رمزنگاری، امنیت اطلاعات و حریم خصوصی

ارائه: دکتر سیدعلی لاجوردی

بخش پنجم

Hash function

Hash functions

- (Cryptographic) hash function: deterministic function mapping arbitrary length inputs to a short, fixed-length output

- Hash functions can be keyed or unkeyed
  - Theoretically, need to be keyed (as in book)
    - Key is public
  - In practice, hash functions are unkeyed
  - Assume unkeyed hash functions for simplicity

Collision-resistance

- Let H: $\{0,1\}^* \rightarrow \{0,1\}^l$ be a hash function
- A collision is a pair of distinct inputs x, x' such that H(x) = H(x')

- H is collision-resistant if it is infeasible to find a collision in H

Generic hash-function attacks

- What is the best “generic” collision attack on a hash function H: {0,1}* $\rightarrow$ {0,1}l ?
  - Note that collisions are guaranteed to exist…

- If we compute H(x1), …, H(x2l + 1), we are guaranteed to find a collision (why?)
  - Can we do better?

“Birthday” attacks

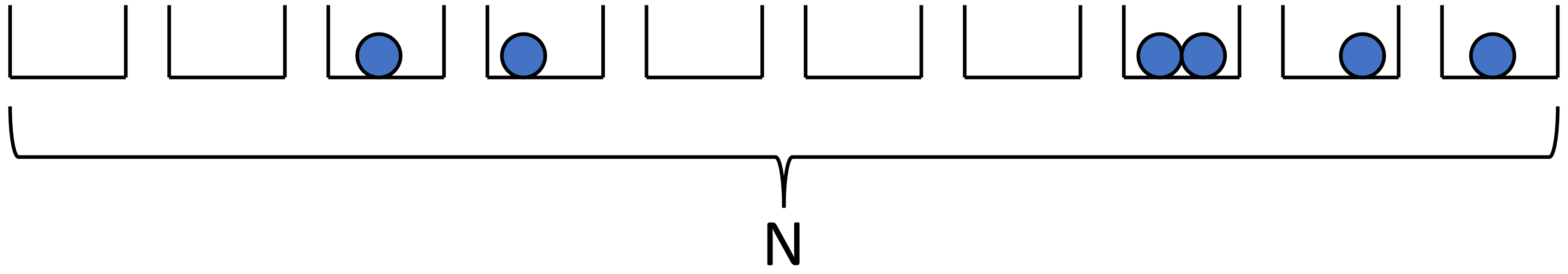
- Compute H(x1), …, H(xt)
  - What is the probability of a collision (as a function of t)?

- Related to the so-called birthday paradox
  - How many people are needed so there is a 50% chance that some two people share a birthday?

Bins: days of the year (N=365)
Balls: k people

Bins: values in $\{0,1\}^{\ell}$ ($N = 2^{\ell}$)
Balls: k hash-function computations

How many balls do we need
to have a 50% chance of a collision?



“Birthday” attacks

- Theorem: the collision probability is Θ(t2/N)

- When t $\approx$ N1/2, probability of a collision is $\approx$ 50%
  - Birthdays: 23 people suffice!
  - Hash functions: O(2l/2) hash-function evaluations

- Need l = 2n to get security against attackers running in time 2n
  - Note: twice as long as symmetric keys (e.g., block-cipher keys or PRG seeds) for the same security

“Birthday bound”

- The birthday bound comes up in many other cryptographic contexts

- Example: IV reuse in CTR-mode encryption
  - If k messages are encrypted, what are the chances that some IV is used twice?
  - Note: this is much higher than the probability that a specific IV is used again

Building a hash function

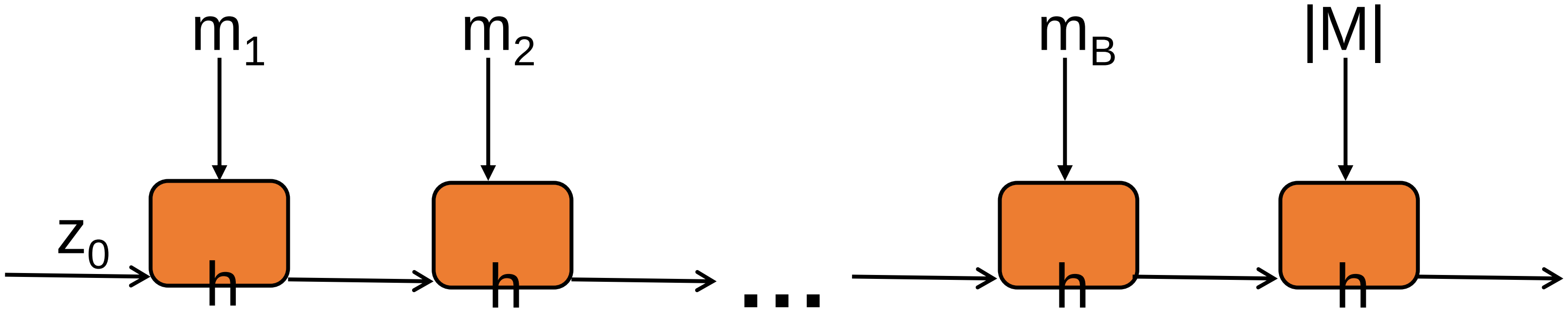
- Two-stage approach
    - Build a compression function h
        - I.e., hash function for fixed-length inputs
    - Build a full-fledged hash function (for arbitrary length inputs) from a compression function h

Building a hash function

- For now…
  - Assume we have a “good” compression function h
    - I.e., collision-resistant for fixed-length inputs
  - Will discuss how to construct such an h later

- Construct a hash function H (for arbitrary length inputs) based on h
  - Prove that collision resistance of h implies collision resistance of H

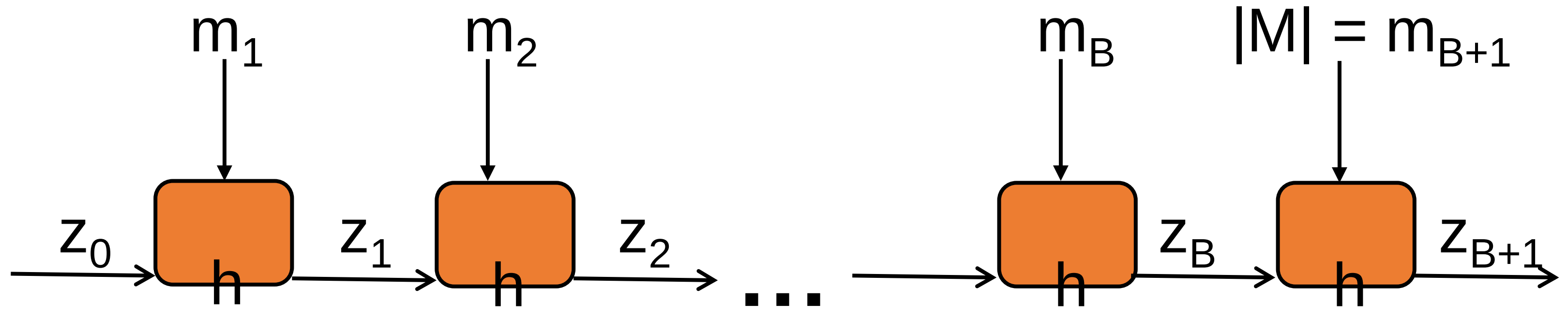
Merkle-Damgard transform



Note: $M = m_1 \ldots m_B$ is padded with 0s if necessary

Merkle-Damgard transform

- Claim: if h is collision-resistant, than so is H



- Proof: Collision in H $\Rightarrow$ collision in h
  - Say H(m1, …, mB) = H(m’1, …, m’B’)
  - $|M| \neq |M'|$, obvious
  - $|M| = |M'|$, look at largest i with (zi-1, mi) $\neq$ (z’i-1, m’i)

Hash functions in practice

- MD5
  - Developed in 1991
  - 128-bit output length
  - Collisions found in 2004, should no longer be used

- SHA-1
  - Introduced in 1995
  - 160-bit output length
  - Collision found by brute force in 2017
  - Subsequent improvements in attacks; no longer recommended; should migrate to SHA-2

Hash functions in practice

- SHA-2
  - Introduced in 2001
  - Versions with 224, 256, 384, and 512-bit outputs
  - No significant known weaknesses

- SHA-3/Keccak
  - Result of a public competition from 2008-2012
  - Very different design than SHA-1/SHA-2
    - Does not use Merkle-Damgard transform
  - Supports 224, 256, 384, and 512-bit outputs

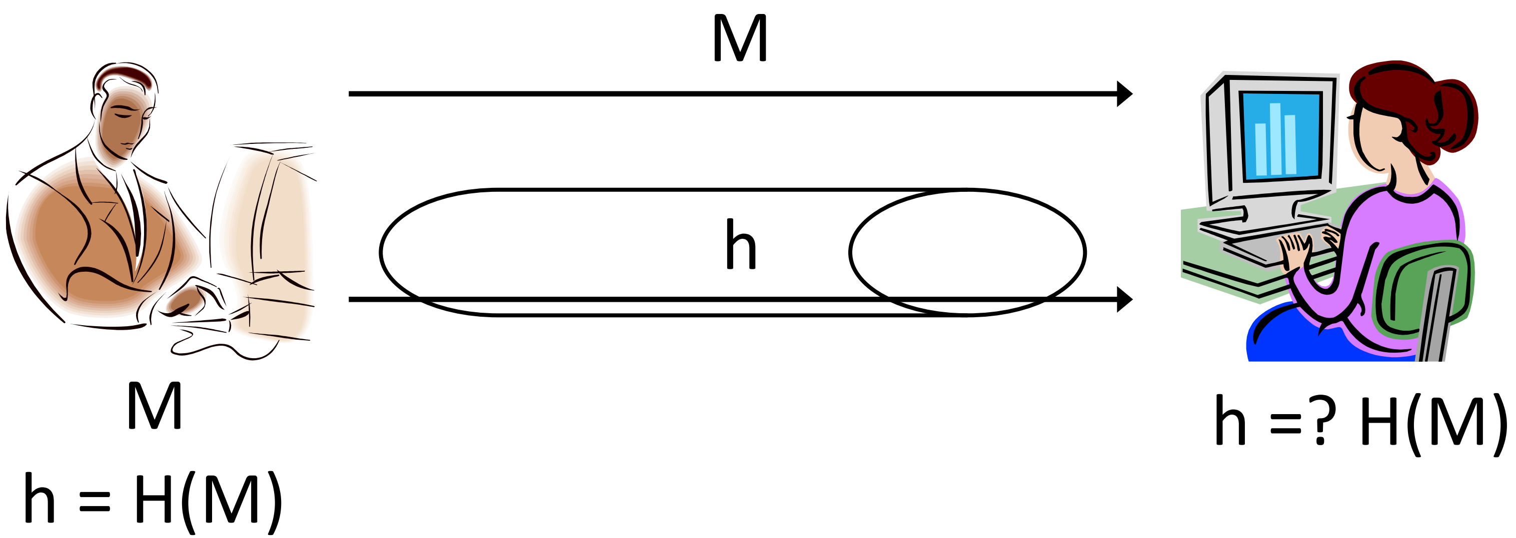
Applications of hash functions to message authentication

Recall…

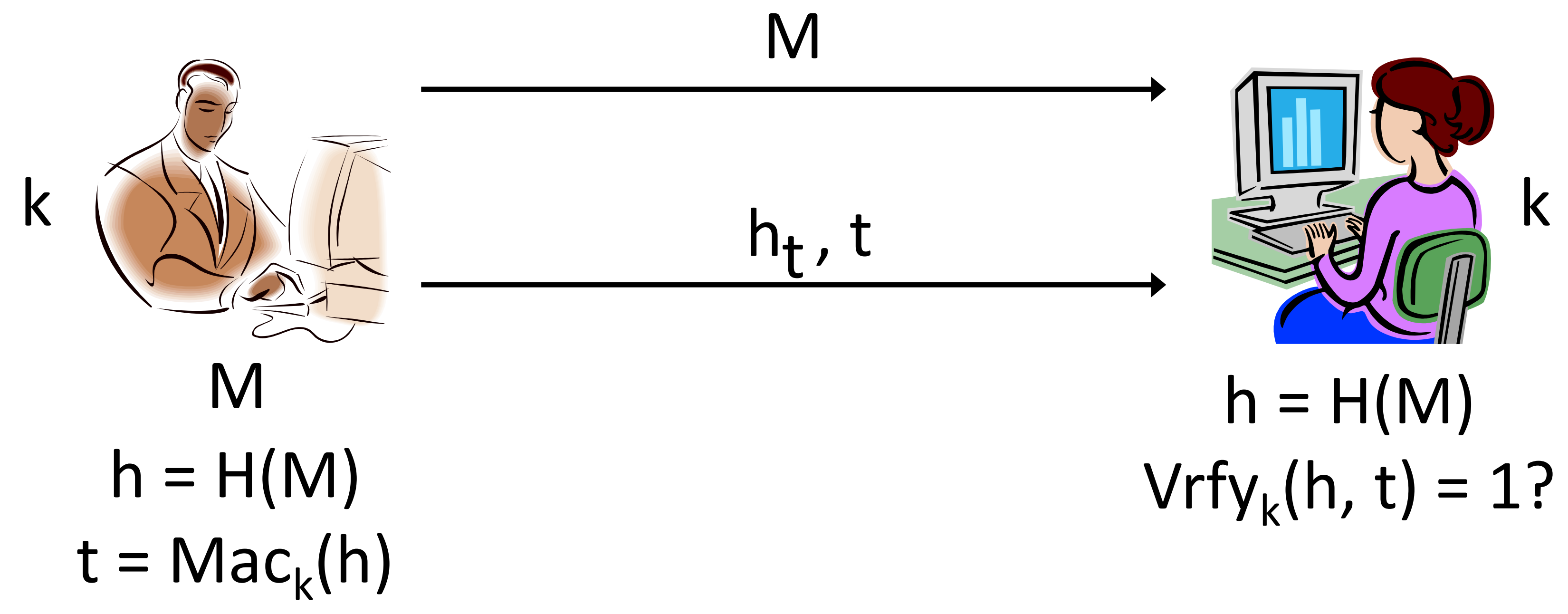
- We showed how to construct a secure MAC for short, fixed-length messages based on any PRF/block cipher

- We want to extend this to a secure MAC for arbitrary-length messages
  - Before: using CBC-MAC
  - Here: using hash functions

Intuition…



Hash-and-MAC



Security?

- If the MAC is secure for fixed-length messages and H is collision-resistant, then the previous construction is a secure MAC for arbitrary-length messages

Proof sketch

- Say the sender authenticates messages m1, m2, …
  - Let hi = H(mi)
- Attacker outputs forgery (m, t), m $\neq$ mi for all i
  - Let h = H(m)

- Two cases:
  - h = H(m) = hi = H(mi) for some i
    - Collision in H!
  - H(m) = h $\neq$ hi for all i
    - Forgery in the underlying, fixed-length MAC

Instantiation?

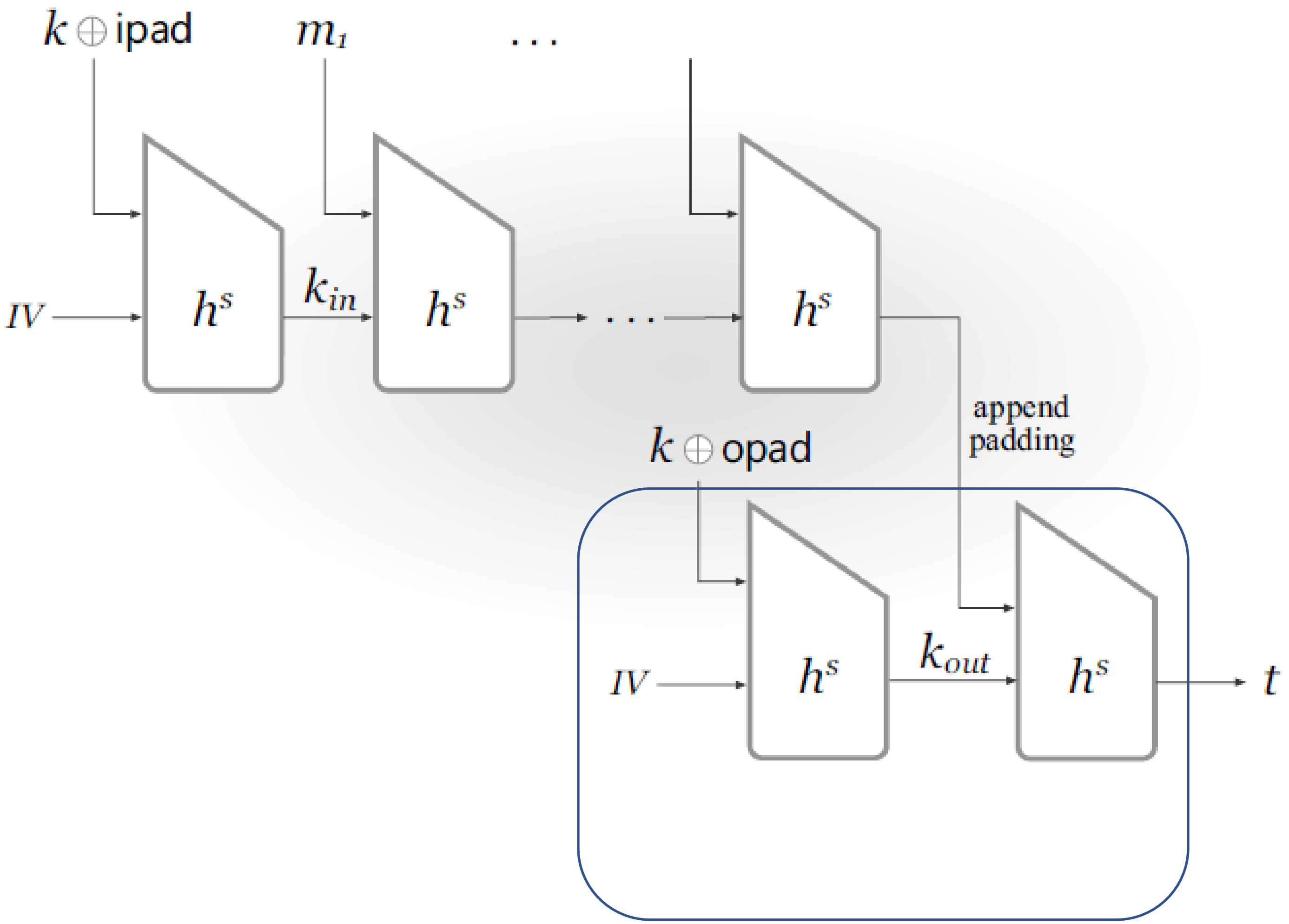
- Hash function + block-cipher-based MAC?
  - Block-length mismatch (e.g., if using AES as the block cipher)
  - Need to implement two crypto primitives (block cipher and hash function)

HMAC

- Constructed entirely from Merkle-Damgard hash functions
  - MD5, SHA-1, SHA-2
  - Not SHA-3

- Can be viewed as following the hash-and-MAC paradigm
  - With (part of the) hash function being used as a pseudorandom function

HMAC



$k \oplus$ ipad
m_1
...
IV
h^s
k_{in}
h^s
...
h^s
append
padding
$k \oplus$ opad
IV
h^s
k_{out}
h^s
t

Other applications of hash functions

Hash functions are ubiquitous

- Collision-resistance $\Rightarrow$ "fingerprinting"
- Outsourced storage
- Used as a "random oracle"
- Used as a one-way function
  - Password hashing
- Key derivation

Fingerprinting

- E.g., hash-and-MAC
- E.g., virus scanning
- E.g., deduplication
- E.g., file integrity
  - Assuming it is possible to get a reliable copy of $H(x)$ for file x
  - Note: different from integrity in the context of message-authentication codes

Outsourced storage

$x_1, \ldots, x_n$

$x_1, \ldots, x_n \longrightarrow$

$h_i = H(x_i)$

$i \longrightarrow$

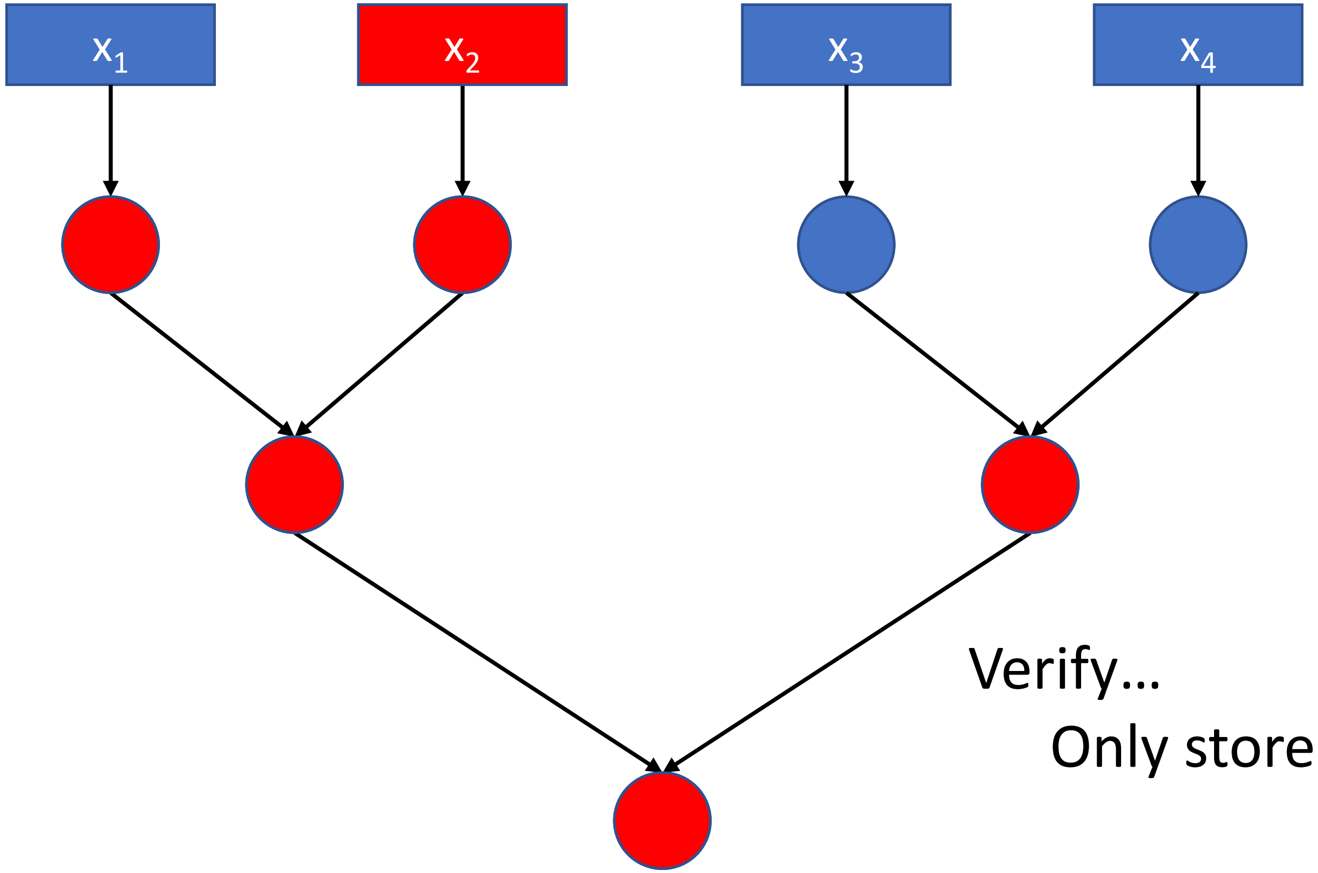
$\longleftarrow x_i$

$H(x_i) =? h_i$

O(n) client storage!

Merkle tree

- Using a Merkle tree, we can solve the outsourcing problem with O(1) client storage and |x| + O(log n) communication



Verify…

Only store the root!

The random-oracle (RO) model

- Treat H as a public, random function

- Then H(x) is uniform for any x…
  - …unless the attacker computes H(x) explicitly

- This implies collision resistance (if output is large enough)
  - Much stronger than collision resistance

The RO model

- Intuitively
  - Assume the hash function “is random”
  - Models attacks that are agnostic to the specific hash function being used
  - Security in the real world as long as “no weaknesses found” in the hash function
- Formally
  - Choose a uniform hash function as part of the security experiment
  - Attacker can only evaluate H via explicit queries to an oracle
  - Simulate H as part of the security proof
- Different from a PRF
  - There is no key here

Pros and cons of the RO model

- In practice
  - Prove security in the RO model
  - Instantiate the RO with a “good” hash function
  - Hope for the best…
- Cons
  - There is no such thing as a public hash function that “is random”
    - Not even clear what this would mean, formally
  - Known counterexamples
    - There are (contrived) schemes secure in the RO model, but insecure when using any real-world hash function
- Pros
  - No known example of “natural” scheme secure in the RO model being attacked in the real world
  - If an attack is found, just replace the hash
  - Proof in the RO model better than no proof at all
    - Evidence that the basic design principles are sound

Many applications of random oracles

- Password hashing
- Key derivation
- Will see many more in the context of public-key cryptography

Password hashing

- Server stores H(pw) instead of pw
  - (Ignore “salting” here)

- Recovering pw from H(pw) in q tries should be as hard as guessing pw in q tries
  - Even if the distribution of pw is non-uniform

Key derivation

- Consider deriving a (shared) key k from (shared) high-entropy information x
  - E.g., biometric data

- Cryptographic keys must be uniform, but shared data is only high-entropy

Min-entropy

- Let X be a distribution

- The min-entropy of X (measured in bits) is
  $$H_\infty(X) = -\log \max_x \{ \Pr[X=x] \}$$
  - I.e., if $H_\infty(X) = n$, then the probability of guessing x sampled from X is (at most) 2^{-n}

- Min-entropy is more suitable for crypto than standard (Shannon) entropy

Key derivation

- Given shared information x (sampled from distribution X), derive shared key k=H(x)
  - In what sense can we claim that k is a good (i.e., uniform) cryptographic key?

- If H is a random oracle, then H(x) is uniform as long as the attacker does not query x to H
  - …but the attacker cannot do that (with high probability) if X has high min-entropy!