Bitcoin:
Programming the Future of Money

Topics in Computer Science - ITCS 4010/5010, Spring 2025

Dr. Christian Kümmerle

Lecture 5

Digital and Cryptographic Money II

UNIVERSITY OF NORTH CAROLINA
CHARLOTTE

The Quest for Digital Cash

KEY INNOVATIONS TOWARDS DIGITAL CASH SYSTEMS

- Chaumian e-Cash
- E-Gold
- Hashcash
- Bit Gold

Q: What were the
- main innovations?
- main limitations?

KEY INNOVATIONS TOWARDS DIGITAL CASH SYSTEMS

- Chaumian e-Cash
- E-Gold
- Hashcash
- Bit Gold
- **b-Money** (proposed by Wei Dai in 1998)
  - Each participant broadcasts transactions and maintains a copy of state

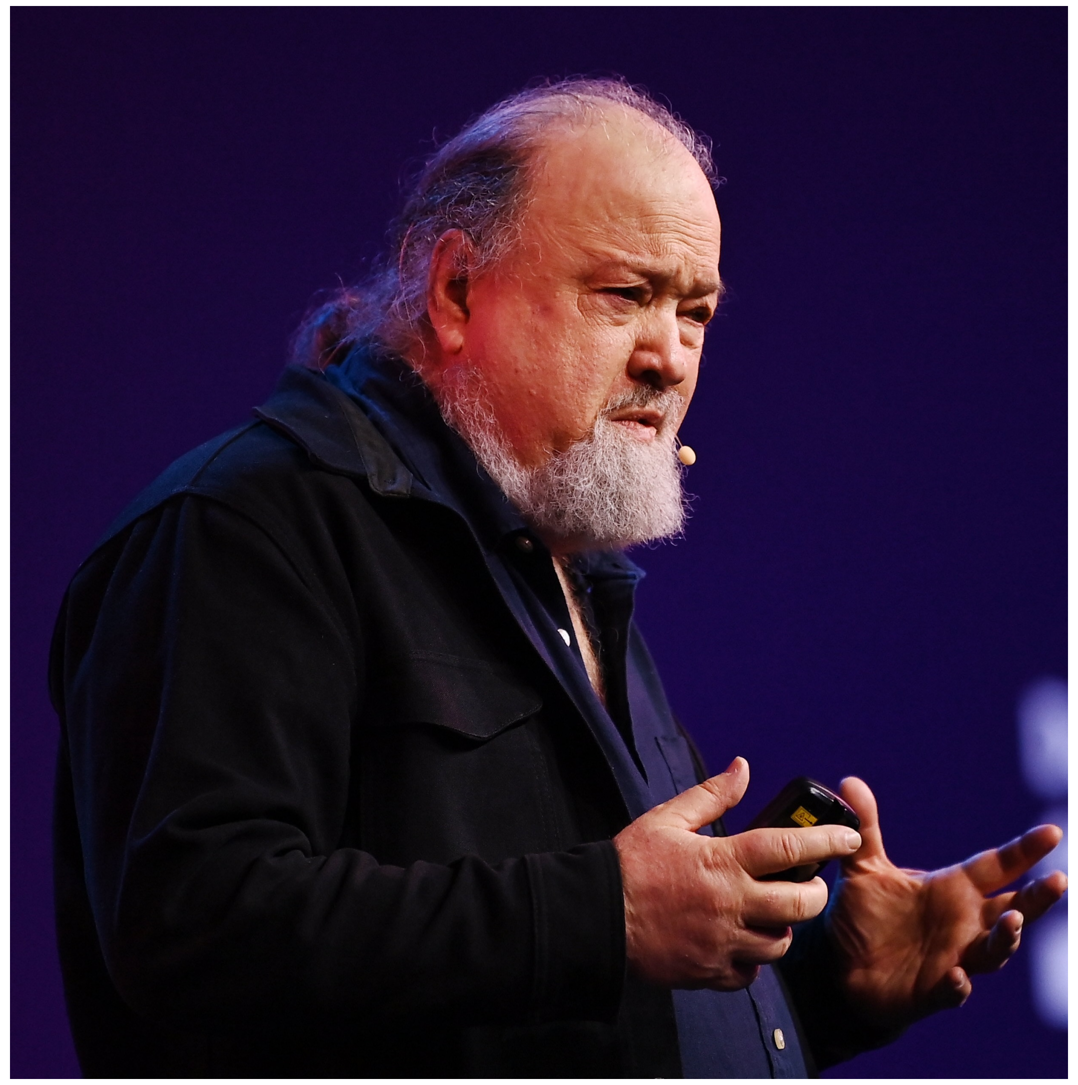
  - Difficulty of hashing regulated by consumer price index
- **Reusable Proof-of-Work (RPOW),** proposed by Hal Finney in 2004

  - Hashes are created through PoW or validated & reissued against double spent by server

  - Central server with “secure component” checks on validity of hashes

CHAUMIAN ECASH

- Proposed in 1983 by David Chaum
- Addresses fundamental problem:
  How to avoid a “double spend” in private transactions?



David Chaum (1955-)

PROS AND CONS OF CHAUMIAN E-CASH

Pros

- Digital payments
- Peer-to-peer
- Privacy
- Offline double-spend detection

Cons

- Bank can censor withdrawals and deposits

Detailed breakdown (on whiteboard) of how to use within Chaumian e-Cash:

- Withdrawal (Obtain e-Cash, debit account at bank)
- Payment (send e-Cash)
- Deposit (deposit e-Cash, credit account at bank)

Also check PDF lecture notes on Canvas.

Digital signatures

- Enable each user to sign a message, and everyone can verify this user's signature of the message.

If user is called Alice, we are given

- Signing function $S_{Alice}: \mathcal{M} \to \Sigma$ that maps message $m \in \mathcal{M}$ to signature $\sigma \in \Sigma$ of a finite signature space Σ.
- Verification function $V_{Alice}: \mathcal{M} \times \Sigma \to \{True, False\}$

s.t. $V_{Alice}(m, \sigma) = \begin{cases} True & \text{if } \sigma = S_{Alice}(m) \\ False & \text{else.} \end{cases}$

Example:
- Elliptic Curve Digital Signature Algorithm (ECDSA)
- RSA (Rivest - Shamir - Adleman)

Blind Digital Signatures

We are given s_{Alice} and v_{Alice} for user Alice.

Now: There exists

- blinding function $b_{Alice,Bob} : \mathcal{M} \to \mathcal{M}$
- unblinding function $u_{Alice,Bob} : \Sigma \to \Sigma$

} both private to Bob

such that

$$u_{Alice,Bob}\left(s_{Alice}\left(b_{Alice,Bob}(m)\right)\right) = s_{Alice}(m) \quad \forall m \in \mathcal{M}.$$

Chaumian e-Cash: *Withdrawal Protocol: Bob withdraws e-cash worth \$1 from bank Alice.*

Input: Availability of \$1 in account of Bob at bank Alice.
Output: Bob has e-cash note information (m, σ) available or False

Bob chooses a random message $m \in \mathcal{M}$.
Bob blinds message m by computing $b_{\text{Alice,Bob}}(m)$.
Bob forwards $b_{\text{Alice,Bob}}(m)$ to bank Alice.
Bank Alice signs blinded message by computing $s_{\text{Alice}}(b_{\text{Alice,Bob}}(m))$.
Bank Alice sends $c = s_{\text{Alice}}(b_{\text{Alice,Bob}}(m))$ back to Bob.
Bank Alice debits account of Bob by \$1.
Bob unblinds the received data by computing $\sigma = u_{\text{Alice,Bob}}(c)$.
Bob checks Alice's signature by applying the verification function $v_{\text{Alice}}(u_{\text{Alice,Bob}}(c), m)$.
if $v_{\text{Alice}}(u_{\text{Alice,Bob}}(c), m) =$ True **then**
    return e-cash note information (m, σ) to Bob. ▹ Withdrawal successful
else
    return False. ▹ Withdrawal unsuccessful, signature not accurate

Chaumian e-Cash: *Payment Protocol: Bob sends e-cash worth* \$1 *to Carol.*

Bob sends $m \in \mathcal{M}$ and σ to Carol.
Carol checks Alice's signature by applying the verification function $v_{\text{Alice}}(m, \sigma)$.
if $v_{\text{Alice}}(m, \sigma) = \text{True}$ **then**
    Carol accepts payments.
else
    return False.   ▷ Payment unsuccessful, as signature not genuine

Chaumian e-Cash: *Deposit Protocol: Carol deposits e-cash worth \$1 at bank Alice.*

Input: List of cleared e-cash notes $\mathcal{L}$ at bank Alice.
Output: Updated list of cleared e-cash notes $\mathcal{L}$ at bank Alice.

Bob sends $m \in \mathcal{M}$ and σ to bank Alice.
Alice checks her own signature by applying the verification function $v_{\text{Alice}}(m, \sigma)$.
if $v_{\text{Alice}}(m, \sigma) = \text{True}$ **then**
  if $m \in \mathcal{L}$ **then**
    Alice denies payment. ▷ Deposit unsuccessful, double spend detected.
  else
    Append m to list of cleared e-cash notes $\mathcal{L}$: $\mathcal{L} = \mathcal{L} \cup \{m\}$.
    Alice accepts deposit.
    Alice credits account of Carol by \$1. ▷ Deposit successful.
else
  return False. ▷ Deposit unsuccessful, signature not genuine.

Cryptographic Hash Functions

HASH FUNCTIONS

D: *domain*, i.e., set on which function is defined

R: *range*, i.e., set in which all outputs of function need to be included in

A hash function $H : D \to R$ is a function that satisfies

1. H takes strings $x \in D$ of **any length** as input
2. Outputs of H (i.e., length elements of R) are of **fixed size**
3. If $x \in \{0{,}1\}^n \in D$, then computing $H(x)$ has a time complexity of $O(n)$ (H is efficiently computable)

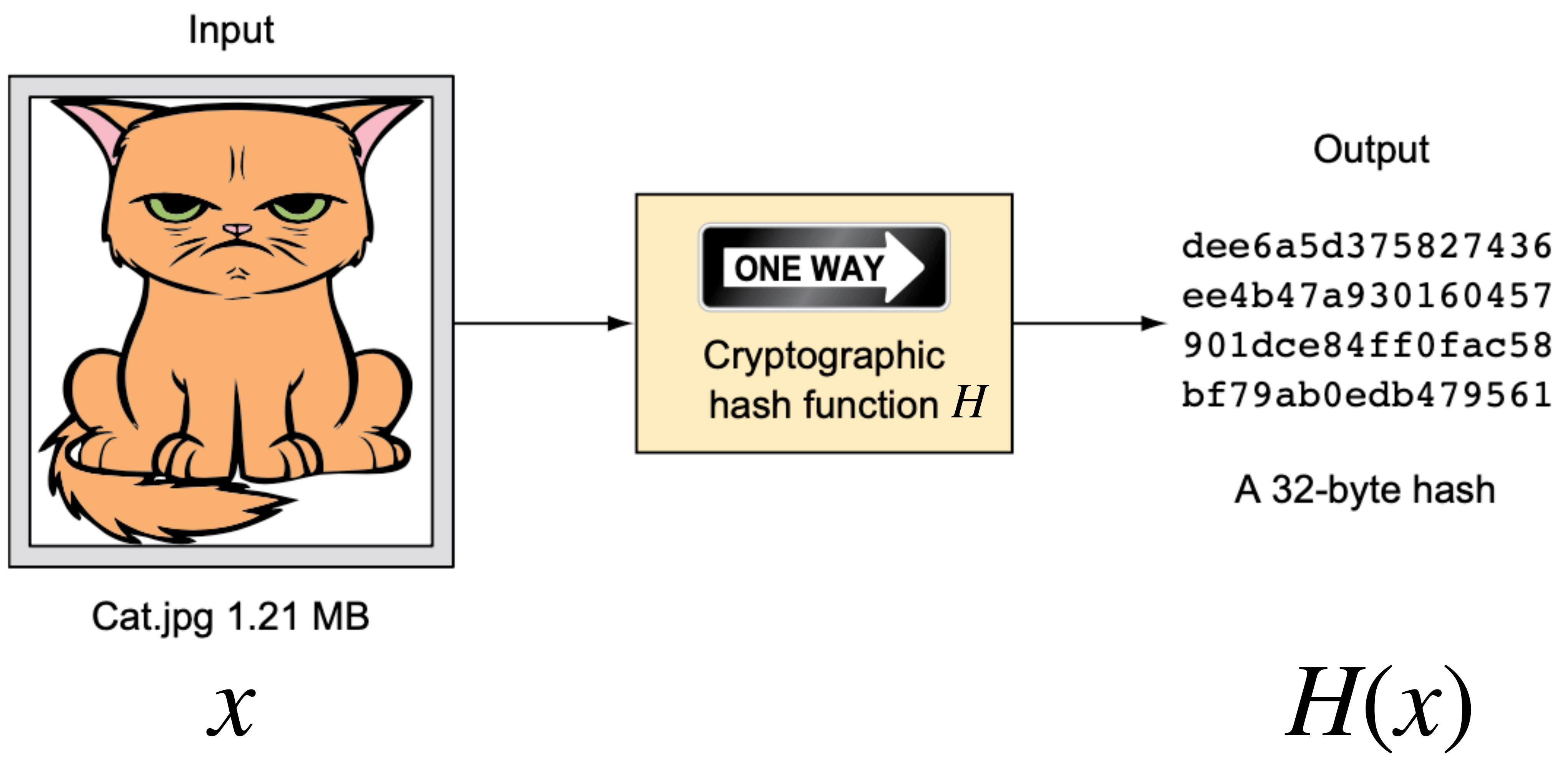
Example: 2. For us often, $R = \{0{,}1\}^{256}$ (256-bit strings)

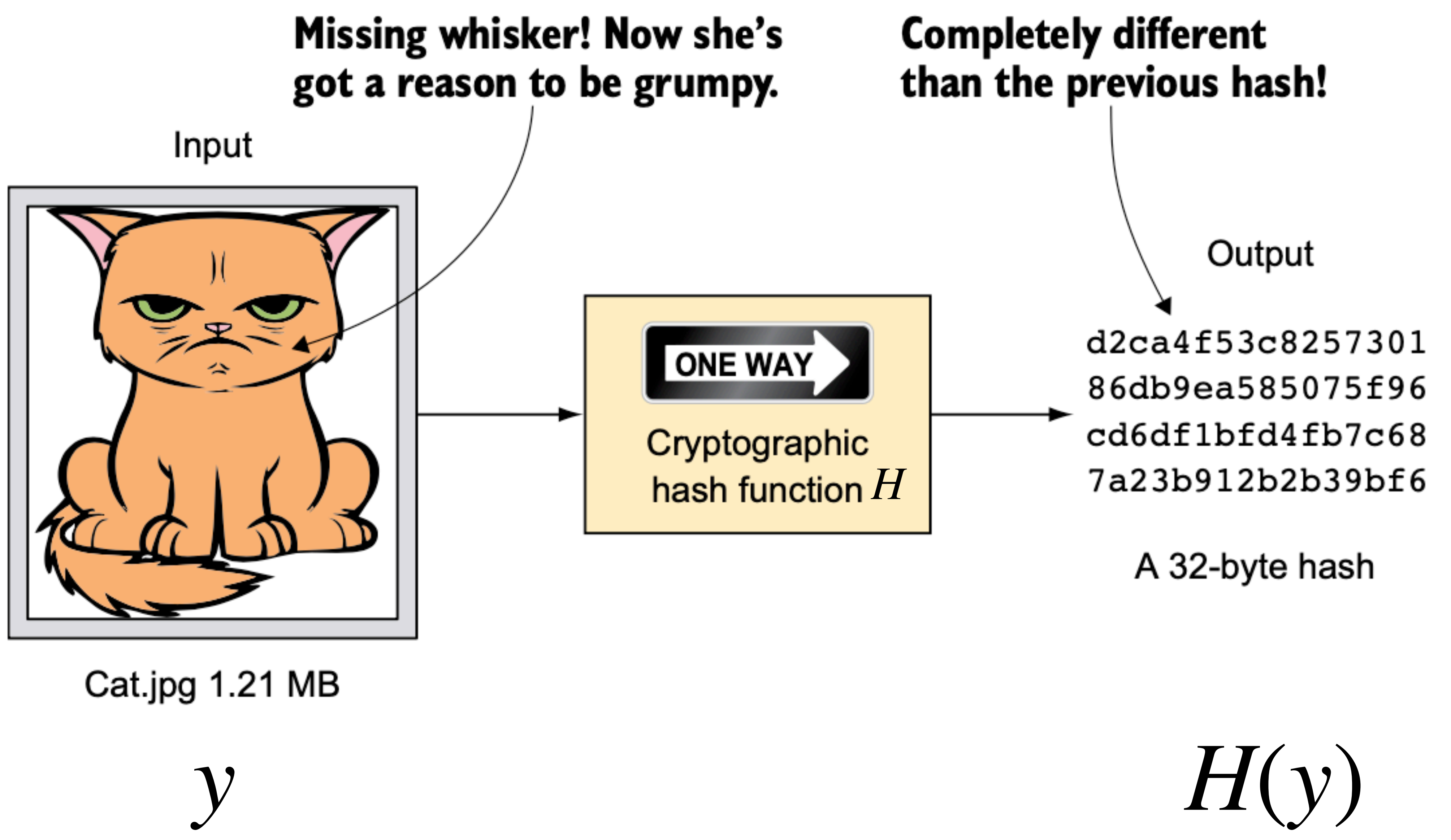
APPLICATIONS OF HASH FUNCTIONS IN THE BITCOIN PROTOCOL

Cryptographic hash functions have multiple purposes for the protocol:

- **Consensus mechanism / bitcoin mining**:
  Central part of cryptographic puzzle to be solved (Double SHA-256)
  -> Ensures agreement on the state
  -> Inflation control
- **Creation of bitcoin addresses from public keys**
- **Checksums** for typed bitcoin addresses (prevention of typos / copy-paste errors)
- **Transaction identification**
- **Construction of transaction "blocks"** via Merkle trees
- **Data integrity** of chain of blocks

APPLICATION 1: MESSAGE DIGESTS

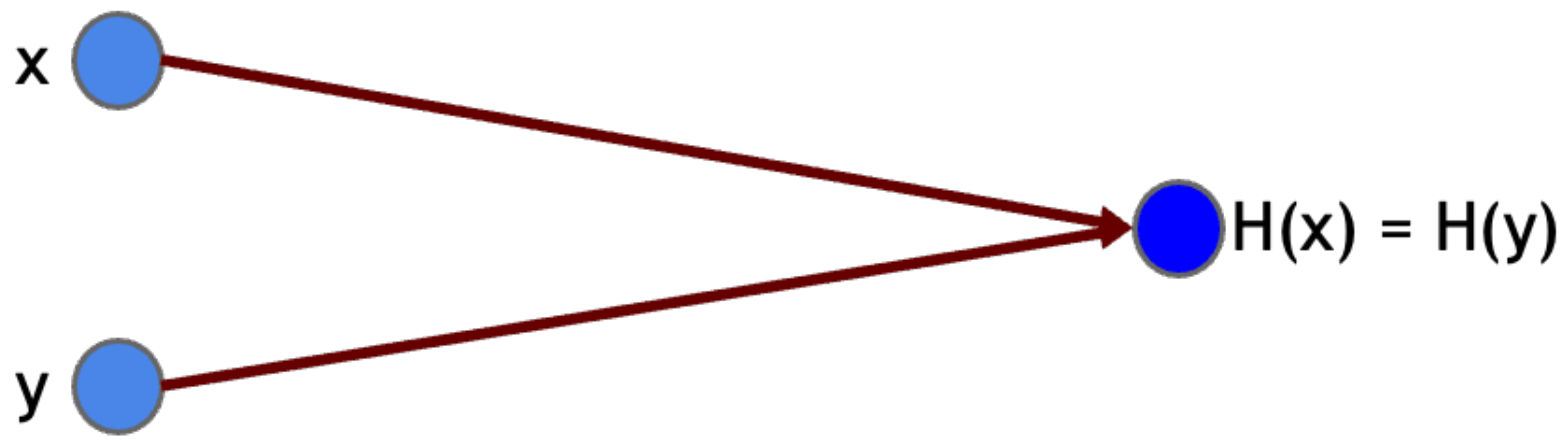




- Integrity of data/message x is encoded into $H(x)$, but $H(x)$ is much smaller than x.
- May help to distinguish x from y even if only $H(x)$ and $H(y)$ are provided.

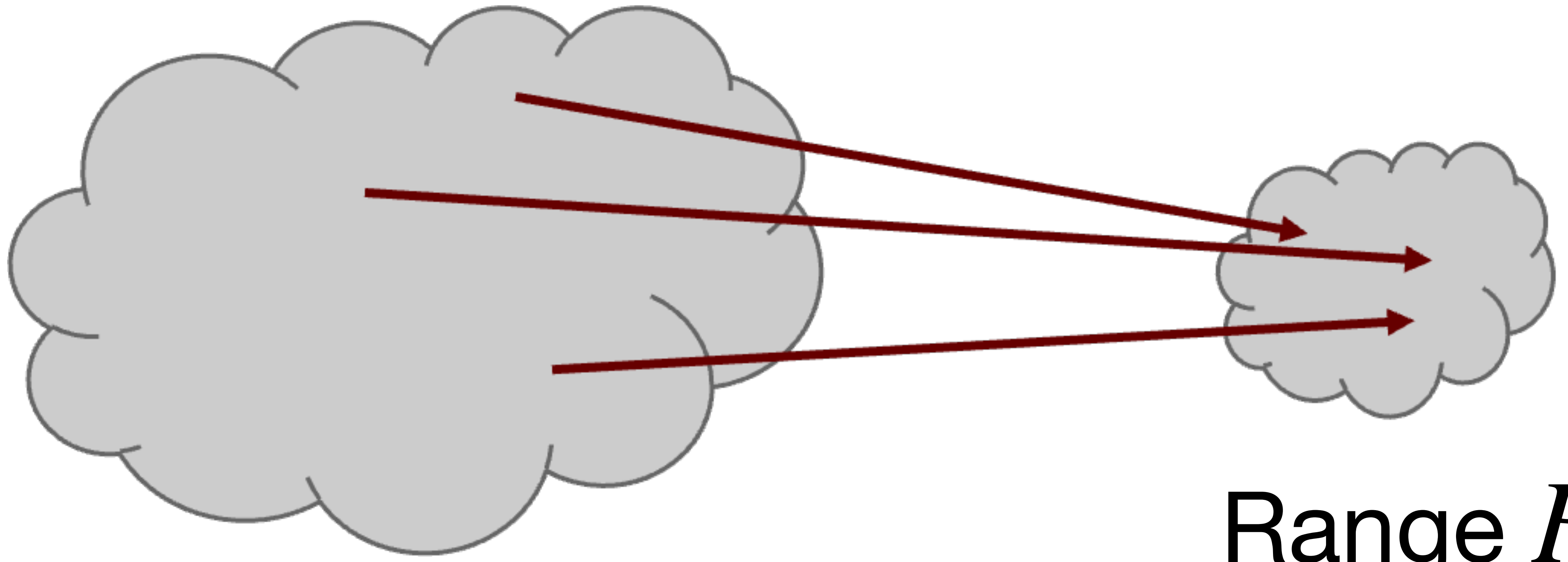
CRPYTOGRAPHIC HASH PROPERTY 1: COLLUSION-FREE

A hash function $H : D \to R$ is called collision-free if it is "infeasible" to find two different inputs $x, y \in D, x \neq y$ with same output $H(x) = H(y)$.



IS COLLUSION-FREENESS ALWAYS VIOLATED?

Collisions do exist …



Domain D
of possible inputs

Range R
of possible outputs

… but is practically feasible to find them
(i.e., in a reasonable amount of time)?

Birthday Paradox

- What is the probability P that there **at least two people** in a room of n people were born on the same day (not considering the year)?
- $P = 1$ if $n > 365$
- $P > 0.5$ if $n > 23$ **if birthdays are uniformly distributed**
  (the threshold θ can be approximated as $\theta \sim \sqrt{365}$)

SHA256 hashes per second for different hardware:

- Standard current MacBook Pro:
  ≈ 2000 MH/s = $2 \cdot 10^9$ H/s
- One state-of-the-art Bitcoin mining Application-Specific Integrated Circuit (ASIC) (Bitmain Antminer S21 XP Immersion, Released in June 2024)
  $\approx$ 300 TH/s = $3 \cdot 10^{14}$ H/s
- Entire Bitcoin mining network
  $\approx$ 600 EH/s = $6 \cdot 10^{20}$ H/s

HOW TO BREAK COLLISION-FREENESS

How long does the entire Bitcoin network need to mine to find a SHA-256 collusion using the “birthday attack”?

Is there a faster way to find collisions?

- For some possible hash functions H: Yes!
- For others (such as SHA-256), we don’t know of one.
- No hash function H has been mathematically proven to be “practically collision-free”.

CRPYTOGRAPHIC HASH PROPERTY 2: HIDING

A hash function $H : D \to R$ is called hiding if it is "infeasible" to find x given $H(r||x)$, where r is chosen from a probability distribution $\mathscr{P}$ with high min-entropy $\log \frac{1}{P_{\text{max}}}$, where $P_{\text{max}} = \max_i p_i$ and p_i is the probability of the i-th outcome of $\mathscr{P}$.

Example: Uniform distribution on $\{0,1\}^n$ has high min-entropy

CRPYTOGRAPHIC HASH PROPERTY 2: HIDING

Ideally, we would like to have something like this:

Given H(x), it is infeasible to find x.

Problem:

H("heads")

H("tails")

easy to find x!

Commitment Scheme

- *commit(msg, nonce)*, returns output *com*
  Takes message msg and random nonce as input and returns commitment *com*.

  Nonce: "Truly" random number that should only be used once.
- *verify(com, msg, nonce),* returns Boolean output. "Opens envelope"
  verify(com, msg, nonce) == True: If *com == commit(msg, nonce).*
  verify(com, msg, nonce) == False: Otherwise.

"Seals message" by computing and publishing *com*

"Open envelope" by publishing *msg and nonce,* as anyone can check validity using *verify()*

Desired properties:

- Hiding property & Binding Property