CSE446: Blockchain & Cryptocurrencies

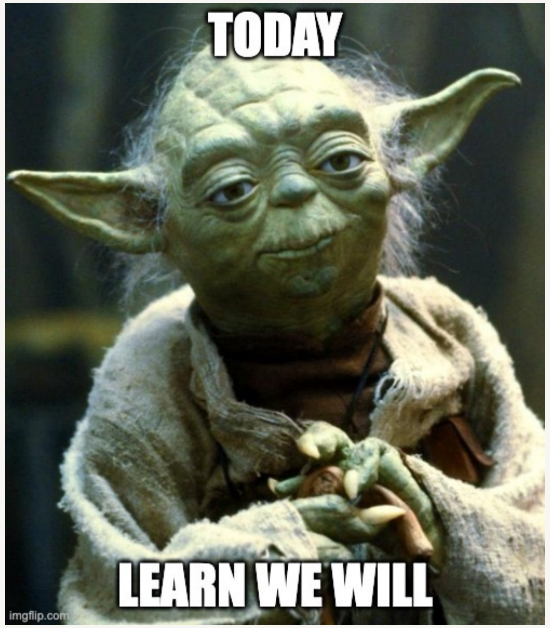
Lecture – 2: Cryptography Review

BRAC UNIVERSITY

Inspiring Excellence

Agenda

- Cryptography review
  - Cryptographic hash functions
  - Symmetric encryption
  - Asymmetric encryption (Public-key encryption)
  - Digital signature
  - Merkle tree



This lecture has been prepared from multiple sources:

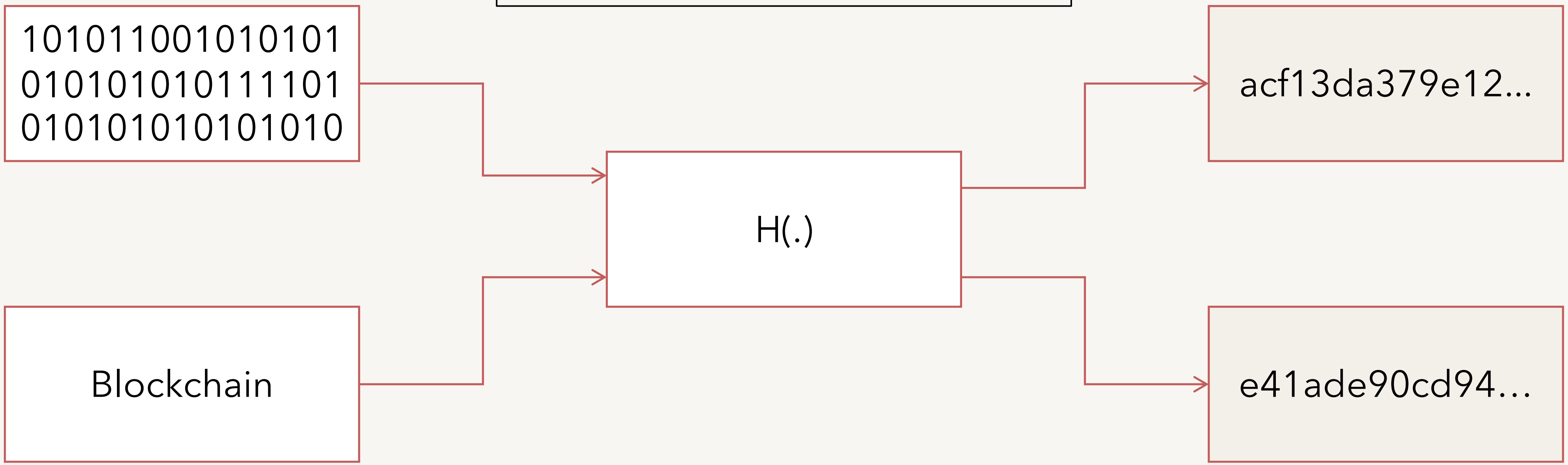
- Textbook
- https://github.com/PratyushRT/blockchainsS21/wiki
- https://github.com/sebischair/bbse

Cryptographic hash function

- A hash function is a mathematical function with the following three properties
  - Its input can be any string of any size
  - It produces a fixed size output
  - It is efficiently computable
    - computing the hash of an n-bit string should have a running time of $O(n)$

Cryptographic hash function

Compression: H: $\{0,1\}^* \rightarrow \{0,1\}^n$



Cryptographic hash function

- A cryptographic hash function is a general hash function that should satisfy these properties
  - collision-resistance
  - preimage resistance
  - hiding
  - puzzle-friendliness

Cryptographic hash function

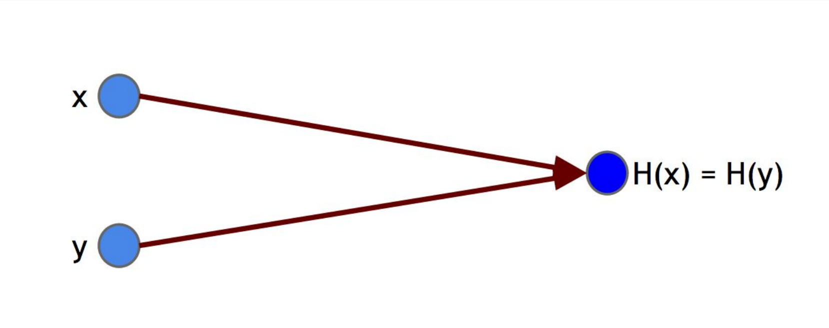
- A cryptographic hash function is a general hash function that should satisfy these three properties
  - collision-resistance
  - preimage resistance
  - hiding
  - puzzle-friendliness

Must-have

Desirable for certain blockchain systems

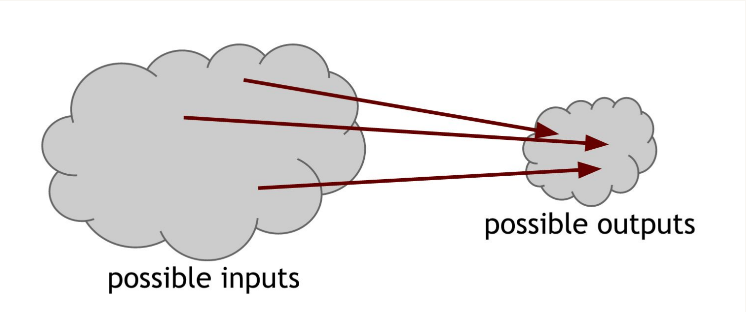
Cryptographic hash function: collision resistance

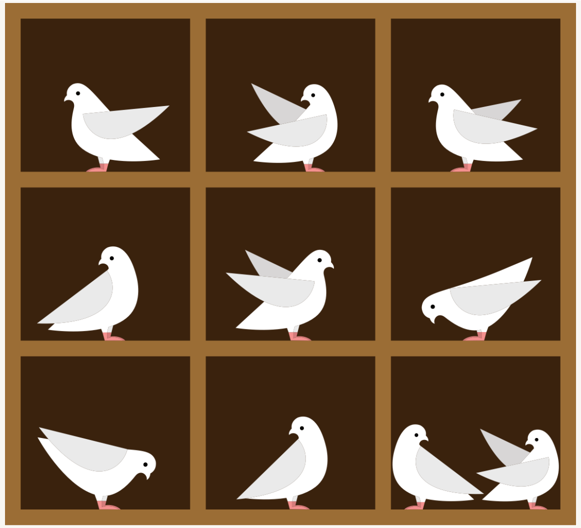
- A hash function is said to be collision resistant if it is infeasible to find two values, x and y, such that $x \neq y$, yet $H(x)=H(y)$



Cryptographic hash function: collision resistance

- A hash function is said to be collision resistant if it is infeasible to find two values, x and y, such that $x \neq y$, yet $H(x)=H(y)$
- Infeasible-> hard to find a collision, but not, no collisions exist
- The input space is > the output space (the input space is infinite, while the output space is finite)
  - there must be input strings that map to the same output string (the pigeonhole principle)
- But it will be hard to find these





https://miro.medium.com/max/1178/1*rj_4nlE4Qw_ok0P7A7chHA.png

Cryptographic hash function: collision resistance

- How to find a collision?
- Choose $2^{256} + 1$ distinct Input for a hash function with 256 bit output
- Calculate hash for each input and check if the output matches with any previous hash
- Since input sine > output size, there must be a match (collision)
- Try 2^{130} randomly chosen inputs, 99.8% chance that two of them will collide
    - Examining roughly the square root of the number of possible outputs (the birthday paradox)
    - The birthday paradox is that, counterintuitively, the probability of a shared birthday exceeds 50% in a group of only 23 people

Cryptographic hash function: collision resistance

- Is finding collision computationally feasible?
- A 256-bit hash function
  - worst case: $2^{256}+ 1$ times
  - best case: 2^{128} times on average
- If a computer calculates 10,000 hashes per second, 10^{27} years to generate 2^{128} hashes!

Cryptographic hash function: collision resistance

- The previous way was a brute-force method
- Is there any other optimised method available for finding collisions?
- Yes, for some hash functions: $H(x) = x \bmod 2^{256}$
  - Generates a 256 bit output and easily computable
  - But returns the last 256 bits of the input. One collision: 3 and $3 + 2^{256}$
- For others (e.g. SHA-256), we don't know yet

Cryptographic hash function: collision resistance

- Real-world adversaries
  - In practice, everyone has bounded resources
  - Therefore, reasonable to model a real-world adversary as such an entity
  - However, we do not make any assumptions about the adversarial strategy
  - He can use their (bounded) resources in any possible way
- Cryptographic adversary: A probabilistic polynomial-time (PPT) algorithm
  - an algorithm that runs in polynomial time (running time grows as a polynomial function of the input size) and may use (true) randomness to produce (possibly) non-deterministic results

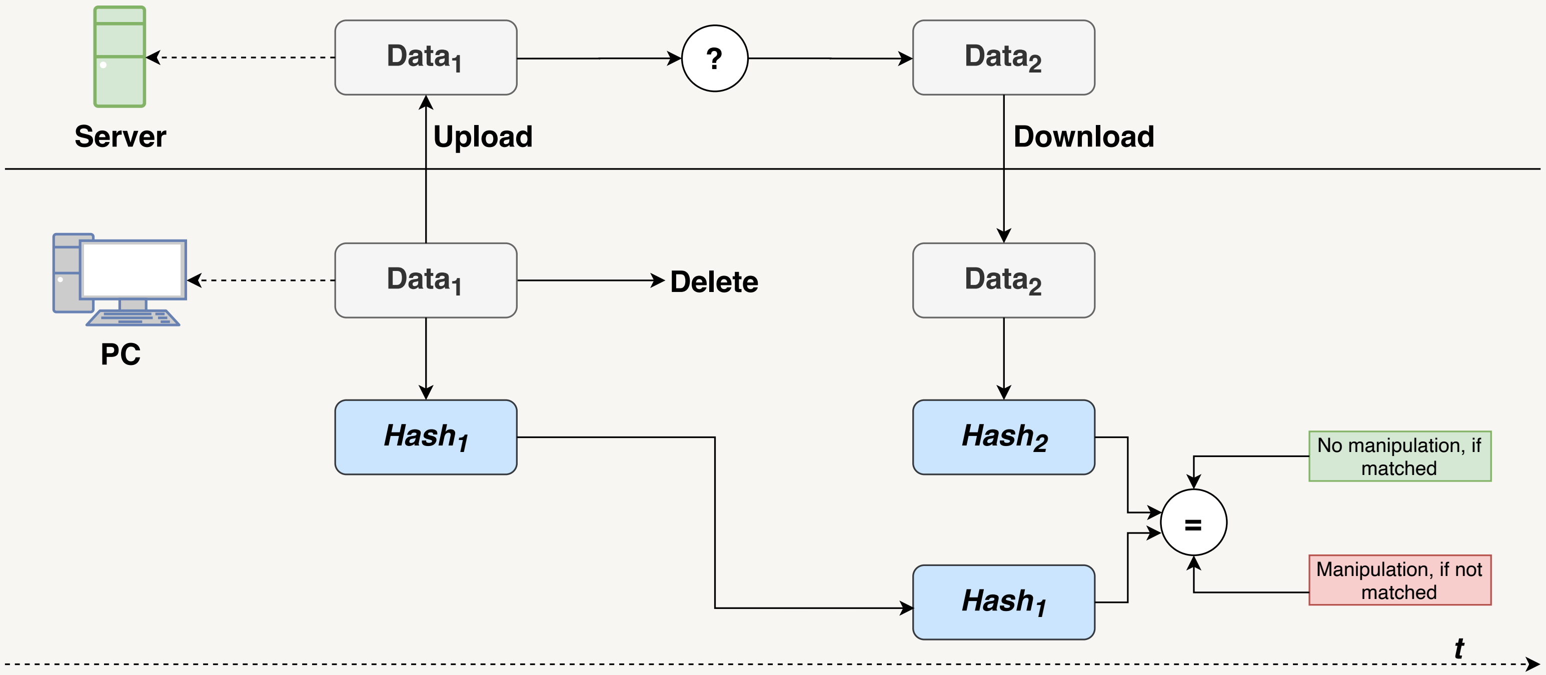
Cryptographic hash function: collision resistance

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

Pr[A outputs x,y s.t. x!=y and H(x)=H(y)] = "very small"

Collision resistance: application

- Message digest: a hash of any input, e.g. bits, random strings, characters or even files



Cryptographic hash function: pre-image resistance

- H is a hash function
- For essentially all pre-specified outputs y, it is computationally infeasible to find an x such that H(x) = y
- H is also called a one-way function

acf13da379e12... → $H^{-1}(.)$ → ???

Cryptographic hash function: pre-image resistance

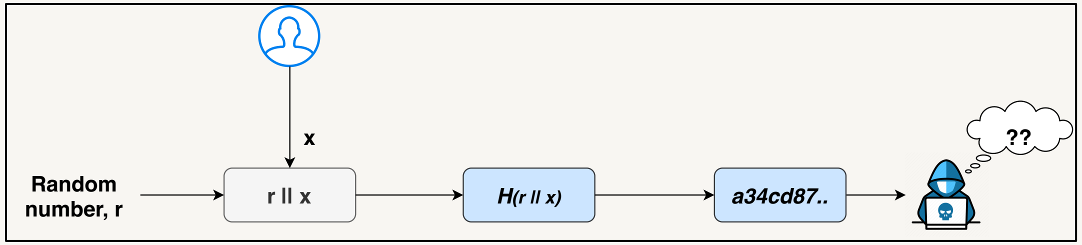
- If x is drawn from a uniform distribution with a large number of elements, then inverting H(x) is hard
- But what if x is drawn from a low min-entropy distribution?
  - In information-theory, min-entropy is a measure of how predictable an outcome is
  - High min-entropy captures the intuitive idea that the distribution (i.e., random variable) is very spread out

Cryptographic hash function: pre-image resistance

- But what if x is drawn from a low min-entropy distribution?
- Let the sample space is $X = \{h, t\}$
  - $H(x) = y$
- Can an attacker find the value of x given y?

Cryptographic hash function: hiding

- A desirable property for a cryptographic hash function is hiding which also tackles x picking up from a low min-entropy distribution
- A hash function H is hiding if
  - when a secret value r is chosen from a probability distribution that has high min-entropy,
  - then given H(r ǁ x) it is infeasible to find x



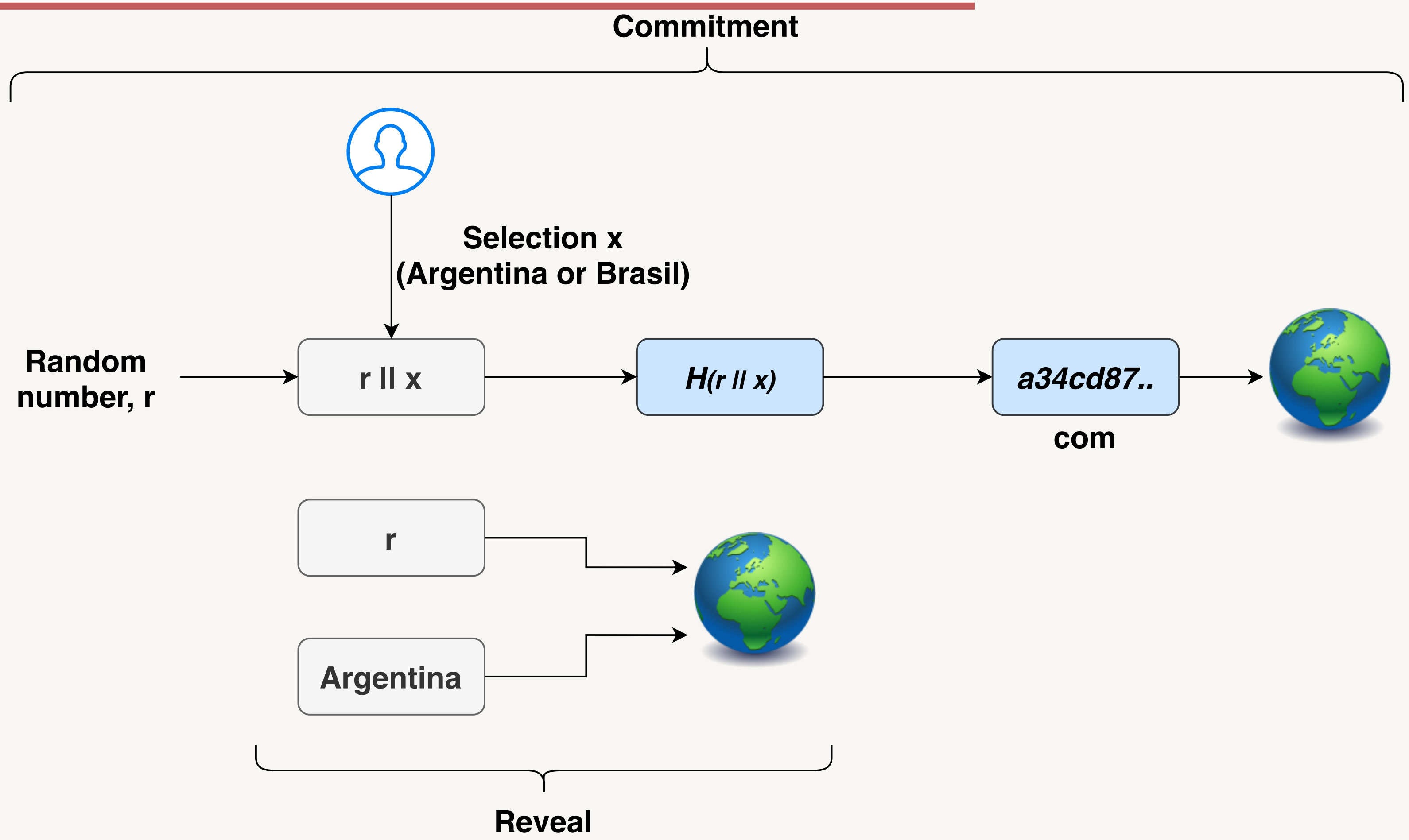
Hiding: application

- Commitment scheme
- Want to “seal a value (who will win the world cup 22?) in an envelope”, and publish it
  - Commit to a value (Argentina ☺) -> this is commitment
- Reveal your commitment to anyone -> open the envelope and verify your commitment

Hiding: commitment scheme

- com := commit(msg, key)
  - msg is the message and key is the random number used once
  - commit is essentially a hash function operating over the concatenation of msg and key
- verification := verify(com, msg, key)
  - Checks and returns whether msg and key produce the same result as com
- Security properties:
  - Hiding: Given com, no PPT adversary can find msg
  - Binding: No PPT adversary can find (msg, key) != (msg',key') such that verify(commit(msg, key), key',msg') == true

Hiding: commitment scheme




ANY QUESTION?