# Lecture 1

Crypto Background

# This lecture

Crypto background

- hash functions
- random oracle model
- digital signatures
- … and applications

# Cryptographic Hash Functions

## Hash function

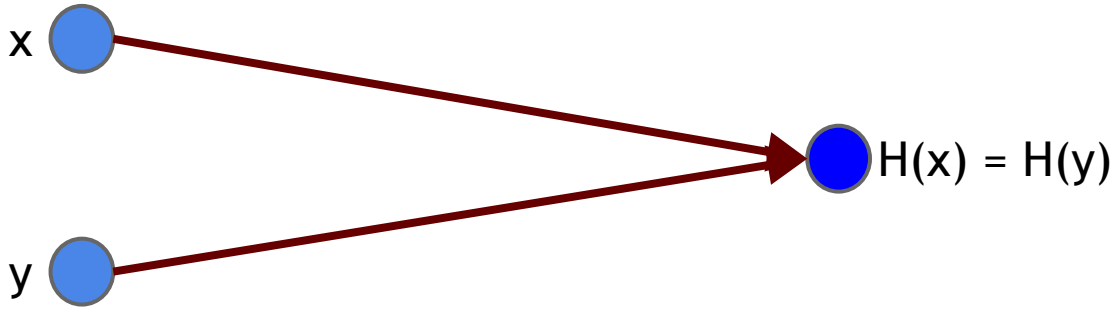
- takes a string of arbitrary length as input
- fixed-size output (i.e., hash function “compresses” the input)
- efficiently computable

Security properties:

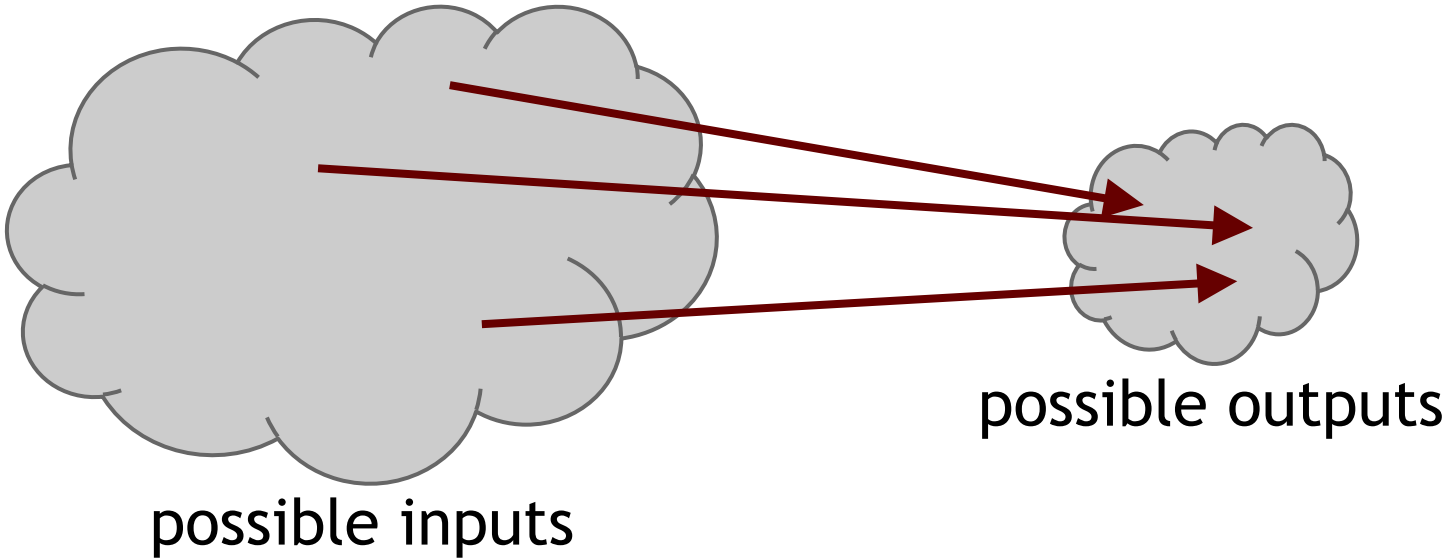
- Collision resistance
- Preimage resistance (one-way)

# Property 1: Collision resistance

No efficient adversary can find x and y such that x != y and H(x)=H(y)



# Collisions do exist ...




… but can a real-world adversary find them?

## How to find a collision (for 256 bit output)

- try $2^{130}$ randomly chosen inputs
- 99.8% chance that two of them will collide

## This works no matter what H is, but it takes too long to matter

- If a computer calculates 10,000 hashes/sec, it would take $10^{27}$ years to compute $2^{128}$ hashes

Is there a faster way to find collisions?

- For some possible H's, yes.
- For others (like SHA-256), we don't know of one.

Provably secure collision-resistant hash functions can be constructed based on hard number-theoretic problems.

# Defining Collision Resistance

- Modeling real-world adversaries
  - In practice, everyone has bounded resources
  - Therefore, reasonable to model adversary as such an entity
  - However, we do not make any assumptions about the adversarial strategy. He can use its (bounded) resources however

We will model adversary as a probabilistic polynomial-time (PPT) algorithm

# Defining Collision Resistance…

- Collision Resistance (informal): A hash function H is collision-resistant if for all uniform PPT adversaries A,

Pr[A outputs x,x’ s.t. x!=x and H(x)=H(x’)] = small

# Negligible functions

- Want to model success probability of adversary as a function of some input parameter k. Want that adversary's success probability quickly decreases as k increases (e.g., 1/2^k)
- Suppose adversary takes as input a parameter k, runs in time polynomial in k, and finds a collision with probability 1/k
- However, 1/k decays very slowly. To make 1/k very small, we would need to use a very large k. However, this would mean that all algorithms (even honest ones) run in very long time.
- Negligible function: Function that decays faster than any inverse poly

**Definition**: A function n(k) is negligible if for every polynomial p(k), there exists $k_0$ s.t. for all $k > k_0$, $n(k) < p(k)$

**Note**: Negligible probability cannot be "amplified" by polynomial number of trials

# Defining Collision Resistance

- **Collision Resistance**: A hash function H is collision-resistant if for all uniform PPT adversaries A, there exists a negligible function n(k), where k denotes the security parameter, s.t.

$$\Pr[A(1^k) \text{ outputs } x,x' \text{ s.t. } x!=x \text{ and } H(x)=H(x')] = n(k)$$

# Application: Hash as message digest

If we know H(x) = H(y),

it's safe to assume that x = y.

To recognize a file that we saw before,

just remember its hash.

Useful because the hash is small.

# Property 2: Pre-image Resistance

Intuition: Given H(x), no efficient adversary can find x (except with negligible probability)

Problem: What if input space of x is very small, or some inputs are much more likely than others?

H(“heads”)

H(“tails”)

easy to find x!

# Defining Preimage Resistance

- **Preimage Resistance**: A hash function H is preimage-resistant if for all PPT adversaries A, there exists a negligible function n(k), where k denotes the security parameter, s.t.

$$\Pr[x \leftarrow \{0,1\}^{poly(k)}, A(1^k, H(x)) \text{ outputs } x' \text{ s.t. } H(x')=H(x)] = n(k)$$

x is drawn from uniform distribution

# Preimage Resistance (contd.)

- If x is drawn from the uniform distribution, then inverting H(x) is hard
- But what if x is drawn from low-entropy distribution?
- Can append a random string r to x and then compute H(r | x) to prevent enumeration attacks

**Theorem**: Collision resistance implies preimage resistance (if the hash function is sufficiently compressing)

# Application: Commitment

Want to “seal a value in an envelope”, and
“open the envelope” later.

Commit to a value, reveal it later.

# Commitment Schemes

(*com*, *key*) := commit(*msg*)

*match* := verify(*com*, *key*, *msg*)

To seal *msg* in envelope:

(*com*, *key*) := commit(*msg*) -- then publish *com*

To open envelope:

publish *key*, *msg*

anyone can use verify() to check validity

# Commitment Schemes

(*com*, *key*) ← commit(*msg*)

*match* ← verify(*com*, *key*, *msg*)

Security properties:

- Weak Hiding: Given *com*, no PPT adversary can find* *msg*.
- Weak Binding: No PPT adversary can find* *msg != msg'* such that verify(commit(*msg*), *msg'*) == true

* Except with negligible probability

# Commitment Schemes

commit(*msg)* → ( H(*key* | *msg*), *key* )

where *key* is a random 256-bit value

verify(*com*, *key*, *msg*) → ( H(*key* | *msg)* == *com* )

Security properties:

- Hiding: If H is a ***random oracle***, given H(*key* | *msg)*, hard to find *msg*.
- Binding: Collision-reistance → Hard to find *msg != msg'* such that H(*key* | *msg)* == H(*key* | *msg')*

# Random Oracle (RO)

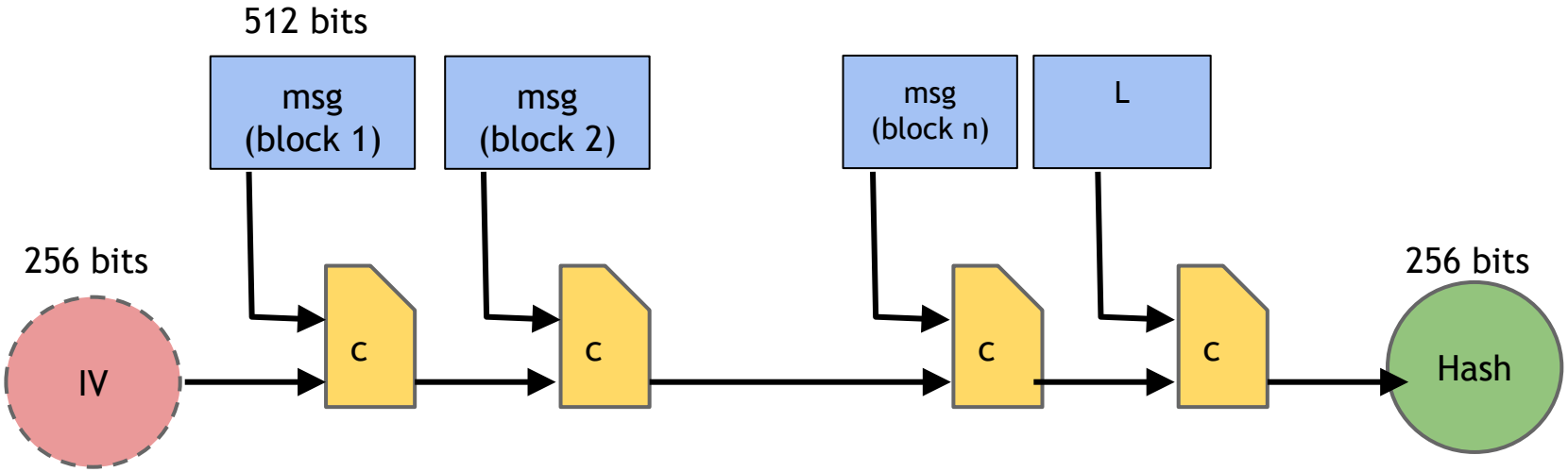
- Imagine an elf in a box with an infinite writing scroll
- Upon receiving an input x, the elf checks the scroll if there is an entry y corresponding to x. If yes, it returns y.
- Otherwise, elf chooses a random value y (from the output space) and returns it. It adds an entry (x,y) to the scroll.

# Random Oracle (RO)

- In practice-oriented provable security, hash functions are often modeled as a random oracle
- Each party (including adversary) is given black-box access to the random oracle. They can query the random oracle any polynomial number of times
- By definition, the answers of random oracle answers are unpredictable
- Random oracle captures many security properties such as one-wayness, collision-resistance .

# SHA-256 hash function

Suppose msg is of length L s.t. L is a multiple of 512 (pad with 0s otherwise)




**Theorem [Merkle-Damgard]**: If c is collision-resistant, then SHA-256 is collision-resistant.