Lecture 15 – Cryptographic Hash Functions

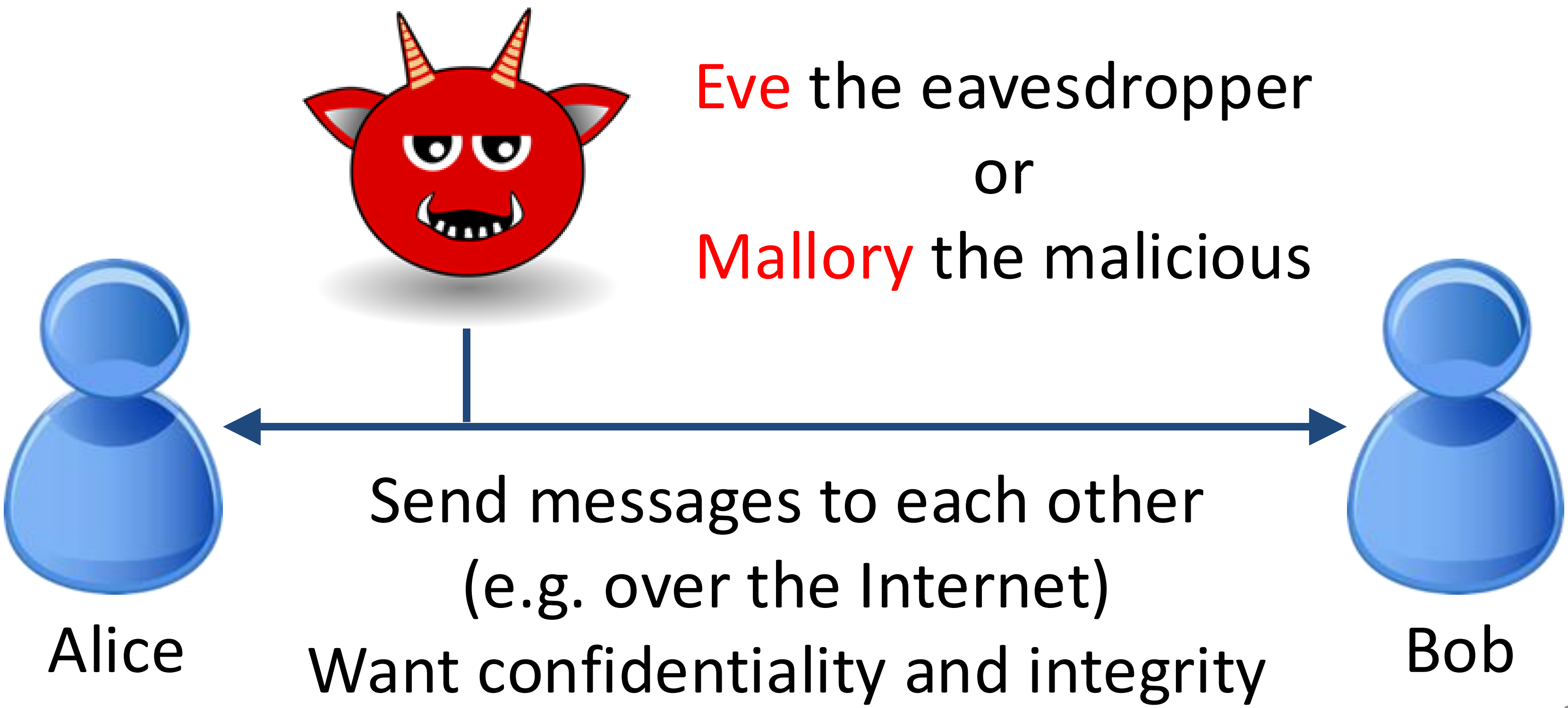
University of Illinois

ECE 422/CS 461

Next 4-5 lectures: Cryptography

Cryptography (or Cryptology)

- Studies techniques for secure communication in the presence an adversary who has control over the communication channel



Cryptography (or Cryptology)

- Studies techniques for secure communication in the presence an adversary who has control over the communication channel

- Also studies techniques for secure storage, secure collaborative computation, …

Goals of the Crypto Module

- Primitives we will cover: cryptographic hashing, symmetric & asymmetric encryption, message authentication codes & digital signatures

- Know the **interfaces** of basic crypto primitives
  - What are their inputs and outputs
  - What it means for them to be secure
  - What guarantees they provide and **not** provide
  - Where and how they are typically used
  - Which schemes to use when you need one

Both Rigorous & Empirical

- Modern cryptography is heavily based on *mathematics* but has to resort to *assumptions*
  - Rigorous vs. empirical

- Often *assume* some problem is hard to solve
  - Why do we believe that?
  - Because many experts have tried to solve them for decades or centuries, and could not

Today: Cryptographic Hash Functions

Goals of this Lecture

- By the end of this lecture you should know the following about crypto hash functions:
  - Interface
  - Desired properties
  - Lifecycle and currently recommended ones
  - Common design paradigms
  - Common applications

Cryptographic Hash Functions

- Input – data of an arbitrary length
- Output – fixed length, e.g., 256 bits
- Same input always produces the same output

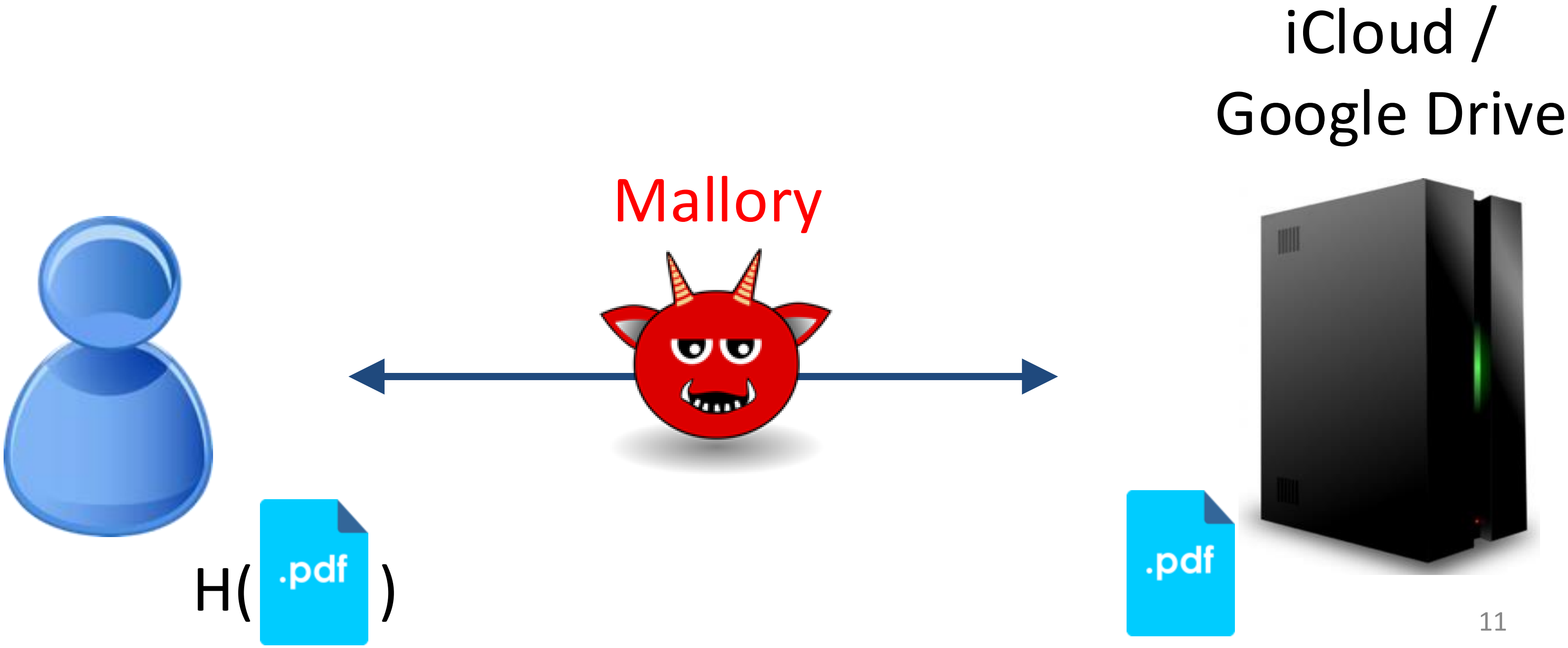
- Examples: MD5, SHA1, SHA2, SHA3
- SHA3-256("welcome") = 64db51f8f79ca7ec522a6b4a e5fc7e896daac5318b2e82730d7c7926b66d36eb
- SHA3-256(”Welcome") = 18ec669de973b4483db9b64 b2746ceda564cd2cdec2277169382944675a2ff9e

Applications

- Password hashing
  - System stores (username, salt, H(pw || salt))
  - User submits (username, pw)
  - System computes H(pw || salt) and compares

Applications

- Integrity of remote/external storage
  - User computes and stores H(file) locally
  - Compare hash upon download



Desired Properties

- Hard to invert (one-way, OW)
- Hard to find collisions (collision-resistant, CR)

- Exercise: which properties are used in the previous two applications and how?

(Slightly) More Formal Definition

- A cryptographic hash function H with n-bit output is a function:

$$y = H(x): \{0,1\}^* \rightarrow \{0,1\}^n$$

- One-way (OW, also called preimage resistance): for *almost* all y, **infeasible** to find x s.t. H(x) = y

- Collision-resistance (CR): **infeasible** to find x and x' s.t. x ≠ x' and H(x) = H(x')

What Does “Infeasible” Mean?

- Infeasible $\neq$ impossible
- In fact, both inversion and collision-finding are clearly possible by brute-force
  - Collisions must exist due to pigeon-hole principle
  - Brute-force collision in $O(2^{n/2})$ time and space due to “birthday paradox”
  - Brute-force inversion in $O(2^n)$ time
- Infeasible = no known attacks (yet) better than brute-force attacks

How to Choose n?

- Make $2^{n/2}$ a prohibitive cost
  - Since collision is the easier brute-force attack
- n = 128 used to be popular but is now too small
  - 2^{64} is no longer prohibitive
- Typical choice: n = 160, 192, 224 or 256
- n = 384 or 512 for the paranoid
  - $2^{256} \approx$ the number of particles in the universe

Desired Properties

- Do OW and CR imply each other?
  - No. We can construct hash functions that have one property but not the other. Fun challenge :)

- A stronger property is **pseudorandom:** for every input x, H(x) "looks random"
  - Implies OW and CR
  - But at the same needs to be deterministic …

(Slightly) More Formal Definition

- Ideal and unattainable hash: **random oracle**
  - Maintain a (infinite sized) table for every input-output pairs (x, y)
  - On new input x, generate a random n-bit value y, store (x, y) in table, and output y
  - On previous input x, output the stored y

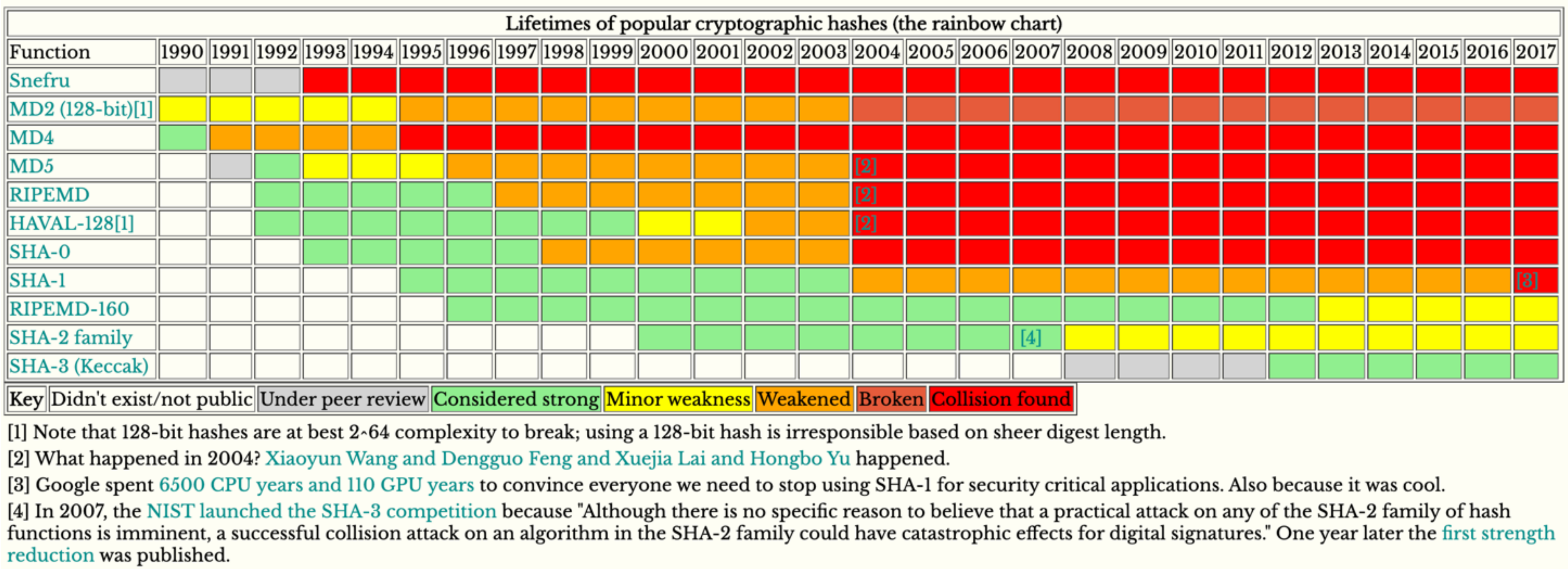
- H is pseudorandom if H "behaves like" a random oracle

Well-known Crypto Hash Functions

- ~~MD2, MD4, MD5, SHA1~~, SHA2, SHA3, …
  - ~~Strikethrough~~ = broken, never use again!
- How are they designed?
- How do we know they are OW and CR?
- Experts use their insights and experience to design and inspect/attack each other's design
  - MD = Message Digest, a series by Ron Rivest
  - SHA = Secure Hash Algorithm, NIST competition

Do NOT roll your own crypto!

Lifecycle of Crypto Hash Functions



- Eventually a function weakened
- Time to move to a new function and (hopefully) stay ahead of attackers (before a collision is found)

Common Design Paradigm

- How do we design a function that takes arbitrarily long input?
- Merkle-Damgård construction
  - Adopted by ~~MD5, SHA1~~, SHA2
  - First, design a compressing function that takes fixed-length inputs

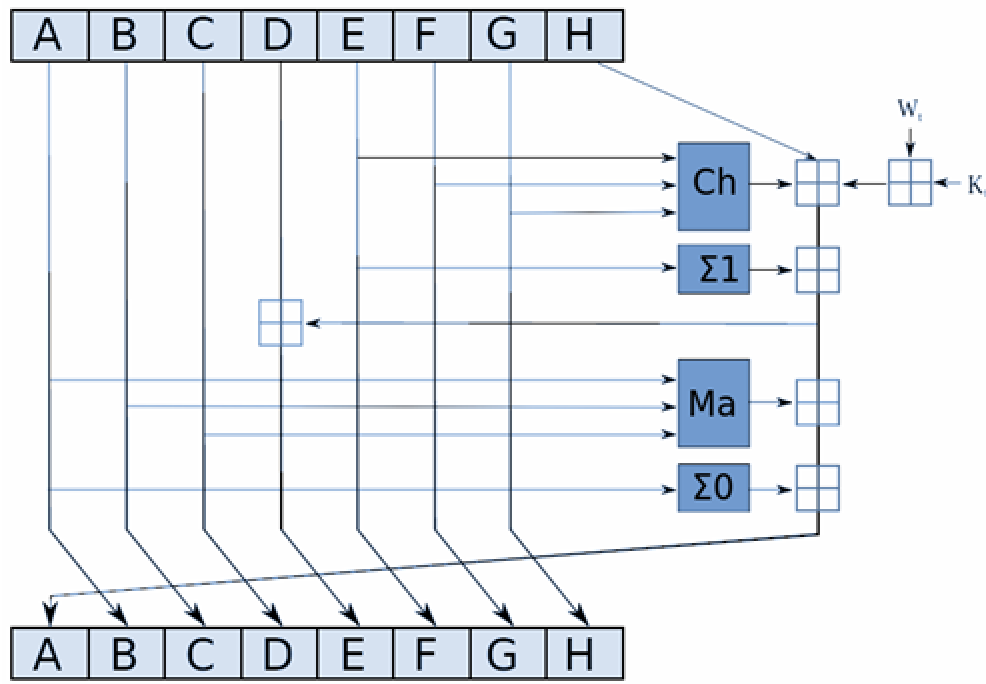
$$\boxed{f} : \{0,1\}^{2n} \rightarrow \{0,1\}^{n}$$

  - Then, “absorb” the input n-bit at a time

SHA2-256 Compressing Function

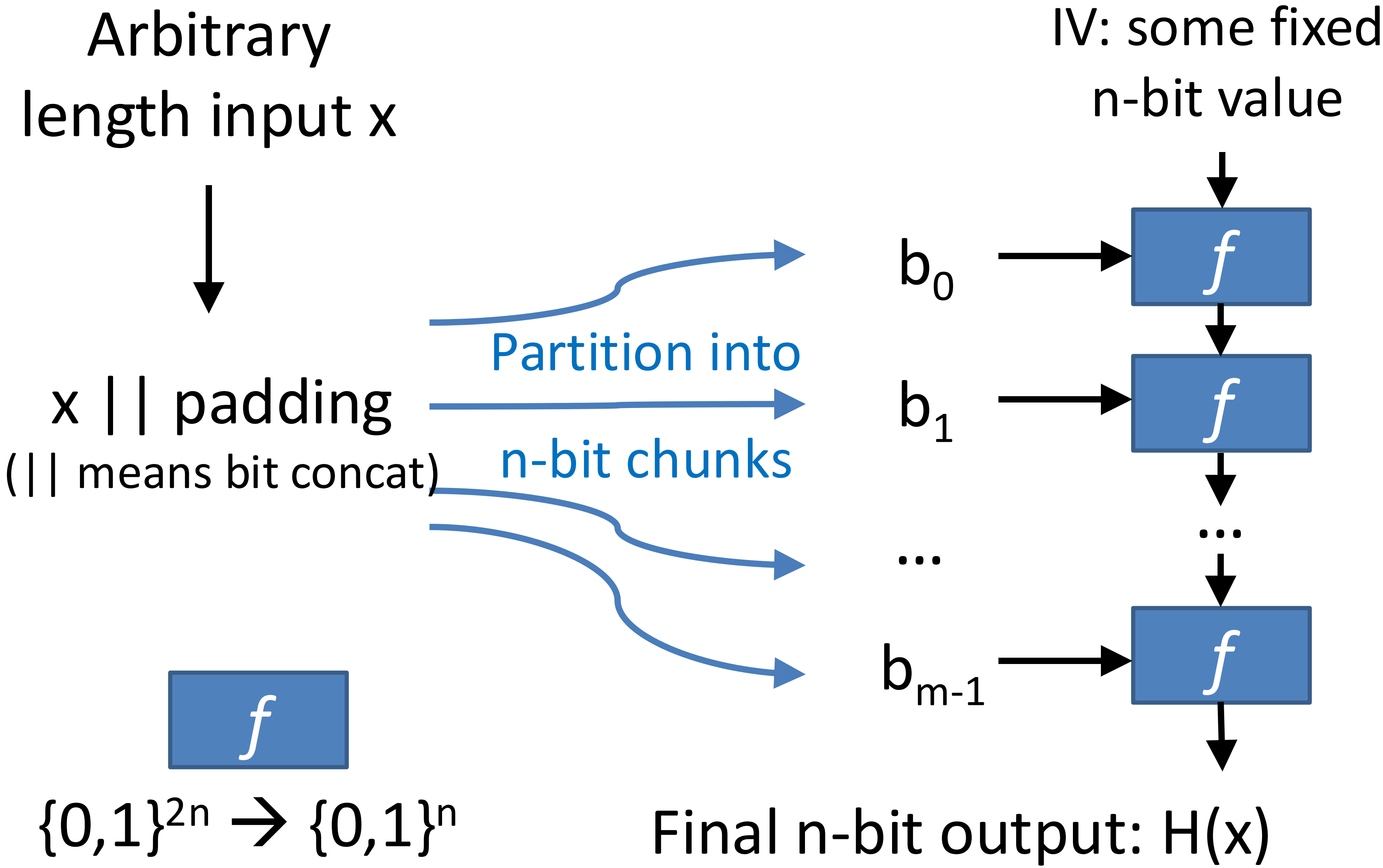
- Intentionally "hairy" and "messy"
- 64 rounds of this

$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$
$$\mathrm{Ma}(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$
$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$
$$\Sigma_1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$$



$\{0,1\}^{512} \rightarrow \{0,1\}^{256}$

Merkle-Damgård Construction



Merkle-Damgård Construction

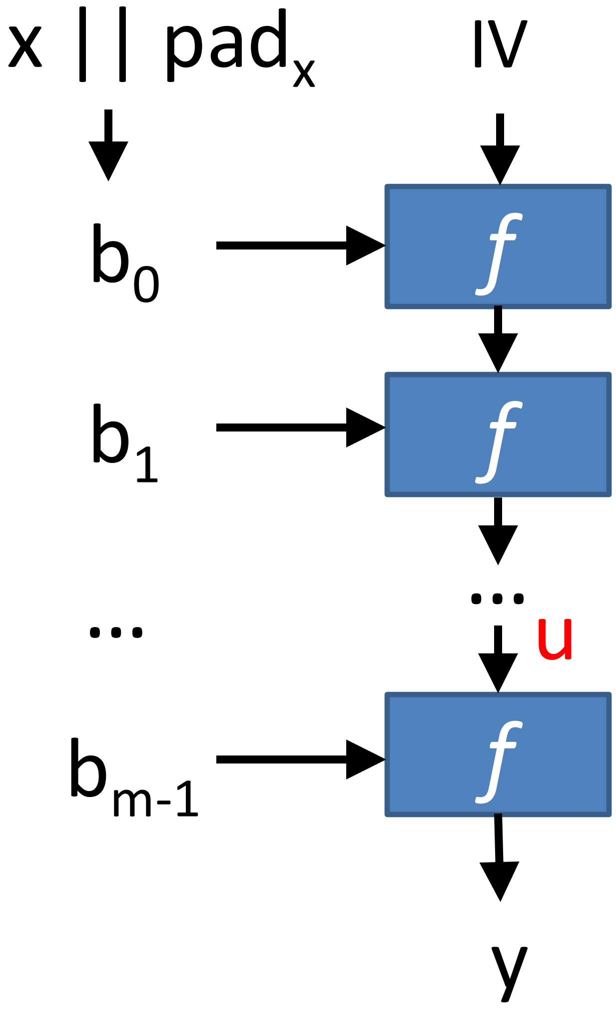
- Theorem: if the compressing function *f* is one-way (OW) and collision-resistant (CR), then a Merkle-Damgård hash is also OW and CR

- Proof idea: suppose some attacker breaks Merkle-Damgård, it also breaks *f*

- This is called security reduction. Allows us to focus on security of basic building blocks.

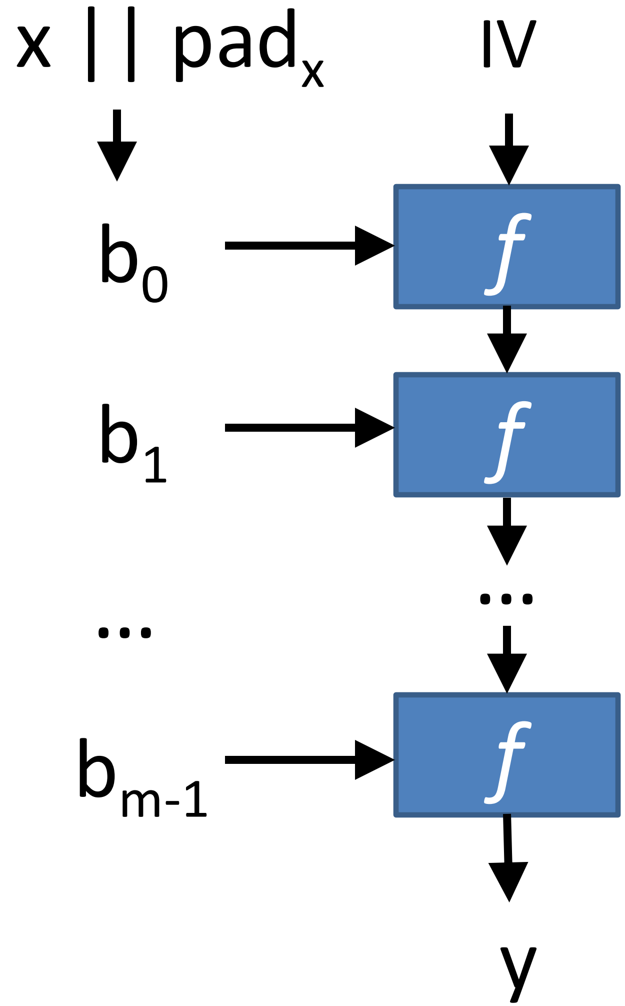
Reduction Proofs

- f OW → Merkle-Damgård OW
  - Suppose Merkle-Damgård is not OW, i.e., there is a feasible algo that finds preimage x s.t. H(x) = y
  - Easy to evaluate f "forward"
  - Then, f (u||b_{m-1}) = y. Preimage found for y under f. QED



Reduction Proofs

- f OW → Merkle-Damgård OW
- "f CR → Merkle-Damgård CR" is also true with "proper" padding



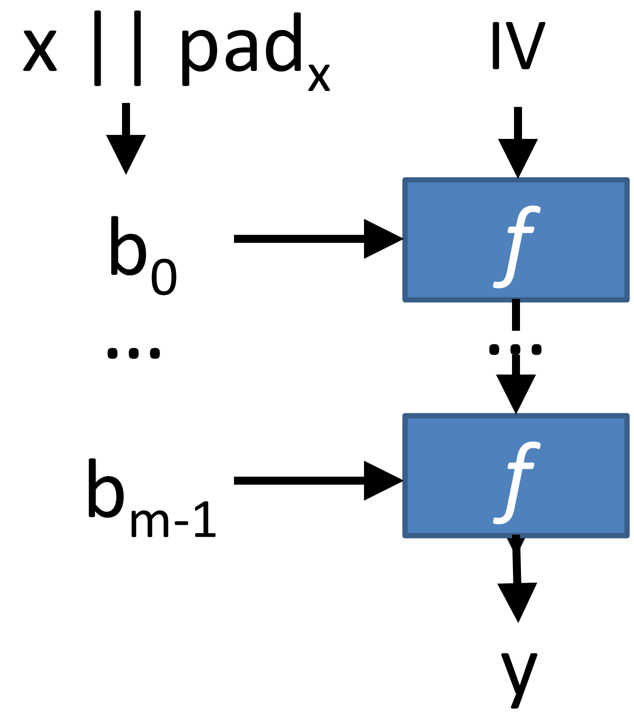
Subtleties in Padding

- Recall that we pad input to a multiple of n bits
- Pad with 0?
  - SomeLongBitString0000000
  - SomeLongBitString0000000

- Merkle-Damgård requires that inputs with different lengths, once padded, differ in their last blocks (typically, embed length in padding)
  - Can now prove CR reduction (exercise)

Merkle-Damgård Construction

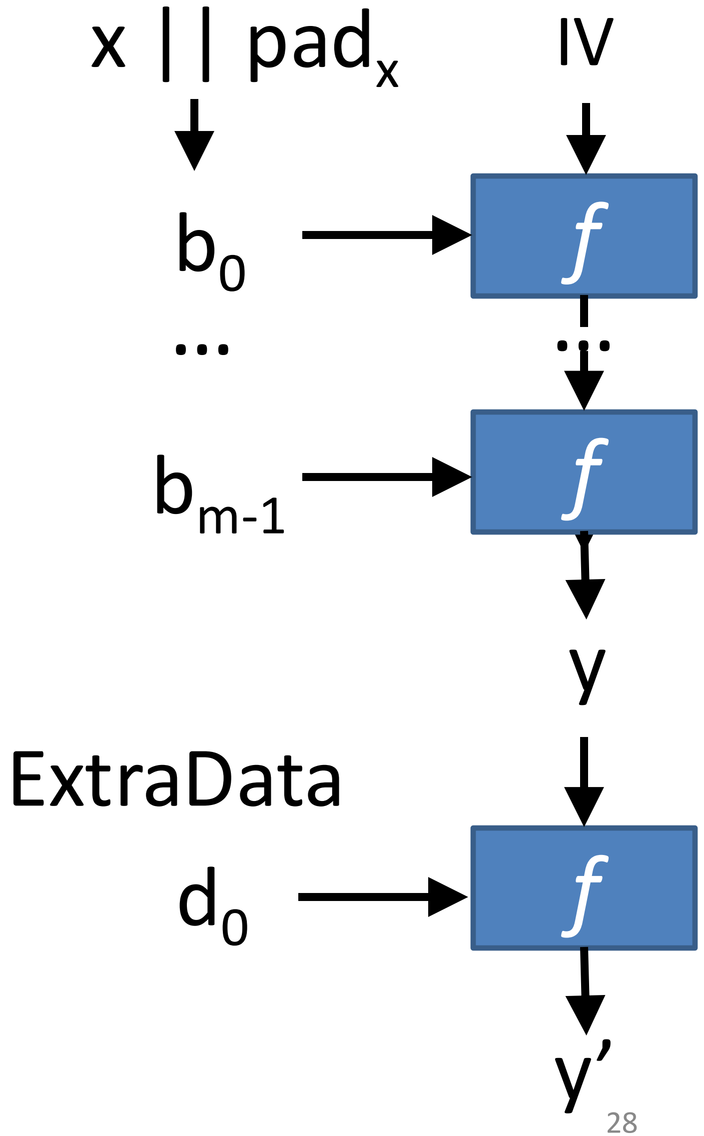
- f OW → Merkle-Damgård OW
- f CR → Merkle-Damgård CR (with proper padding)
- How about pseudorandomness?
  - Merkle-Damgård is NOT pseudorandom even if f is



Length Extension Attack

- Given H(x), one can compute H(x||pad$_x$||ExtraData) with more rounds of *f*

- A random oracle would not exhibit this behavior
  - Given { H(x_i)=y_i }, for a new x’, H(x’) would be random

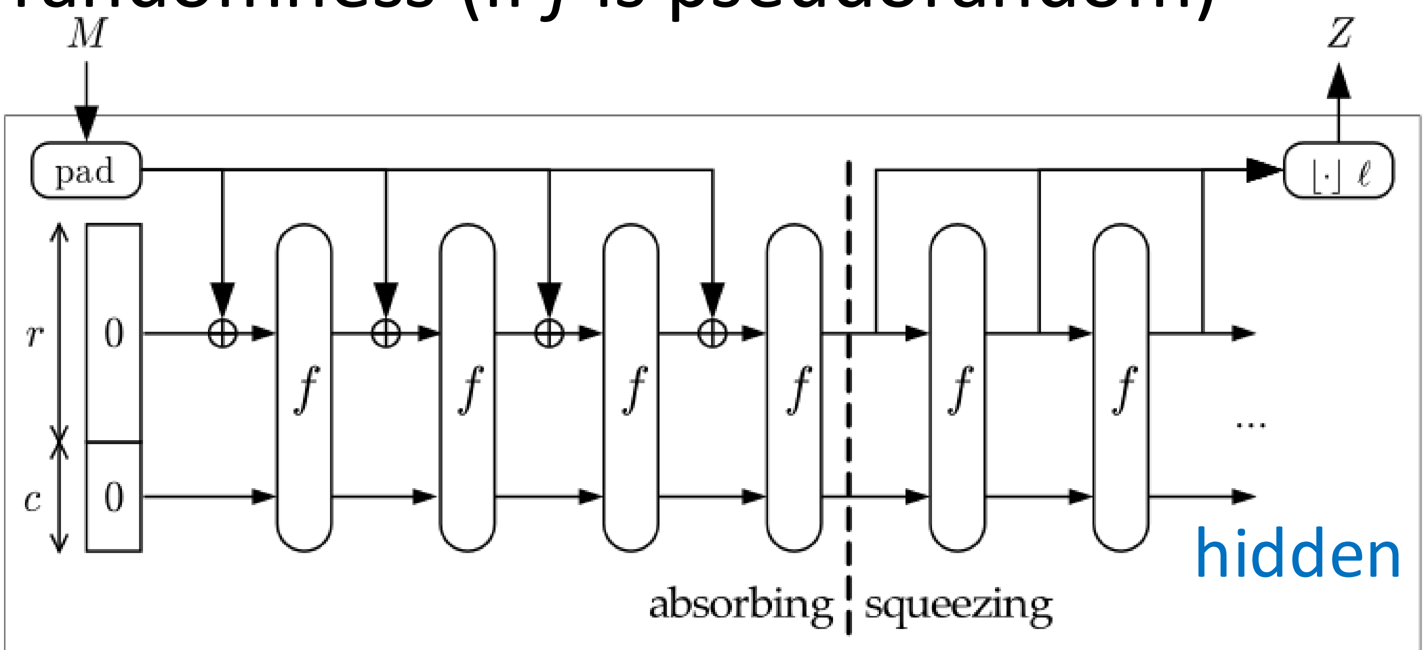


Length Extension Attack

- Applicable to all Merkle-Damgård constructions including ~~MD5, SHA1~~, SHA2
- Not a show-stopper as they do not affect OW and CR
- If you need pseudorandomness, use SHA3 (not a Merkle-Damgård construction)

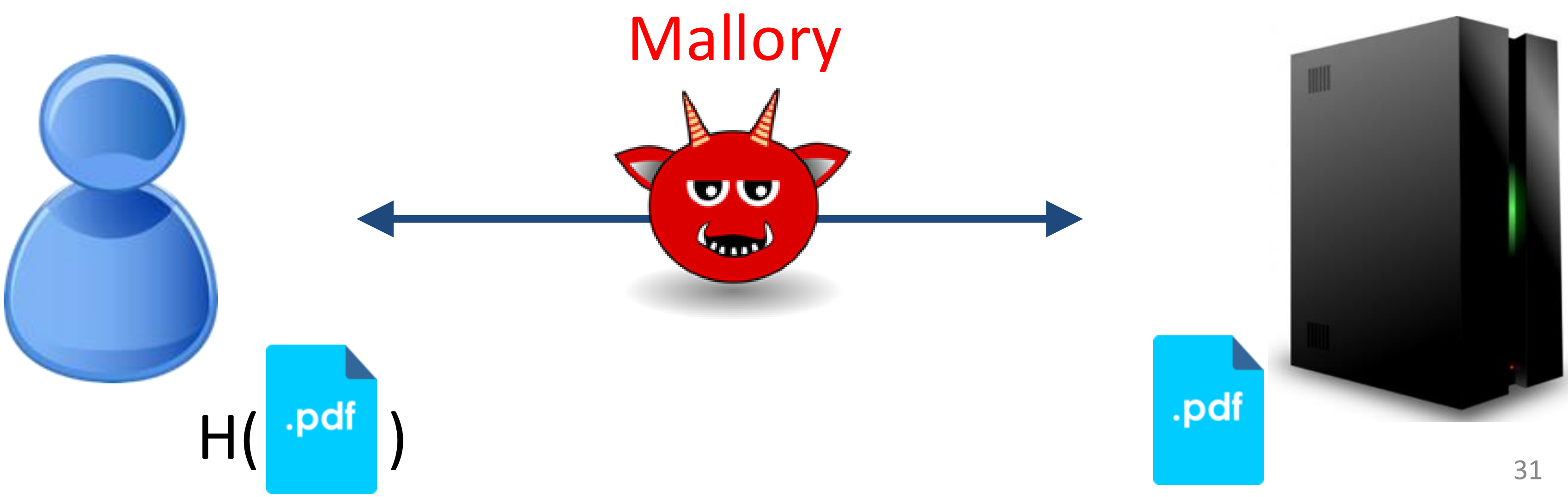
SHA3: Sponge Construction

- Final hash output does not expose the entire internal state → cannot length-extend
- Along with other techniques, achieves pseudo-randomness (if *f* is pseudorandom)

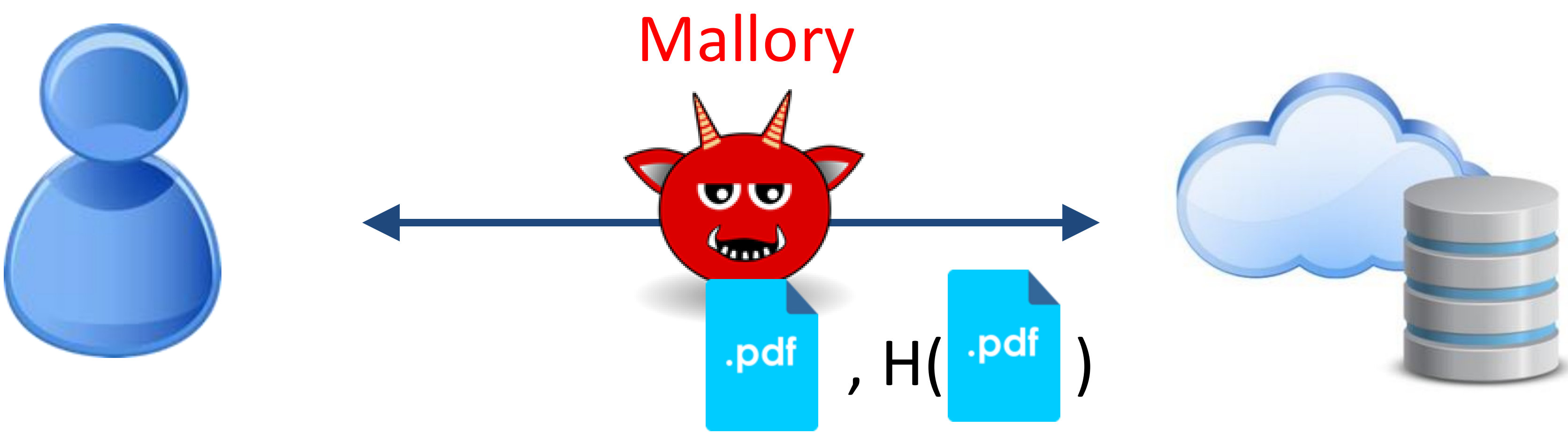


Recall External Storage Application

- Integrity of remote/external storage
  - User computes and stores H(file) locally
  - Compare hash upon download
  - Collision-resistance protects integrity

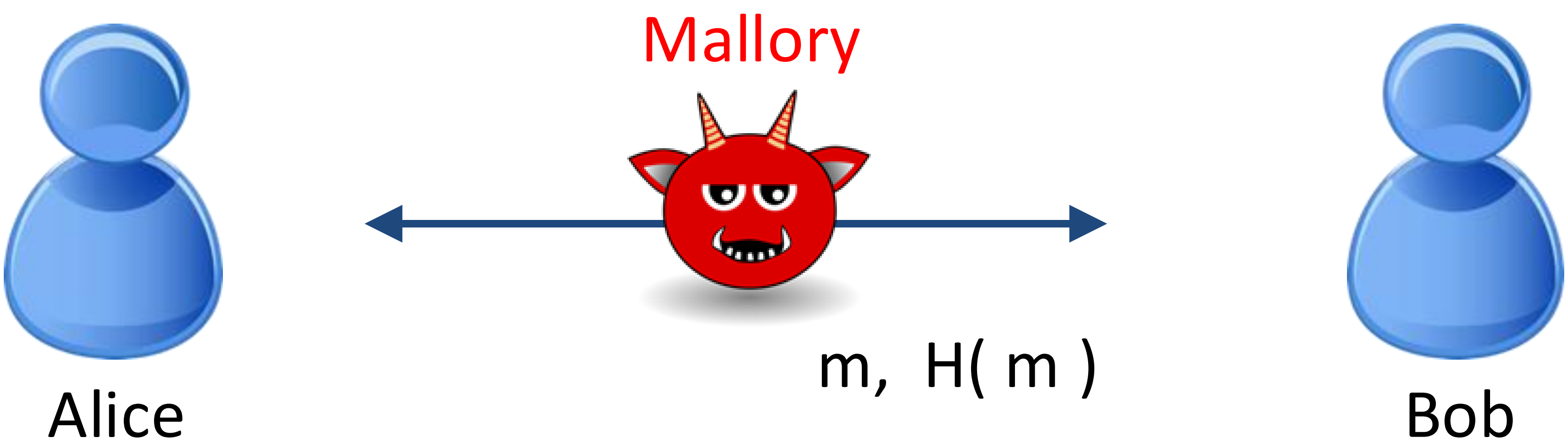


Integrity of Download?



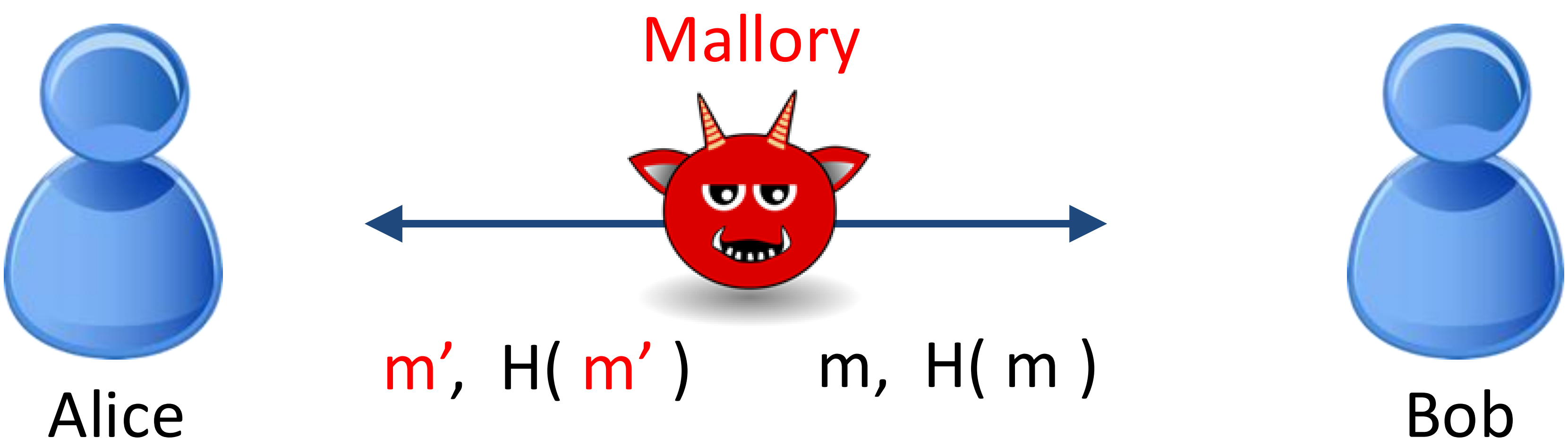
Integrity of Communication?

- I.e., message authentication



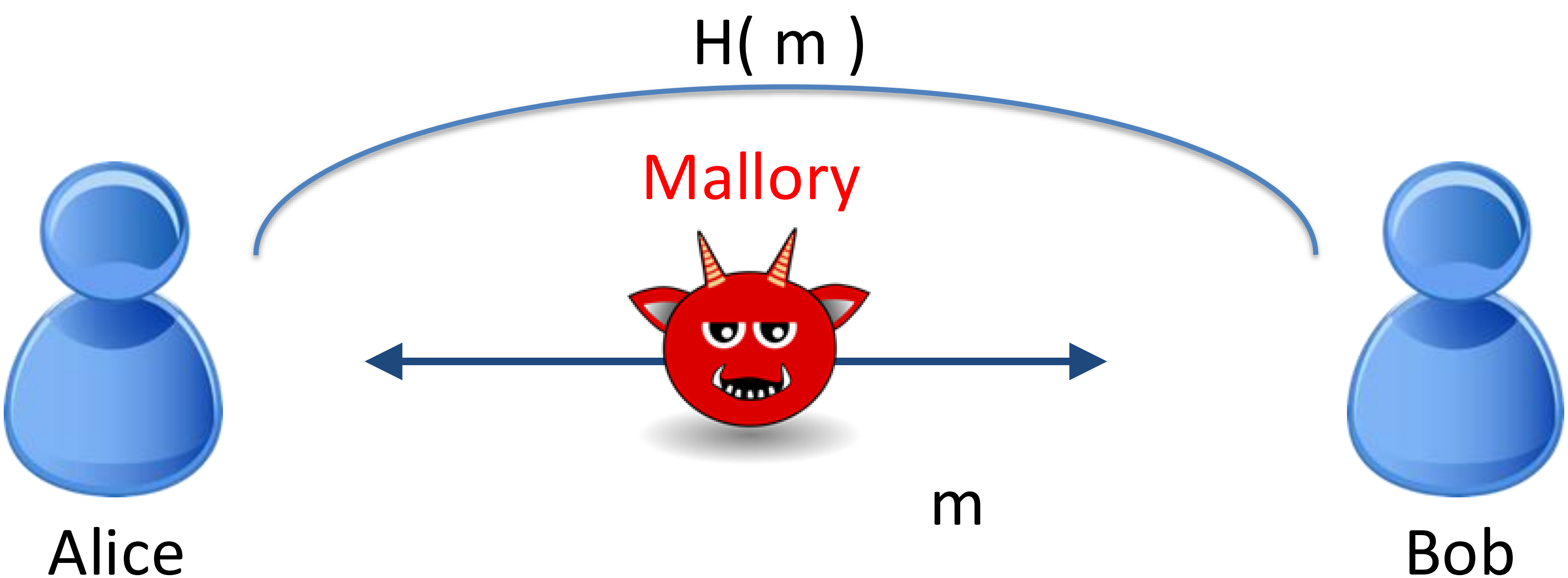
Message Authentication

- In its simplest form, a cryptographic hash does NOT work for a message authentication. Attacker can hash the modified message.



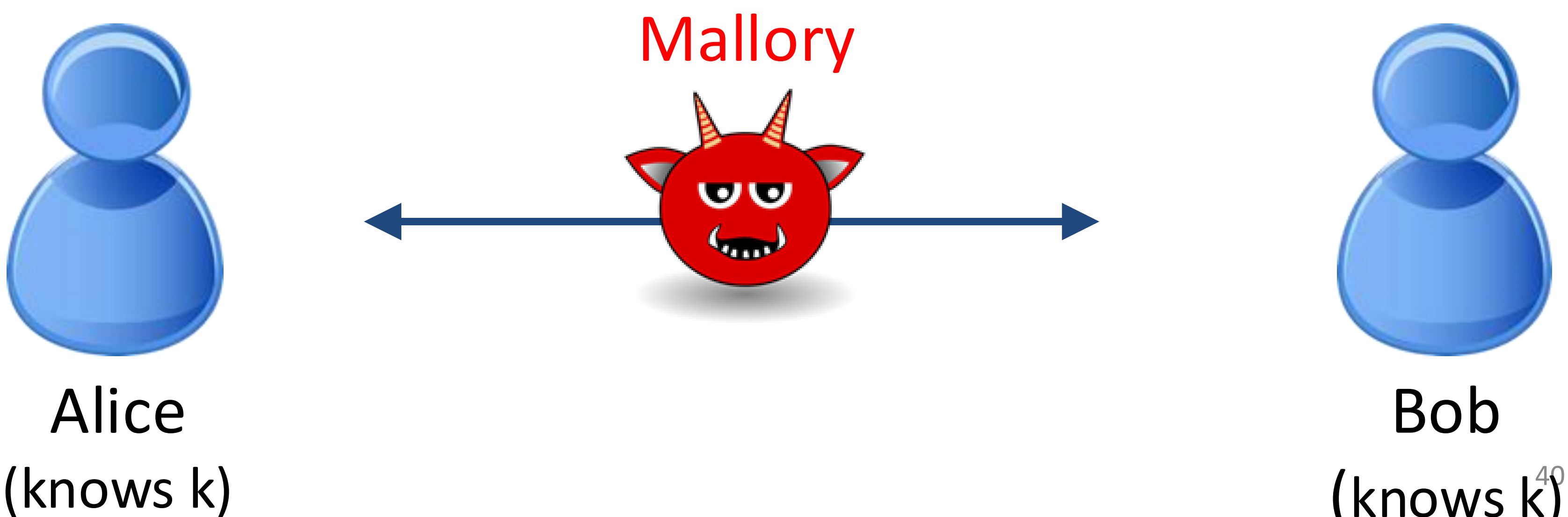
Message Authentication

- Two settings it can work:
  - If the hash can be transmitted in another *trustworthy* but low-bandwidth channel



Message Authentication

- Two settings it can work:
  - If the hash can be transmitted in another *trustworthy* but low-bandwidth channel, or
  - If Alice and Bob share a secret key k
    - Will come back to this in a future lecture



Another Application

- Let’s play a game online: if you guess my favorite 2-digit number in one try, you get A+
  – You will never win ☺

- Need a “sealed envelop” for a fair game
  – Alice sends Bob c = Commit(m)
  – Alice later “opens” m for Bob to verify against c
  – Hiding: Bob cannot find out m from c
  – Binding: Alice cannot open to another m’ ≠ m

Commitment

- Alice sends Bob c = Commit(m) = H(m||r) where r is a long, fixed-length & random string
  - Why do we need r?
- Alice can later reveal m and r to “open”

- Hiding: Bob cannot find m from c
  - If H is pseudorandom (OW is insufficient, why?)
- Binding: Alice cannot open to another m’ ≠ m
  - If H is collision-resistant

Summary

- Cryptographic hash function:
  - Definition H: $\{0,1\}^* \rightarrow \{0,1\}^n$
  - Desired properties: one-way, collision-resistant, pseudorandom (behave like a random oracle)
  - Currently recommended: SHA3 (SHA2 if must)
  - Paradigms: Merkle-Damgård and sponge
    - Note length extension attacks for Merkle-Damgård!
  - Applications: password hashing, external storage, commitment, hash-based message authentication (need extra assumptions), …