific piece of data exists at a specific position in your tree. That's why you need to provide the sibling hashes - they allow the other person to:

1. Take the element you're proving ('a')
2. Follow the exact path through the tree using the sibling hashes
3. Arrive at the ROOT hash through the correct sequence of operations

Blockchain