bitcoin

CS1699: Blockchain Technology and Cryptocurrency

3. Cryptographic Hashing Basics

Bill Laboon

What is a hash function?

- A function which accepts some arbitrary input x and returns a fixed-length “summary”
- Let’s look at “bad hash”

Bad Hash

- Converts all values in a string to their ASCII values
- Sums up all of these values
- Returns value modulo 256 (results 0x0 - 0xFF)
- Note: this is hashing function, but not a good one for our purposes!
- See ./sample_code/hash/bad_hash.rb to run an example, e.g. *ruby bad_hash.rb "meow"*

Hash vs Cryptographic Hash

- Hash values can be used for a variety of purposes (e.g. hash maps, data distribution, nearest neighbor search etc.)
- These different kinds of hash functions will have different properties (e.g. continuous hash functions are great for nearest neighbor search, horrible for data distribution!)
- For our purposes, we are interested in using hash functions for *cryptographic hashes*

Cryptographic Hashes

- Should have the following properties:
  - collision-free
  - hiding
  - puzzle-friendly

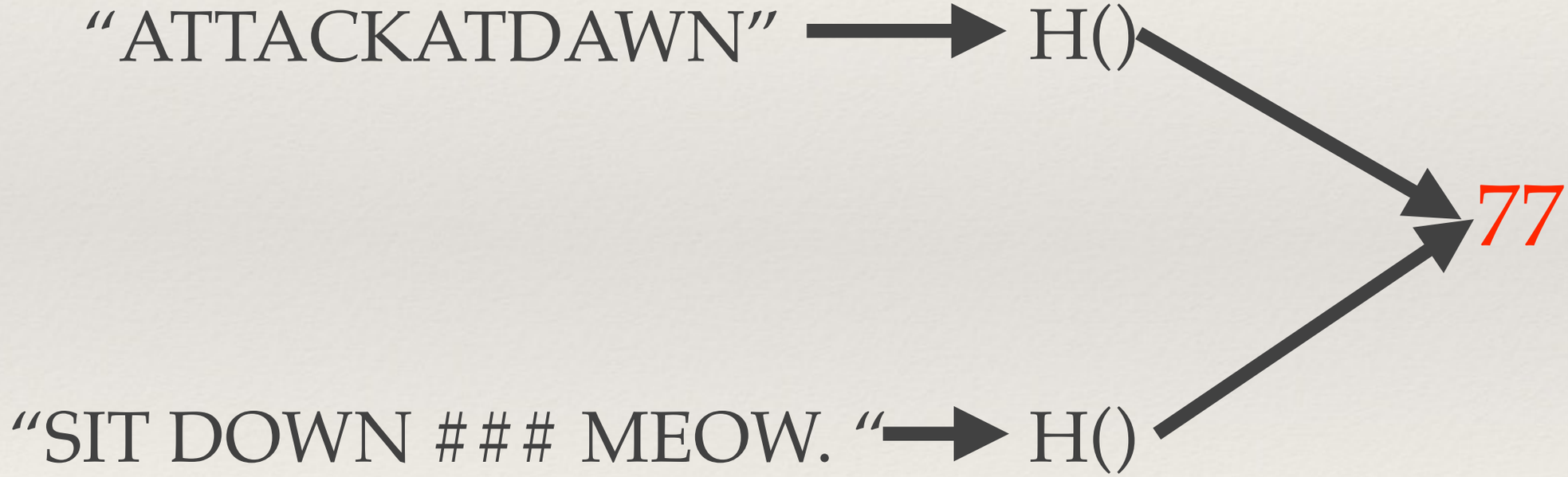
Collision-Free

- For hash function H(x), should be computationally infeasible to find a collision, that is, some y where :
  - x != y
  - H(x) == H(y)

Collision Example with Bad Hash

```
(23477) $ ruby bad_hash.rb "ATTACKATDAWN"
Hash of 'ATTACKATDAWN' is: 77

(23478) $ ruby bad_hash.rb "SIT DOWN ### MEOW. "
Hash of 'SIT DOWN ### MEOW. ' is: 77
```



Collisions Always Exist!

- A mapping from an infinite domain (x) to a finite codomain ($H(x)$) will have, in fact, infinite collisions
- So while collision-free is our goal, usually we say "collision-resistant"
- But can they be found in a computationally feasible way (i.e., a reasonable amount of time)?

Finding Collisions

- Brute force is always possible, but timescales for large hash functions are massive (millions of years using today's computers or even networks of computers)
- BUT - holes are always possible in any hash function H(x) - and have been exploited/broken in the past!
  - Broken = collisions can be found in a way significantly faster than brute force
  - SHA-1, GOST, MD2, MD4, MD5, Panama
- No hash function has been PROVEN secure

Can We Find A Collision?

- With bad hash, sure
- Easy to calculate result - can move up or down knowing ASCII values of characters in original message *m*
- Only 256 possible hash values, thus vulnerable to brute force

Using A Hash As a Message Digest

- Generally speaking, if H(x) == H(y), we can be reasonably certain that x == y
- So if we want to check, say, if we have seen something before, we can store its hash instead of the actual data
- Example: git (Side note: git uses the broken SHA-1 algorithm! See https://shattered.io/ and https://blog.github.com/2017-03-20-sha-1-collision-detection-on-github-com/)

Hiding

- Given the result of a hash function H(x), it is infeasible to "go backwards" to determine x
- That is, H(x) should be a one-way function

No hiding: given H(x), very easy to find x!

Hiding: A Formal Definition

- "A hash function H is said to be hiding when a secret value 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: a measure of how predictable an outcome is
- In other words, for any given input, if r is chosen uniformly, the result is likely to be any one of the possible output values

Bad Hash - Is it hiding?

- No, assuming low-ASCII (first 128 possibilities)
- Adding an additional character will strongly favor one half of the possibilities in our output!

Hiding

```
(23480) $ ruby bad_hash.rb "AAAA"
Hash of 'AAAA' is: 04

(23481) $ ruby bad_hash.rb "AAAA "
Hash of 'AAAA ' is: 24

(23482) $ ruby bad_hash.rb "AAAAZ"
Hash of 'AAAAZ' is: 5E

(23483) $ ruby bad_hash.rb "AAAAz"
Hash of 'AAAAz' is: 7E

(23484) $ ruby bad_hash.rb "AAAA~"
Hash of 'AAAA~' is: 82
```

Application of Hiding: Commitment

- Assume you want to wager on something happening, but you don't want others to know your guess
- e.g., "The price of one bitcoin will be over $10,000 in one year"

Commitment Scheme

com := commit(m, *nonce*)
verify(com, msg, *nonce*)

A nonce is random value that can be used only once (i.e., the *r* in our hiding definition). Nonces will come up very often in our exploration of Bitcoin!

Assumes the commit function is hiding and binding

> *Binding* = collision-resistant
>
> Given two pairs *m*/*nonce* and *m'*/*nonce'*, it is infeasible to find *m != m'* and *commit(m, nonce) == commit(m, nonce)*

Commitment Scheme

- To commit to something:
  - *(com, key) = commit(m)*
    - Publish com publicly
- To “open the envelope”
  - Publish *key* and *m*
  - Anyone can use *verify* function to verify *m*

Puzzle Friendliness

- "For every possible n-bit output value y, if k is chosen from a distribution with high min-entropy, then it is infeasible to find x such that $H(K \mid\mid x) == y$ in time significantly less than 2^{n}"

Puzzle Friendliness

- In other words, if someone wants the hash value result $H(x)$ to be a particular value y, if $H(x)$ produces a result of size n bits, then finding some value x such that $H(x) == y$ should require at least 2^n attempts ($O(2^n)$)
- In other other words, the best way to find the result of a hash function is to just do the hash with different inputs, and brute force it

Puzzle Friendliness Application

- Proof of work
- I want to prove that you have done some work before you get a prize
- So I may ask you to find a value x where H(x) gives me 0
- Best way to do it SHOULD be to just try every possible input

Is Bad Hash Puzzle Friendly?

- No
- Very easy to determine output and modify input to modify output

A Shortcut for Bad Hash

```
(23485) $ ruby bad_hash.rb "AAAA"
Hash of 'AAAA' is: 04

(23486) $ ruby bad_hash.rb "AAA<"
Hash of 'AAA<' is: FF

(23487) $ ruby bad_hash.rb "AAA?"
Hash of 'AAA?' is: 02

(23488) $ ruby bad_hash.rb "AAA="
Hash of 'AAA=' is: 00
```

Hash Functions Used in Bitcoin

- SHA-256: Created by NSA, 256-bit output
- RIPEMD-160: Developed by researchers at Katholieke Universiteit Leuven, 160-bit output

SHA 256

- SHA 256 is a "good" hash for our purposes
- As far as we know (it has not been proven!), it is:
  - Collision-resistant
  - Hiding
  - Puzzle-friendly

SHA-256 Shortcut?

```
(23489) $ ruby sha256_hash.rb "AAAA"
Hash of 'AAAA' is:
63c1dd951ffedf6f7fd968ad4efa39b8ed584f162f46e715114ee184f8de9201

(23490) $ ruby sha256_hash.rb "AAA<"
Hash of 'AAA<' is:
ad587d6af700f2625287ce9b0c66120a120a0b56fd3c8049cc9d0596832d60a6

(23491) $ ruby sha256_hash.rb "AAAB"
Hash of 'AAAB' is:
5e01d7a18db038714fcbb6de5708f3f3c36ef61c206c72ce5dd91c149ecc910c

(23492) $ ruby sha256_hash.rb "AAA="
Hash of 'AAA=' is:
e2d4768b3472b90ca749600da34e622122d16f61517c459a5d7915693ec4ccf3
```

SHA-256 in Bitcoin

- Mining: SHA-256 hashing
- Internal integrity checks
- Hash pointers for Merkle tree
- SHA-256 hashing of ECDSA as part of key generation