Intro to IT Security

CS306C—Fall 2022

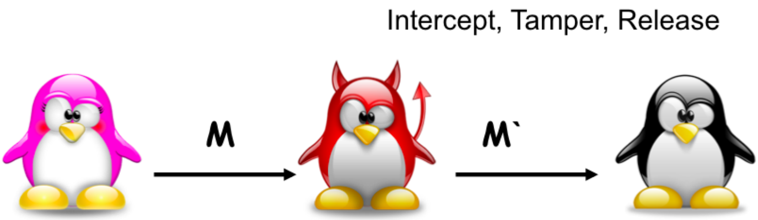
Prof. Antonio R. Nicolosi

Antonio.Nicolosi@stevens.edu

Data Integrity in the Symmetric Setting

Data Integrity

Integrity: Preventing unauthorized changes



- The receiver should be able to check whether the msg was modified during transmission
  - No one should be able to tamper with the msg, without the recipient noticing the alteration

Data Integrity: Example

- A wants to email an executable file F to B
- A wants to ensure that the executable file is received by B without modifications
  - A sends out the file to B
  - A gives the *hash* of the file to B (out of band, *e.g.*, on a piece of paper)
- Goal: Integrity not Confidentiality
- Idea: Given F and $hash(F)$, very hard to find $badF$ such that $hash(F) = hash(badF)$

Integrity *vs.* Confidentiality

- Encryption does not guarantee integrity
  - Attacker may be able to modify the encrypted msg without learning the msg itself
- Example:
  - Use OTP to encrypt m using key k: $c = m \oplus k$
  - Take a different message m'. Compute
$$c' = c \oplus m' = (m \oplus m') \oplus k$$
  - c' is a valid encryption of message $\bar{m} = m \oplus m'$

Message Authentication Codes (MACs)

- In the symmetric setting, the correct tool to get msg integrity is a *MAC*
- Functionality
  - $Mac_k(m) = t$; t is called tag
  - $Vrf_k(m,t) = \begin{cases} 1 & accept \\ 0 & reject \end{cases}$
- Sender and Receiver share MAC key k
- Sender sends $(m, Mac_k(m))$
  - m could be $Enc_{k'}(m')$
- Note: Careful with reply attacks!

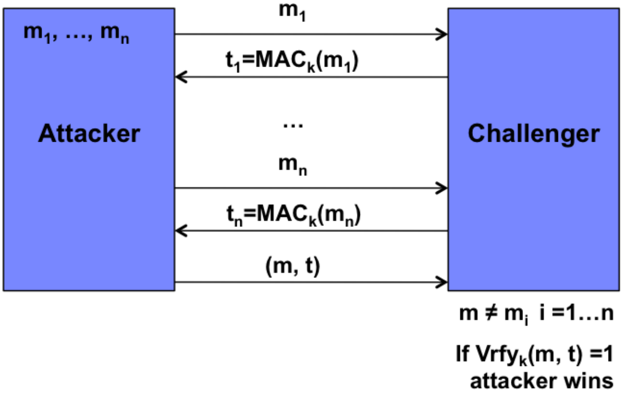
Message Authentication Codes (MACs) (con'd)

A message authentication code (MAC) is a tuple of probabilistic polynomial-time algorithms (Gen, Mac, Vrf) such that:

- $Gen(1^n)$: A key-generation algorithm that takes as input the security parameter 1^n and outputs a key k with $|k| \geq n$
- $Mac_k(m)$: The tag-generation algorithm that takes as input a key k and a message $m \in \{0,1\}^*$, and outputs a tag t. Since this algorithm may be randomized, we write this as $t \leftarrow Mac_k(m)$
- $Vrf_k(m,t)$: The verification algorithm that takes as input a key k, a message m, and a tag t. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We assume without loss of generality that Vrf is deterministic, and so write this as $b := Vrf_k(m,t)$

MAC: Security

- Attack Game



- A MAC is secure if, for all attacker running for some time T ($T = 100$ years), the probability that the attacker creates a “forgery” is at most $\epsilon = 2^{-80}$

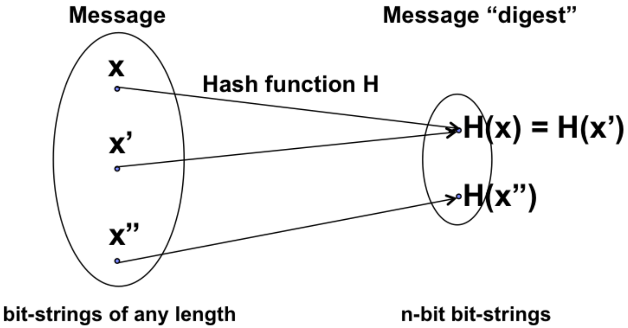
Digression: Cryptographic Hash Functions

Let $H : X \to Y$ be a function. H is a *hash function* if it satisfies the following properties:

- It is efficiently computable
- Many elements in the domain are mapped to the same elements in the codomain

Cryptographic Hash Functions: Definition

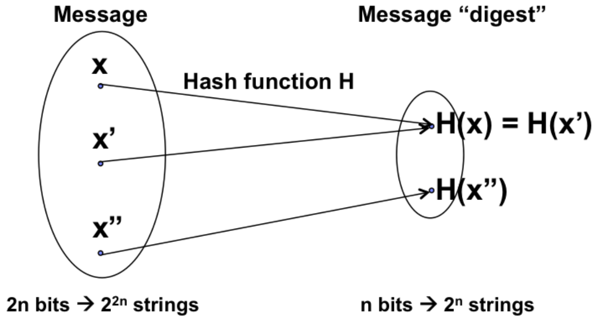
- $H : \{0,1\}^* \to \{0,1\}^n$



Cryptographic Hash Functions: Definition

- H is a lossy compression function
- H hashes arbitary-length input to fixed-size output
  - Typical output size: 160-512 bits
  - Cheap to compute on large input
- Collision: $H(x) = H(x')$, for distinct x, x'
- Result of hashing should look random
  - Even if $|x| \neq |x'|$ or x is a prefix of x' $(x' = x||x'')$

Do we always have collisions? Yes!



- Pigeon Hole Principle
  - On average $\frac{2^{2n}}{2^n} = 2^n$ collisions!

How Long Does it Take to Find a Collision?

- For a "good" hash function, roughly $\sqrt{2^n} = 2^{n/2}$ evaluations
- Brute-force attack:
  - Take random x_0; compute $H(x_0)$
  - Take random x_1; check if
  $$H(x_0) = H(x_1)$$
  - If not, take random x_2; check if
  $$H(x_2) = H(x_0) \quad \vee \quad H(x_2) = H(x_1)$$
  - If not, take random x_3; check if
  $$H(x_3) = H(x_0) \quad \vee \quad H(x_3) = H(x_1) \quad \vee \quad H(x_3) = H(x_2)$$
  - If not, ...
  - ...take random x_i; check if
  $$H(x_i) = H(x_0) \quad \vee \quad H(x_i) = H(x_1) \quad \vee \quad \ldots \quad \vee \quad H(x_i) = H(x_{i-1})$$

How Long Does it Take to Find a Collision?

- For a “good” hash function, roughly $\sqrt{2^n} = 2^{n/2}$ evaluations
- Brute-force attack:
  - Take random x_0; compute $H(x_0)$
  - Take random x_1; check if
  $$H(x_0) = H(x_1)$$
  - If not, take random x_2; check if
  $$H(x_2) = H(x_0) \quad \vee \quad H(x_2) = H(x_1)$$
  - If not, take random x_3; check if
  $$H(x_3) = H(x_0) \quad \vee \quad H(x_3) = H(x_1) \quad \vee \quad H(x_3) = H(x_2)$$
  - If not, ...
  - ...take random x_i; check if
  $$H(x_i) = H(x_0) \quad \vee \quad H(x_i) = H(x_1) \quad \vee \quad \ldots \quad \vee \quad H(x_i) = H(x_{i-1})$$

How Long Does it Take to Find a Collision?

- For a “good” hash function, roughly $\sqrt{2^n} = 2^{n/2}$ evaluations
- Brute-force attack:
  - Take random x_0; compute $H(x_0)$
  - Take random x_1; check if
  $$H(x_0) = H(x_1)$$
  - If not, take random x_2; check if
  $$H(x_2) = H(x_0) \quad \vee \quad H(x_2) = H(x_1)$$
  - If not, take random x_3; check if
  $$H(x_3) = H(x_0) \quad \vee \quad H(x_3) = H(x_1) \quad \vee \quad H(x_3) = H(x_2)$$
  - If not, ...
  - ... take random x_i; check if
  $$H(x_i) = H(x_0) \quad \vee \quad H(x_i) = H(x_1) \quad \vee \quad \ldots \quad \vee \quad H(x_i) = H(x_{i-1})$$

How Long Does it Take to Find a Collision?

- For a "good" hash function, roughly $\sqrt{2^n} = 2^{n/2}$ evaluations
- Brute-force attack:
  - Take random x_0; compute $H(x_0)$
  - Take random x_1; check if
  $$H(x_0) = H(x_1)$$
  - If not, take random x_2; check if
  $$H(x_2) = H(x_0) \quad \vee \quad H(x_2) = H(x_1)$$
  - If not, take random x_3; check if
  $$H(x_3) = H(x_0) \quad \vee \quad H(x_3) = H(x_1) \quad \vee \quad H(x_3) = H(x_2)$$
  - If not, ...
  - ... take random x_i; check if
  $$H(x_i) = H(x_0) \quad \vee \quad H(x_i) = H(x_1) \quad \vee \quad \ldots \quad \vee \quad H(x_i) = H(x_{i-1})$$

How Long Does it Take to Find a Collision?

- For a "good" hash function, roughly $\sqrt{2^n} = 2^{n/2}$ evaluations
- Brute-force attack:
  - Take random x_0; compute $H(x_0)$
  - Take random x_1; check if

$$H(x_0) = H(x_1)$$

  - If not, take random x_2; check if

$$H(x_2) = H(x_0) \quad \vee \quad H(x_2) = H(x_1)$$

  - If not, take random x_3; check if

$$H(x_3) = H(x_0) \quad \vee \quad H(x_3) = H(x_1) \quad \vee \quad H(x_3) = H(x_2)$$

  - If not, ...
  - ... take random x_i; check if

$$H(x_i) = H(x_0) \quad \vee \quad H(x_i) = H(x_1) \quad \vee \quad \ldots \quad \vee \quad H(x_i) = H(x_{i-1})$$

How Long Does it Take to Find a Collision?

- For a "good" hash function, roughly $\sqrt{2^n} = 2^{n/2}$ evaluations
- Brute-force attack:
  - Take random x_0; compute $H(x_0)$
  - Take random x_1; check if
$$H(x_0) = H(x_1)$$
  - If not, take random x_2; check if
$$H(x_2) = H(x_0) \quad \vee \quad H(x_2) = H(x_1)$$
  - If not, take random x_3; check if
$$H(x_3) = H(x_0) \quad \vee \quad H(x_3) = H(x_1) \quad \vee \quad H(x_3) = H(x_2)$$
  - If not, ...
  - ... take random x_i; check if
$$H(x_i) = H(x_0) \quad \vee \quad H(x_i) = H(x_1) \quad \vee \quad \ldots \quad \vee \quad H(x_i) = H(x_{i-1})$$

How Long Does it Take to Find a Collision? (cont'd)

- After k steps, we have checked roughly $k^2/2$ pairs:

$$\sum_{i=0}^{k-1} i = \frac{k(k-1)}{2} \simeq \frac{k^2}{2}$$

- For each pair, roughly $1/2^n$ chance to get collision
  - Think of one element as fixed, the other as random in $\{0,1\}^n$
- So after $2^{n/2}$ steps = $2^n/2$ pairs, roughly $1/2^n \cdot 2^n/2 = 1/2$ chance of (at least one) collision

Preimage Resistant Hash Functions

- Hard to win the following game b/w adversary A and challenger C
  - Hard = the best you can get is by following the brute-force attack
- $C \rightarrow A : k, y$
- $A \rightarrow C : x'$ s.t. $H_k(x') = y$

Second Preimage Resistant Hash Functions

- $C \rightarrow A : k, x$
- $A \rightarrow C : x'$ s.t. $H_k(x) = H_k(x')$

Collision Resistant Hash Functions

- $C \rightarrow A : k$
- $A \rightarrow C : x, x'$ s.t. $H_k(x) = H_k(x')$

Universal One-Way Hash Functions

- $A \rightarrow C : x$
- $C \rightarrow A : k$
- $A \rightarrow C : x'$ s.t. $H_k(x) = H_k(x')$

ϵ-Universal Hash Functions Family

- $A \rightarrow C : x, x'$
- $C \rightarrow A : k$
- Again, the goal of the adversary is to pick x, x' such that $H_k(x) = H_k(x')$

Common Hash Functions

- MD5: Message Digest Algorithm 5
  - 128-bit output
  - Designed by Ron Rivest (’91)
  - Collision resistance broken (’04, ’08)
  - Pre-image resistance broken (’09)
- SHA: Secure-Hash Algorithm
  - Designed by NSA (National Security Agency)
  - SHA-1: 160-bit output
  - Also, SHA-256, SHA-512
  - Collision resistance broken (’05)

Remarks

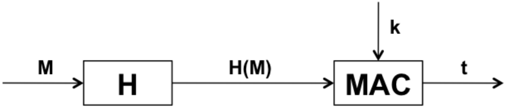
- The definition requires not just a family of hash functions, but a parameterized family of families (often called a *function ensemble*)
- Other flavors of hash function defined similarly
  - the power of the adversary varies by increasing or decreasing the information available to her at the time she must guess

A Consequence of the Definition

- At a minimum, all of our definitions require the hash functions to be one way (hard to find preimages of an element)
- Hence, we must have a large domain and generally, a large codomain as well in order to prevent an exhaustive search
- For example, SHA-1 maps $\{0, 1\}^* \to \{0, 1\}^{160}$

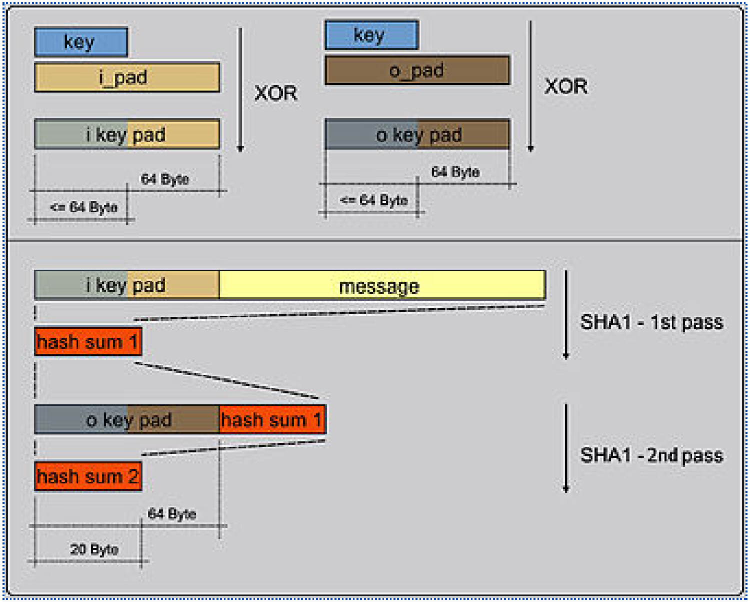
Hash and MAC

- Suppose we want to create a MAC for a long message
  - Hash message to create short "digest"
  - MAC the short digest



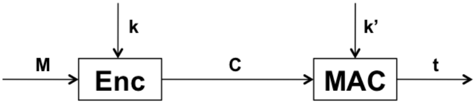
MACs in Practice

- $HMAC(k, m) = H(k \oplus opad, H(k \oplus ipad, m))$
  - H: cryptographic hash function
  - *ipad* is (0011 0110)=0x36 repeated to match the block-length of H
  - *opad* is (0101 1100)=0x5c repeated to match the block-length of H



Encryption and MAC

- To get confidentiality and integrity
  - Encrypt then MAC



Other Applications of Hash Functions: Fingerprinting

- Suppose two parties have files x, x' and would like to know "does $x = x'$?"
- Examples:
  - Verifying submitted work (uploads)
  - Verifying binaries and source code (downloads)
- Rather than communicating the entire file x or x', just send $H(x)$
- This resolves the question with very high probability and very low communication