Do not distribute …

- These slides are not always prepared by me.
- Most of the content comes from the reference book
  - Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction
  - Arvind Narayanan, Joseph Bonneau,
  - Edward Felten, Andrew Miller, Steven Goldfeder

Review

- What is the difference between Bitcoin and Blockchain?
- What is a distributed ledger, consensus and mining?
- Same puzzle

Cryptographic Hash Functions

Cryptographic Hash Functions

Hash function:

- takes any string as input
- fixed-size output (we'll use 256 bits)
- efficiently computable

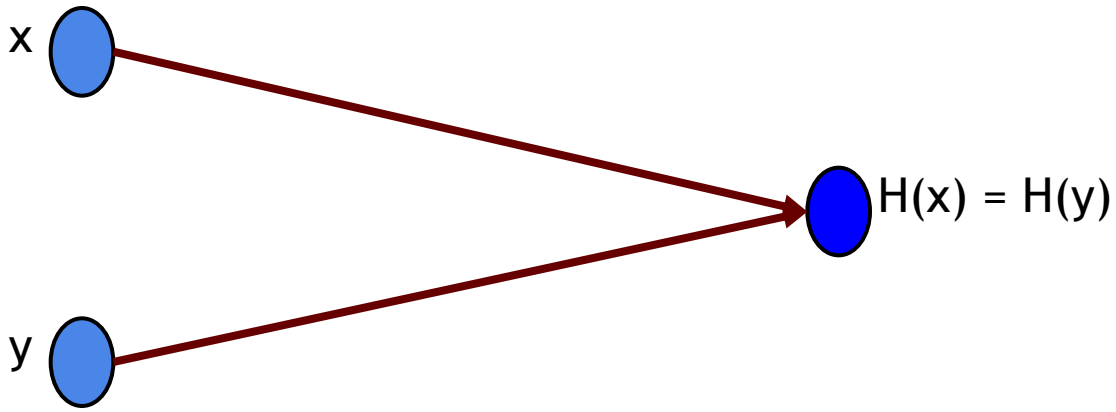
Security properties:

- collision-free
- hiding
- puzzle-friendly

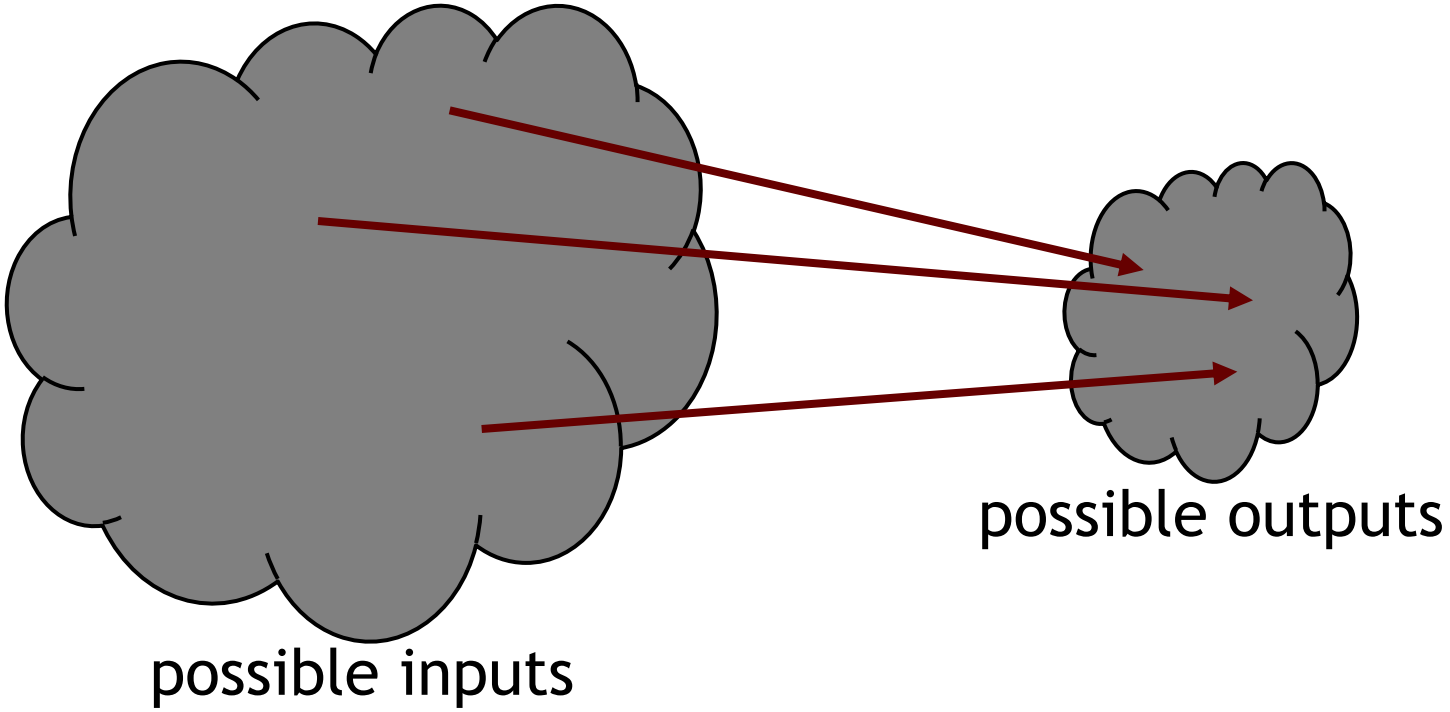
Hash property 1: Collision-free

Nobody can find x and y such that

x != y and H(x)=H(y)



Collisions do exist ...



… but can anyone find them?

How to find a collision

try 2^{130} randomly chosen inputs

99.8% chance that two of them will collide

if a computer calculates 10,000 hashes per second, it would take more than one octillion (10^{27}) years to calculate 2^{128} hashes!

This works no matter what H is …

… but it takes too long to matter

Is there a faster way to find collisions?

For some possible H's, yes.

For others, we don't know of one.

No H has been proven collision-free.

Application: Hash as message digest

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

it's safe to assume that x = y.

- Suppose that Bob uploads really large file and wants to be able to verify later that the file he downloads is the same as the one he uploaded.
- Compare (hash created at time of upload, hash obtained after the download) (fix length)

Hash property 2: Hiding

- The hiding property asserts that if we're given the output of the hash function $y = H(x)$, there's no feasible way to figure out what the input, x, was.

We want something like this:

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

x= apple

H(x)= 3a7bd3e2360a3d29eea436fcfb7e44c735d117c42d1c1835420b6b9942dd4f1b

Hash property 2: Hiding

- This will not work when x has to be chosen from a set that's, in some sense, NOT spread out. For example, flip a coin gives two possible values: head or tail.
- In response, compute both the hash of the string "heads" and the hash of the string "tails", and they could see which one they were given as an input

H("heads")

H("tails")

easy to find x!

Hash property 2: Hiding

Can we achieve the hiding property when the values that we want do not come from a spread-out set?

Answer is YES

we can hide an input that's not spread out by concatenating it with another input that is spread out.

H(r | x) (the vertical bar | denotes concatenation)

Hash property 2: Hiding

So, for a concrete example, if **r** is chosen uniformly from among all of the strings that are 256 bits long, then any particular string was chosen with probability $1/2^{256}$, which is an infinitesimally small value

Hash property 2: Hiding

Hiding property:

If r is chosen from a probability distribution that has *high min-entropy*, then given H(r | x), it is infeasible to find x.

min-entropy is a measure of how predictable an outcome is, and high min-entropy captures the intuitive idea that the distribution (i.e., random variable) is very spread out.

Hash property : Hiding use case

Common use of hiding property of hash is to secure passwords

- Instead of storing the password in the system, a more secure approach is just storing the hash of the password and compare the hash to verify the user.
- this way, the user password won't be at risk even the system is broken by attackers

Problem!

- many people tend to use simple words as their passwords

- Compare the hash password with a of commonly used passwords (hashed)

Hash property : Hiding use case

Solution!

- Use a randomly generated ‘salt’ to safeguard the password.
- *hash(password|salt)=output*

By appending a salt to the password, attackers can no longer use a pre-calculated password-hash map to attack the system. Even two users happened to choose the same string as their passwords, the hashes stored in the system are different because their salts are different, which is randomly generated.

Username	Salt value	String to be hashed	Hashed value = SHA256 (Salt value + Password)
user1	E1F53135E559C253	password123E1F53135E559C253	72AE25495A7981C40622D49F9A52E4F1565C90F048F59027BD9C8C8900D5C3D8
user2	84B03D034B409D4E	password12384B03D034B409D4E	B4B6603ABC670967E99C7E7F1389E40CD16E78AD38EB1468EC2AA1E62B8BED3A

Application: Commitment

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

Commit to a value, reveal it later.

Hash property 3: Puzzle-friendly

For every possible output value y,

if k is chosen from a distribution with high min-entropy,

then it is infeasible to find x such that H(k | x) = y.

Puzzle-friendly property implies that no solving strategy is much better than trying random values of *x*.

Hash-Functions

- Some of the most commonly used:
  - Secure Hashing Algorithm (SHA-2 and SHA-3)
  - Message Digest Algorithm 5 (MD5)
  - Keccak – used by Ethereum

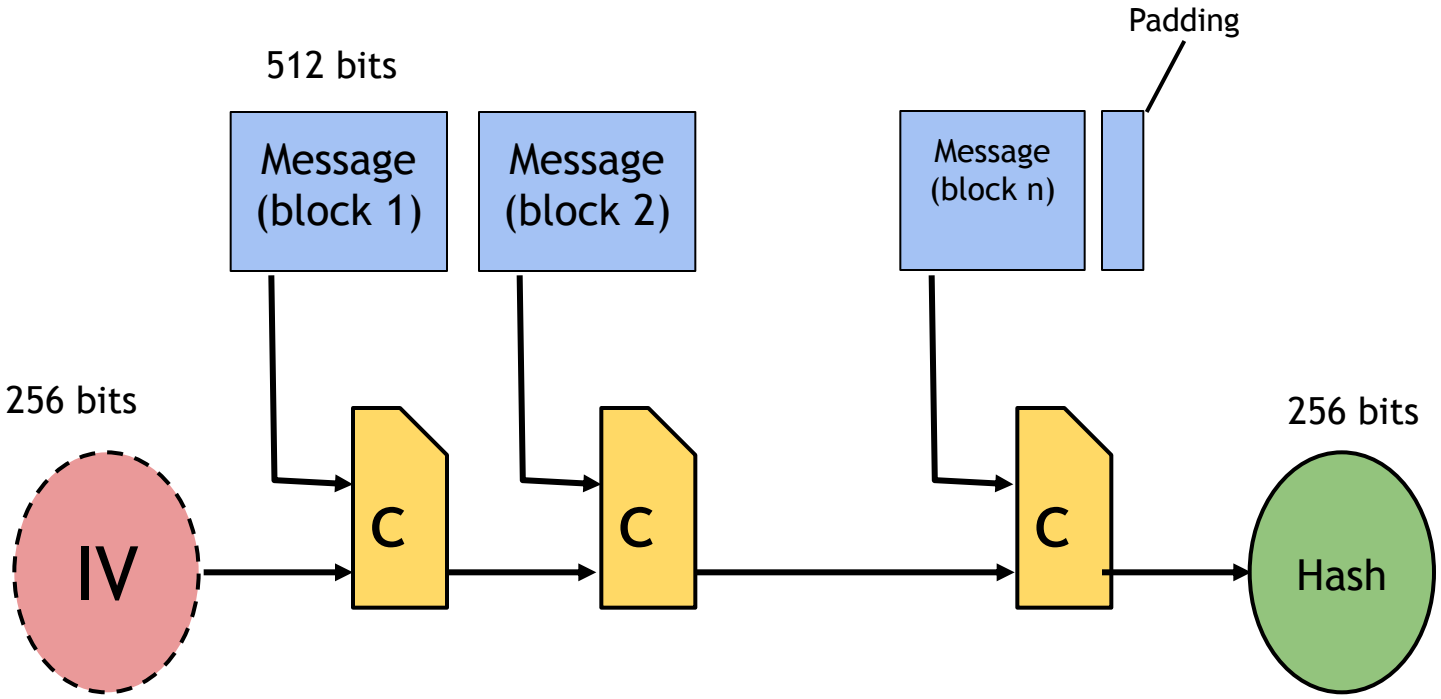
- SHA-2 is a family of hash functions that includes SHA-224, SHA-256, SHA-384, SHA-512, …

Problem?

- SHA-256 works on fixed length
- However, we require that our hash functions work on inputs of arbitrary length

- as long as we can build a hash function that works on fixed-length inputs, there's a generic method to convert it into a hash function that works on arbitrary-length inputs.
- It's called the Merkle-Damgard transform.

SHA-256 - Merkle-Damgard transform



Theorem: If c is collision-free, then SHA-256 is collision-free.

Do not distribute …

- These slides are not always prepared by me.
- Most of the content comes from the reference book
  - Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction
  - Arvind Narayanan, Joseph Bonneau,
  - Edward Felten, Andrew Miller, Steven Goldfeder