IERG 4130 – 2026 Fall

香港中文大學
CUHK

Cryptographic Hashes

Yajin ZHOU

https://yajin.org

Credits: the slides are from CS161 of UC Berkeley,

Summary: One-Time Pads



- **One-time pads**
  - Symmetric encryption scheme: Alice and Bob share a secret key.
  - Encryption and decryption: Bitwise XOR with the key.
  - No information leakage if the key is never reused.
  - Information leaks if the key is reused.
  - Impractical for real-world usage

Summary: Block Ciphers



- **Encryption: input a *k*-bit key and *n*-bit plaintext, receive *n*-bit ciphertext**
- **Decryption: input a *k*-bit key and *n*-bit ciphertext, receive *n*-bit plaintext**
- **Correctness: when the key is fixed, $E_K(M)$ should be bijective**
- **Security**
  - Without the key, E_K(m) is computationally indistinguishable from a random permutation
  - Brute-force attacks take astronomically long and are not possible
- **Efficiency: algorithms use XORs and bit-shifting (very fast)**
- **Implementation: AES is the modern standard**
- **Issues**
  - Not IND-CPA secure because they're deterministic
  - Can only encrypt *n*-bit messages

Summary

- **ECB mode: Deterministic, so not IND-CPA secure**
- **CBC mode**
  - IND-CPA secure, assuming no IV reuse
  - Encryption is not parallelizable
  - Decryption is parallelizable
  - Must pad plaintext to a multiple of the block size
  - IV reuse leads to leaking the existence of identical blocks at the start of the message

Summary

- **CTR mode**
  - IND-CPA secure, assuming no IV reuse
  - Encryption and decryption are parallelizable
  - Plaintext does not need to be padded
  - Nonce reuse leads to losing all security
- **Lack of integrity and authenticity**

Cryptographic Hash Function: Definition



- **Hash function: *H*(*M*)**
  - Input: *Arbitrary* length message *M*
  - Output: *Fixed* length, *n*-bit hash
  - Sometimes written as $\{0, 1\}^* \rightarrow \{0, 1\}^n$
- **Properties**
  - **Correctness**: Deterministic
    - Hashing the same input always produces the same output
  - **Efficiency**: Efficient to compute

Cryptographic Hash Function: Definition

- **Properties**
  - **Security**: One-way-ness (“preimage resistance”)
  - **Security**: Collision-resistance
  - **Security:** Random/unpredictability, no predictable patterns for how changing the input affects the output
    - Changing one bit in the input causes the output to be completely different
    - Also called “random oracle” assumption

Hash Function: Intuition

- **A hash function provides a fixed-length "fingerprint" for a sequence of bits**
- **Example: Document comparison**
  - If Alice and Bob both have a 1 GB document, they can both compute a hash over the document and (securely) communicate the hashes to each other
  - If the hashes are the same, the files must be the same, since they have the same "fingerprint"
  - If the hashes are different, the files must be different

Hash Function: One-way-ness or Preimage Resistance



- **Informal:** Given an output *y*, it is infeasible to find *any* input *x* such that *H*(*x*) = *y*
- **More formally: For every polynomial time adversary,**

  Pr[*x* chosen randomly from plaintext space; *y* = *H*(*x*):

  Adversary(*y*) outputs *x′* s.t. *H*(*x′*) = *y*] is negligible
- **Intuition: Here's an output. Can you find an input that hashes to this output?**
  - Note: The adversary just needs to find *any* input, not necessarily the input that was actually used to generate the hash
- **Example: Is H(x) = 42 one-way?**

Hash Function: Collision Resistance

- **Collision: Two different inputs with the same output**
  - $x \neq x'$ and $H(x) = H(x')$
  - Can we design a hash function with no collisions?
    - No, because there are more inputs than outputs (pigeonhole principle)
  - However, we want to make finding collisions *infeasible* for an attacker
- **Collision resistance**: It is infeasible to (i.e. no polynomial time attacker can) find any pair of inputs $x' \neq x$ such that $H(x) = H(x')$
- Intuition: Can you find *any* two inputs that collide with the same hash output for *any* output?

Hash Function: Examples



- **MD5**
  - Output: 128 bits
  - Security: Completely broken
- **SHA-1**
  - Output: 160 bits
  - Security: Completely broken in 2017
  - Was known to be weak before 2017, but still used sometimes

Hash Function: Examples



- **SHA-2**
  - Output: 256, 384, or 512 bits (sometimes labeled SHA-256, SHA-384, SHA-512)
  - Not currently broken (but some variants are vulnerable to a length extension attack)
  - Current standard
- **SHA-3 (Keccak)**
  - Output: 256, 384, or 512 bits
  - Current standard (not meant to replace SHA-2, just a different construction)

Length Extension Attacks



- **Length extension attack**: Given $H(x)$ and the length of x, but not x, an attacker can create $H(x \,||\, m)$ for any m of the attacker's choosing
  - Note: This doesn't violate any property of hash functions but is undesirable in some circumstances
- **SHA-256 (256-bit version of SHA-2) is vulnerable**
- **SHA-3 is not vulnerable**

Length Extension Attacks

- **One simple example**

```
state = 0
For each byte b :
    state = (state + b) mod 256
H(x) = state
```

- **H(“ABCD”) = (**0x41 + 0x42 + 0x43 + 0x44**) mode 0x100 = 0xa**

Length Extension Attacks



- Suppose the attacker knows that H(x) = 0x and length = 4
- Can we calculate the value of H (x || “!”) without knowing the value of x?

Length Extension Attacks

- **Suppose the attacker knows that H(x) = 0x and length = 4**
- **Can we calculate the value of H (x || “!”) without knowing the value of x?**
  - **Yes. Hash value leaks the state**
    - old state = 0x0a
    - new byte = 0x21
    - new state = 0x0a + 0x21 = 0x2b
    - H(x || "!") = 0x2b

Do hashes provide integrity?



- **It depends how you use them**
- **Scenario**
  - Mozilla publishes a new version of Firefox on some download servers
  - Alice downloads the program binary
  - How can she be sure that nobody tampered with the program?

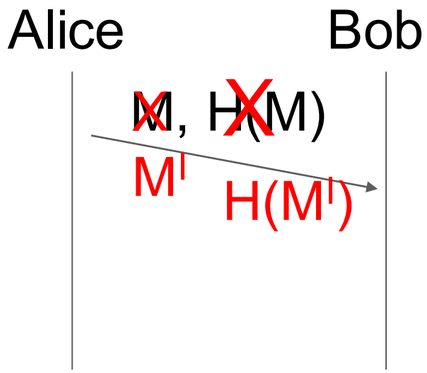
Do hashes provide integrity?



- **Idea: use cryptographic hashes**
  - Mozilla hashes the program binary and publishes the hash on its website
  - Alice hashes the binary she downloaded and checks that it matches the hash on the website
  - If Alice downloaded a malicious program, the hash would not match (tampering detected!)
  - An attacker can't create a malicious program with the same hash (one-way property)
- **Threat model: We assume the attacker cannot modify the hash on the website**
  - We have integrity, as long as we can communicate the hash securely

Do hashes provide integrity?

- **It depends how you use them**
- **Scenario**
  - Alice and Bob want to communicate over an insecure channel
  - Mallory might tamper with messages
- **Idea: Use cryptographic hashes**
  - Alice sends her message with a cryptographic hash over the channel
  - Bob receives the message and computes a hash on the message
  - Bob checks that the hash he computed matches the hash sent by Alice
- **Threat model: Mallory can modify the message *and the hash***
  - No integrity!



Do hashes provide integrity?



- **It depends how you use them**
- **If the attacker can modify the hash, hashes don't provide integrity**
- **Main issue: Hashes are *unkeyed* functions**
  - There is no secret key being used as input, so any attacker can compute a hash on any value

First Try



- Alice and Bob share a secret K
- Alices computes tag = H(K || M)
- Alices sends the (tag, M) to Bob
- Does this construction provide cryptographic integrity?

First Try

- Alice and Bob share a secret K
- Alices computes tag = H(K || M)
- Alices sends the (tag, M) to Bob
- Does this construction provide cryptographic integrity?
  - **Not necessarily. For some hash functions, this construction is vulnerable to length-extension attacks**
  - What if we use a hash function that is not vulnerable to length-extension attacks?

First Try

- Alice and Bob share a secret K
- Alices computes tag = H(K || M)
- Alices sends the (tag, M) to Bob
- Does this construction provide cryptographic integrity?
  - **Not necessarily. For some hash functions, this construction is vulnerable to length-extension attacks**
  - What if we use a hash function that is not vulnerable to length-extension attacks?
    - Then this particular attack may not apply. However, this alone does not prove that H(K || M) is a secure MAC construction. In practice, use a standard and analyzed MAC construction, such as HMAC -> discuss in next lecture.

Summary



- **Map arbitrary-length input to fixed-length output**
- **Output is deterministic and unpredictable**
- **Security properties**
  - One way: Given an output y, it is infeasible to find any input x such that $H(x) = y$.
  - Second preimage resistant: Given an input x, it is infeasible to find another input $x' \neq x$ such that $H(x) = H(x')$.
  - Collision resistant: It is infeasible to find another any pair of inputs $x' \neq x$ such that $H(x) = H(x')$.
- **Some hashes are vulnerable to length extension attacks (H(M) → H(M||M'))**
- **Hashes don't provide integrity (unless you can publish the hash securely)**