Crypto Background II

Blockchains & Cryptocurrencies

Course logistics

- Assignment 1 is due 9/24 (11:59pm ET)

News?

Hash power broker Nicehash denies that it enables bad actors to use its hash renting platform to launch 51% attacks on blockchain networks. The broker insists that it does not have any way of monitoring or determining which blockchain is benefitting from a particular algorithm hash data. Only buyers of hashrate know this, as well as pools that receive such hashpower.

The comments by Nicehash are in response to pointed allegations from Ethereum Classic (ETC) Labs developers stating that the hash power used to initiate the 51% attacks on their network was purchased from the broker. To further support their allegations, the ETC devs claim an unnamed Nicehash cofounder has already been convicted in Slovenia on similar offenses.

Last time

- We talked about hash functions, pre-image resistant and collision-resistance
- We didn't much talk about why we cared about this
- Today: finishing up some crypto background, and hopefully on to consensus and Bitcoin (soon!)

Hash Pointers and Data Structures

Reminder: hash functions

- Take as input an arbitrary-length string
- Output a (shorter) fixed-size string

Cryptographic hash function security:

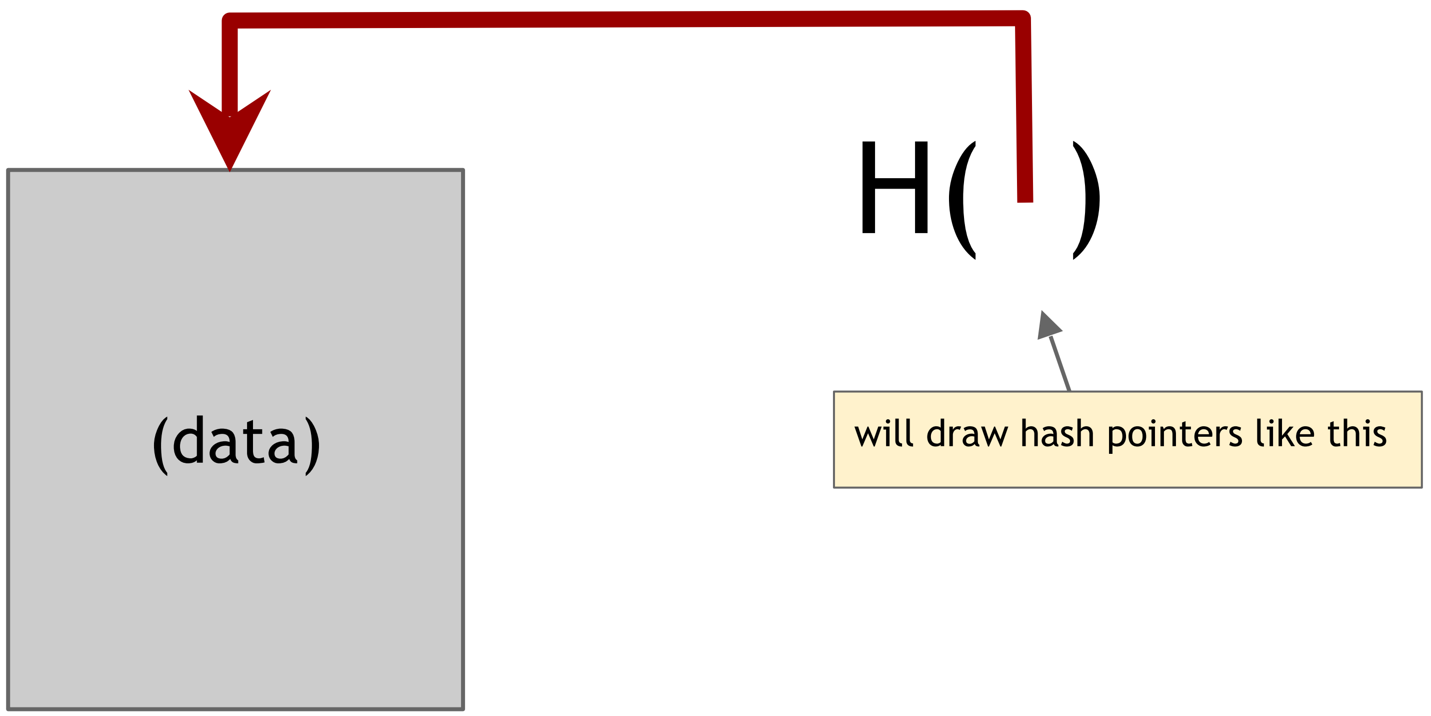
- Collision-resistant
- Pre-image resistant
- “Random oracle”-like (for some cases)

Hash pointer

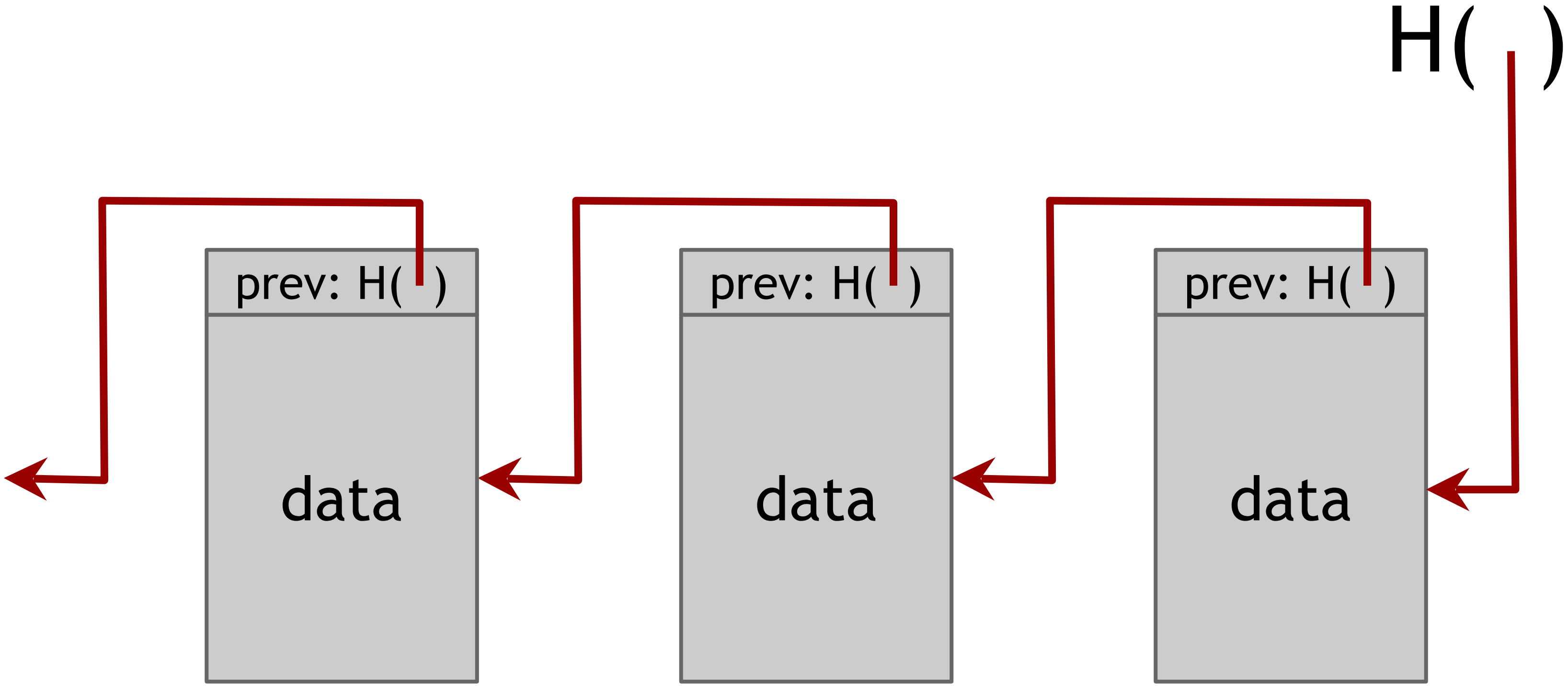
- pointer to where some info is stored, *and*
- cryptographic hash of the info

If we have a hash pointer, we can

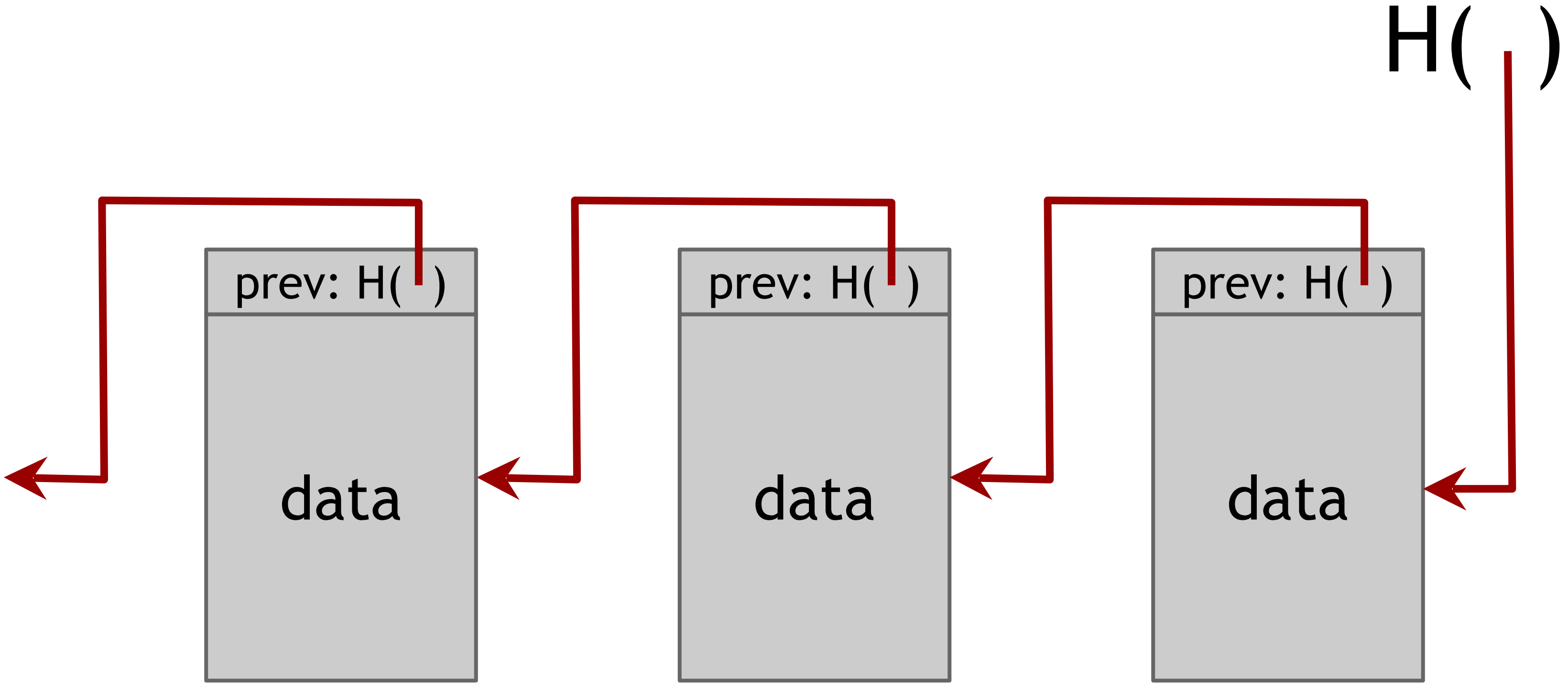
- ask to get the info back, and
- verify that it hasn't changed


(data)
H( )
will draw hash pointers like this

Building data structures with hash pointers

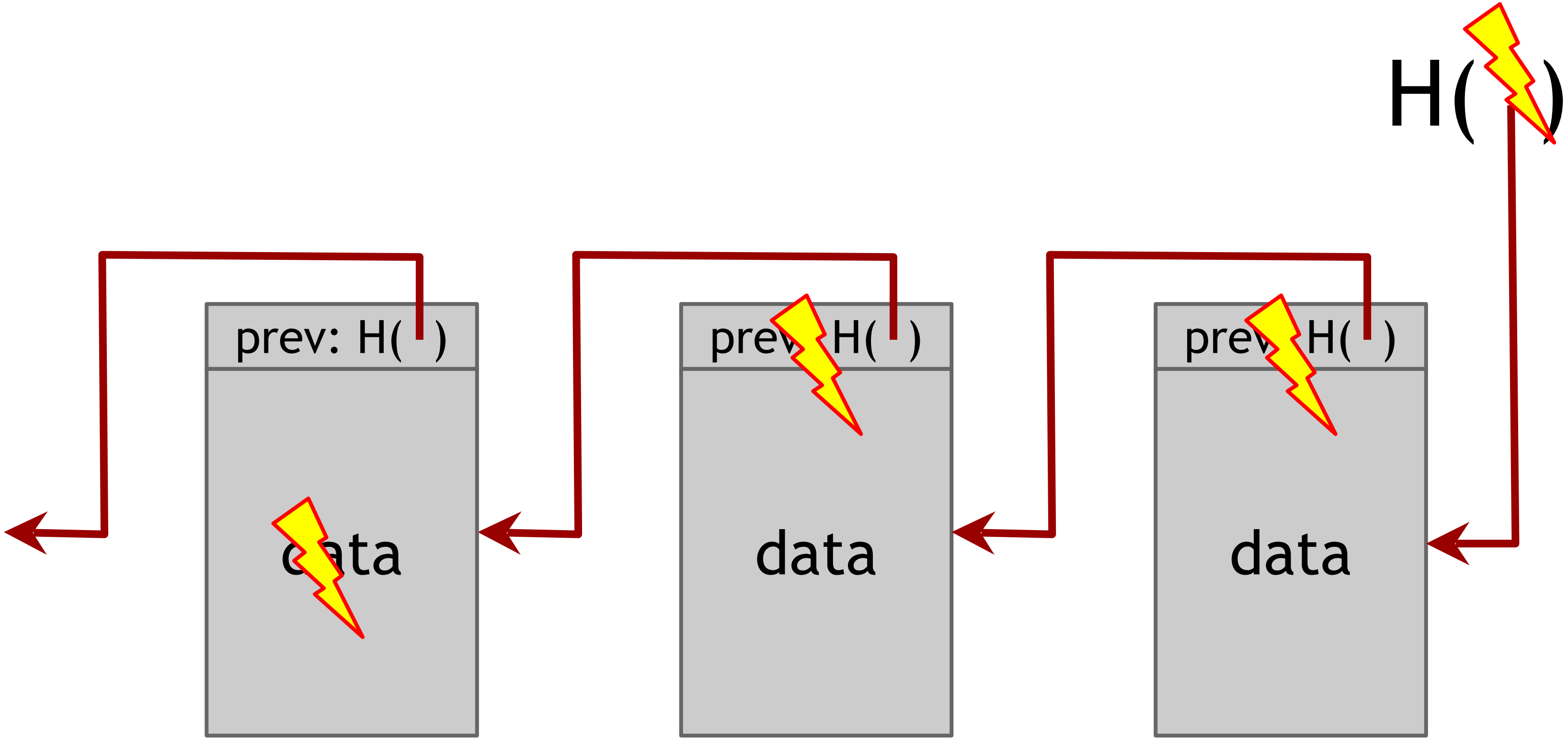

H( )
prev: H( )
data
prev: H( )
data
prev: H( )
data

Linked list with hash pointers = “Blockchain”



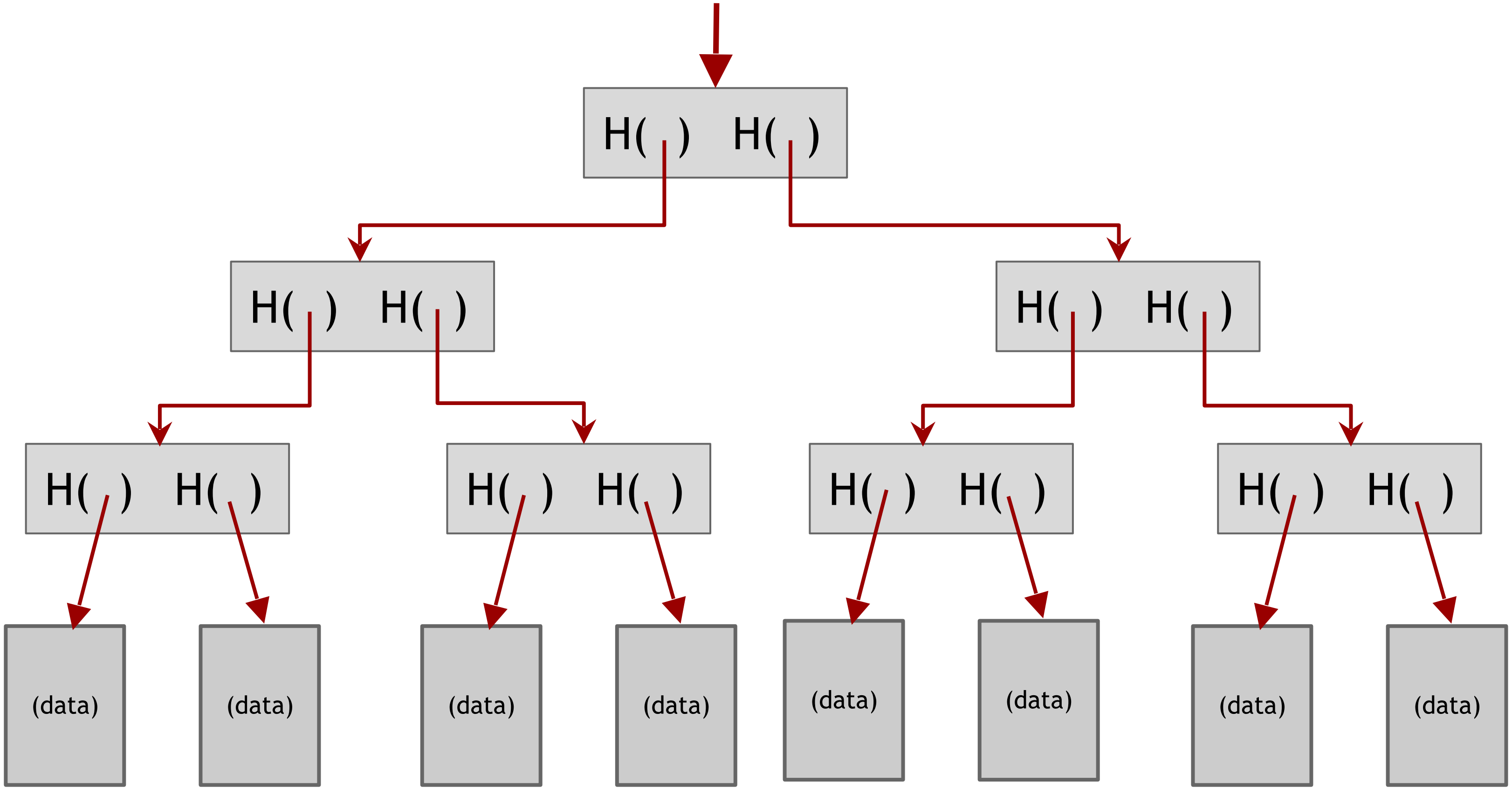
use case: tamper-evident log

detecting tampering



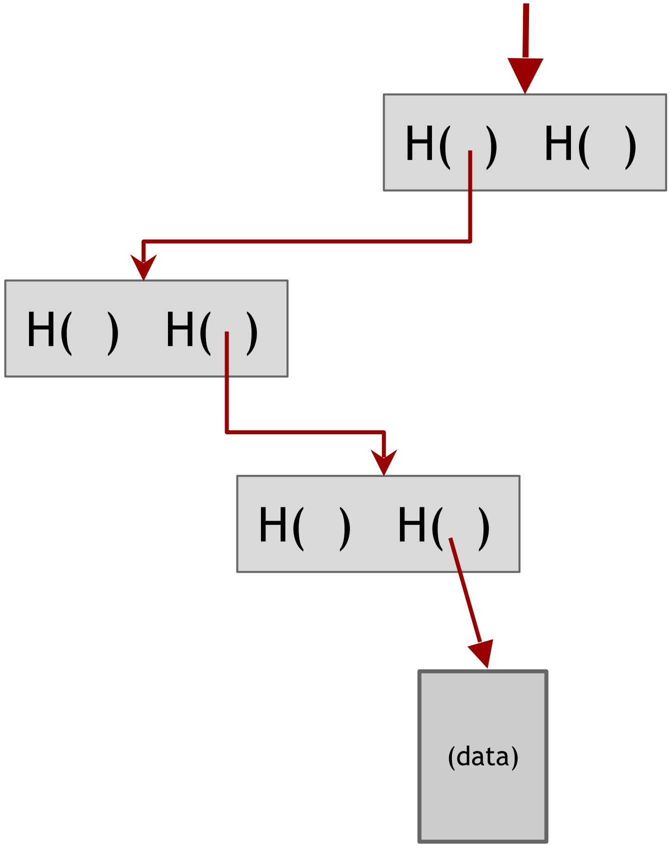
use case: tamper-evident log

binary tree with hash pointers = “Merkle tree”



proving membership in a Merkle tree

proving membership in a Merkle tree



show O(log n) items

Advantages of Merkle trees

- Tree holds many items, but just need to remember the root hash
- Can verify membership in O(log n) time/space

Variant: *sorted* Merkle tree

- can verify non-membership in O(log n)
- show items before, after the missing one

Before Bitcoin…

NOTICES &
LOST AND
FOUND
(5100-5102)

Universal Registry Entries:
Zone 2 -
dS8492cgVOFAoP9kyE1XzMOrQ
HgEwzkVbVafNylkUz99qvq8/ME
p5y9EFSG8XxzMBaIGQQ==
Zone 3 -
JnFCg+HCmvhj8GmmUP7VZna71
NgZup/RfuKUQNzCHWXMuqLK
durxHQV5pSHLqBGPRIy+mg==
These base64-encoded values represent the combined fingerprints of all digital records notarized by Surety between 2009-06-03Z 2009-06-09Z.
www.surety.com 571-748-5800

Before Bitcoin...

NOTICES & LOST AND FOUND

(5100-5102)

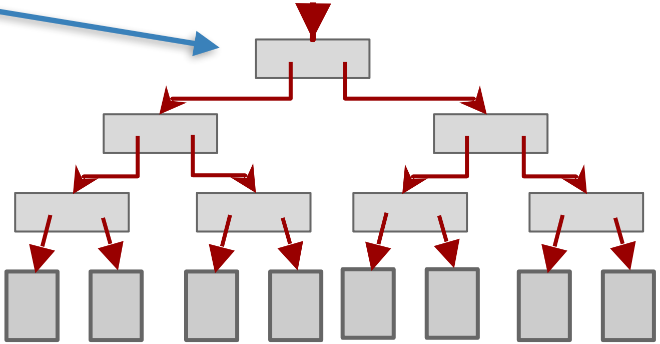
Universal Registry Entries:

Zone 2 -
dS8492cgVOFAoP9kyE1XzMOrQ
HgEwzkVbVafNylkUz99qva8/ME
p5y9EFSG8XxzMBaIGQQ==

Zone 3 -
JnFCg+HCmvhj8GmmUP7VZna71
NgZup/RfuKUQNzCHWXMuqLK
durxHQV5pSHLqBGPRIy+mg==

These base64-encoded values represent the combined fingerprints of all digital records notarized by Surety between 2009-06-03Z 2009-06-09Z.

www.surety.com 571-748-5800



More generally ...

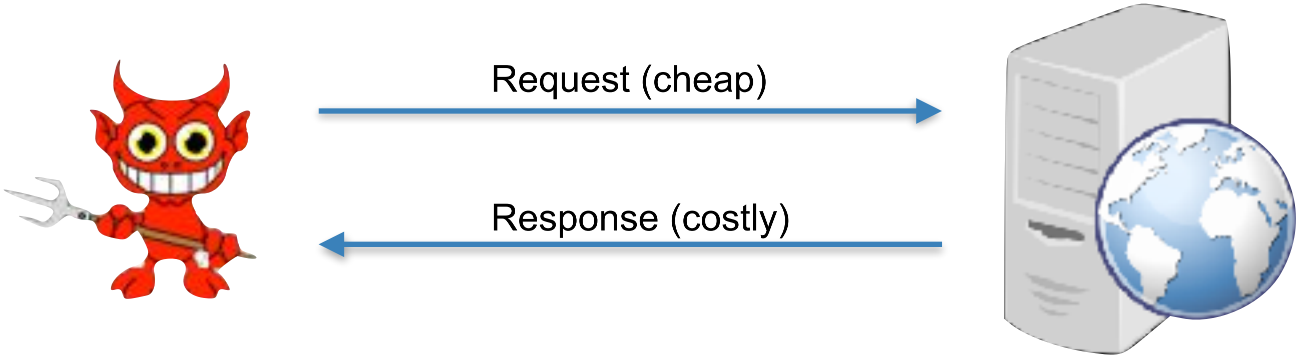
Can use hash pointers in any pointer-based data structure that has no cycles

Cryptographic puzzles

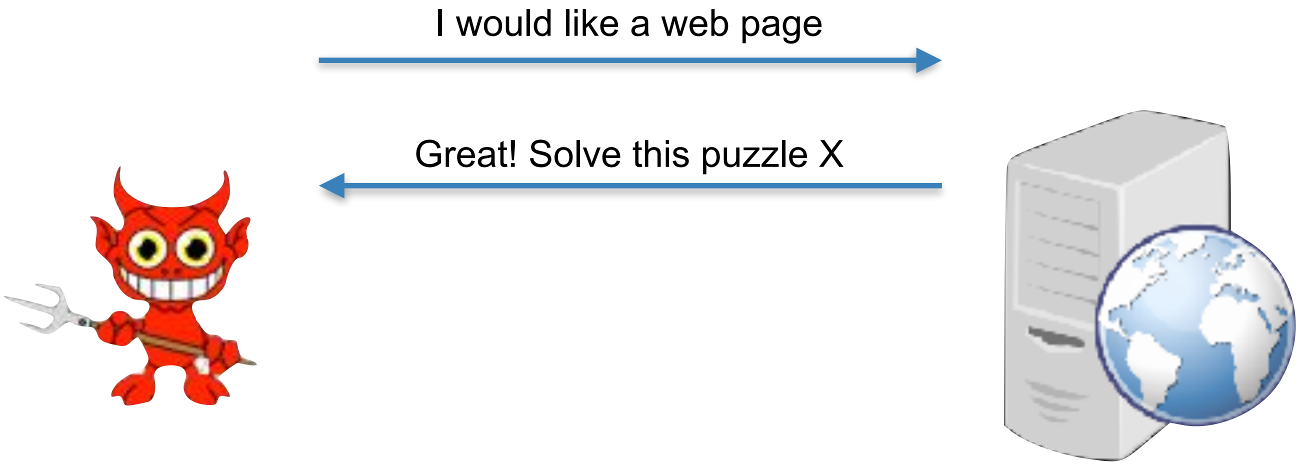
Problem setting

- People are trying to get resources from my server, and this is overloading my capacity
  - Denial of Service (DDoS) attacks
  - Email spam
- It’s “cheap” for someone to make a request, “expensive” for me to serve the response
- One attacker can pretend to be millions of clients (sybil attack)

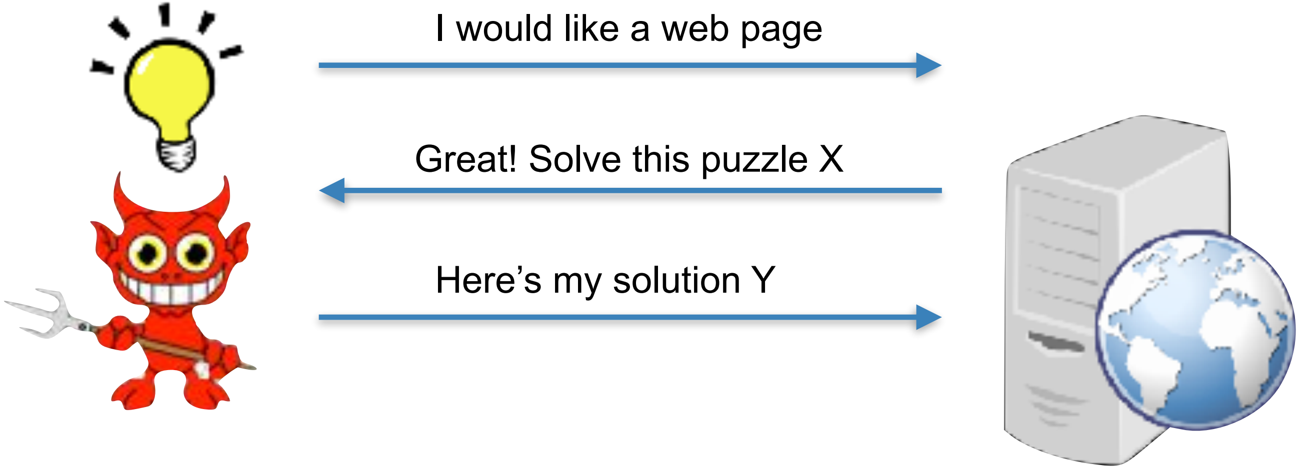
Initial picture



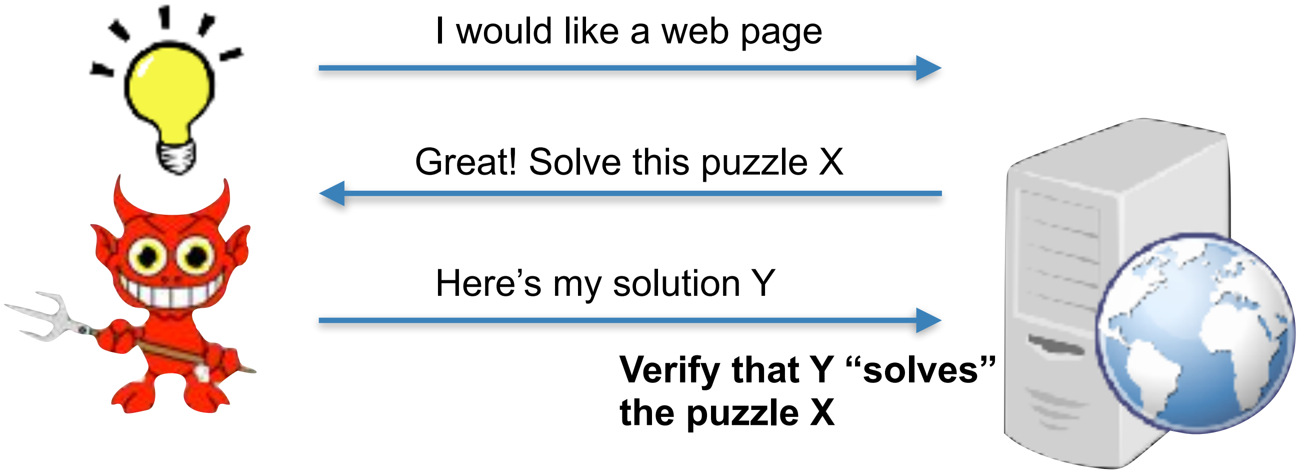
Dwork & Naor, Back: puzzles



Dwork & Naor, Back: puzzles



Dwork & Naor, Back: puzzles



Dwork & Naor, Back: puzzles

I would like a web page

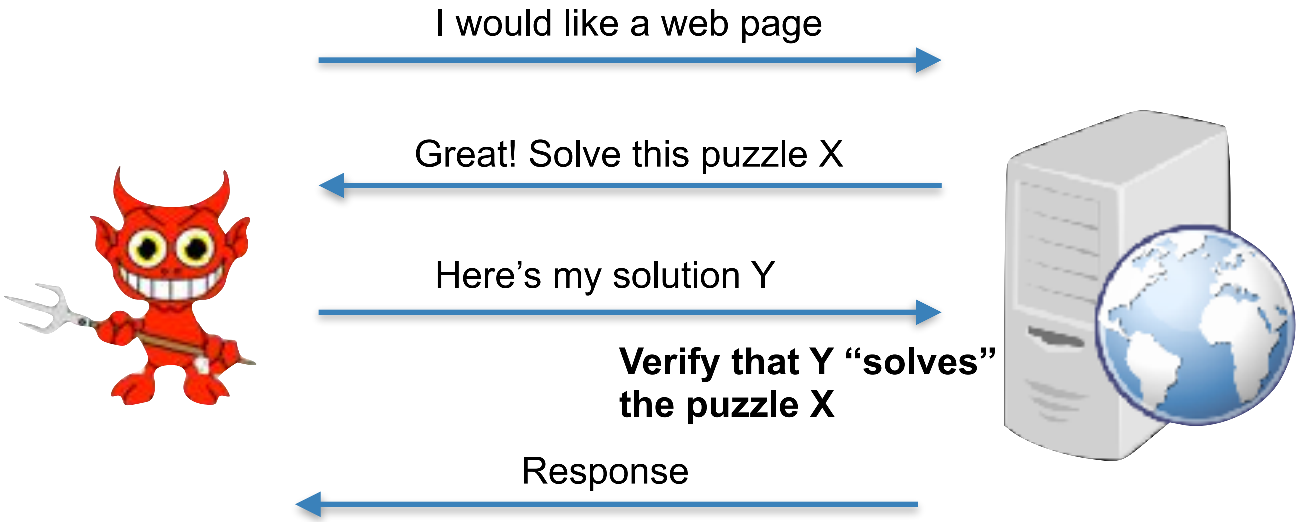
Great! Solve this puzzle X

Here's my solution Y

Verify that Y "solves" the puzzle X

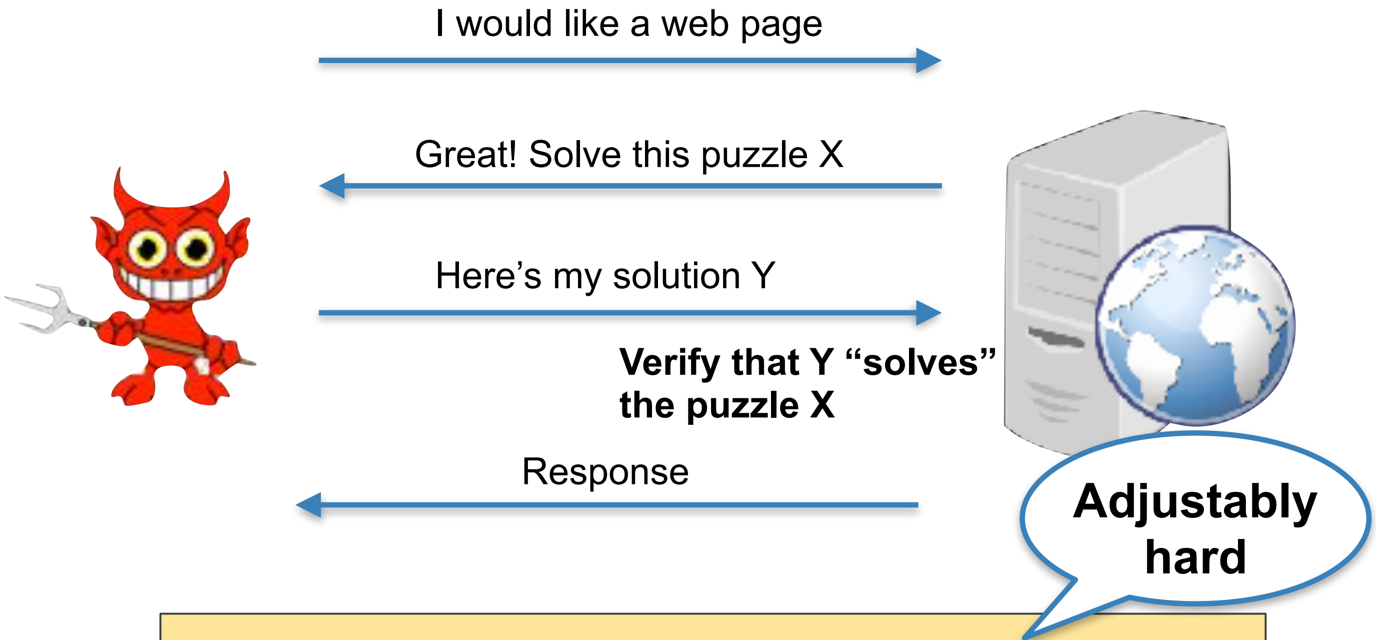
Response

Dwork & Naor, Back: puzzles



We need the puzzle to be "hard", and checking the solution to be "easy"

Dwork & Naor, Back: puzzles

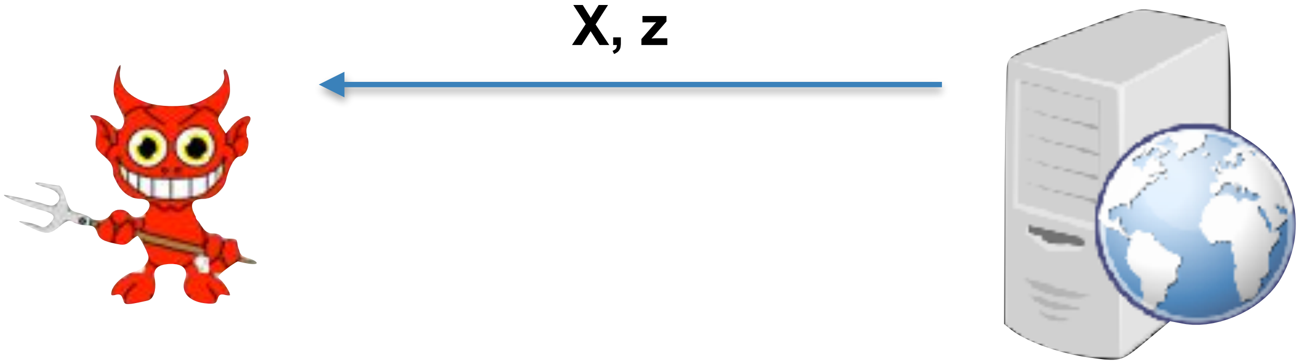


We need the puzzle to be "hard", and checking the solution to be "easy"

Example (hash) puzzle (PoW)

X is a random bit string (128 bits)

z is an integer between 0…k

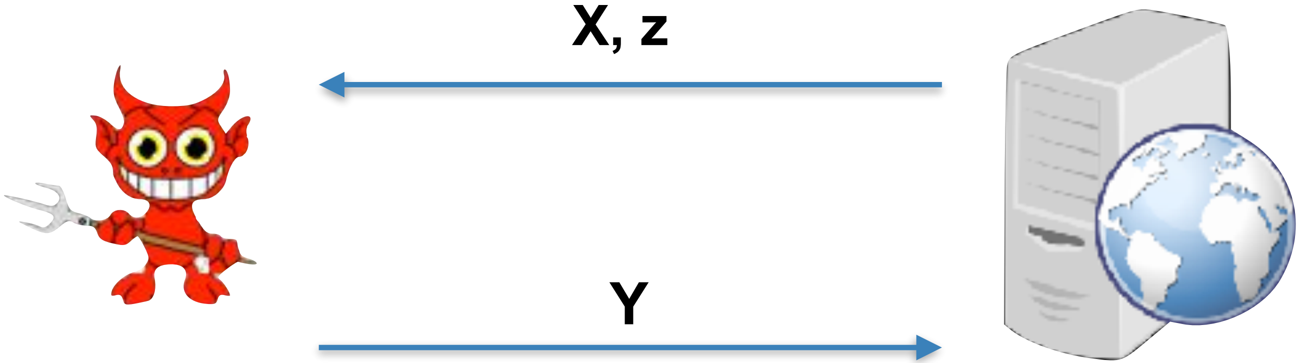


Assume H: $\{0,1\}^* \rightarrow \{0,1\}^k$

Example (hash) puzzle (PoW)

X is a random bit string (128 bits)

z is an integer between 0…k



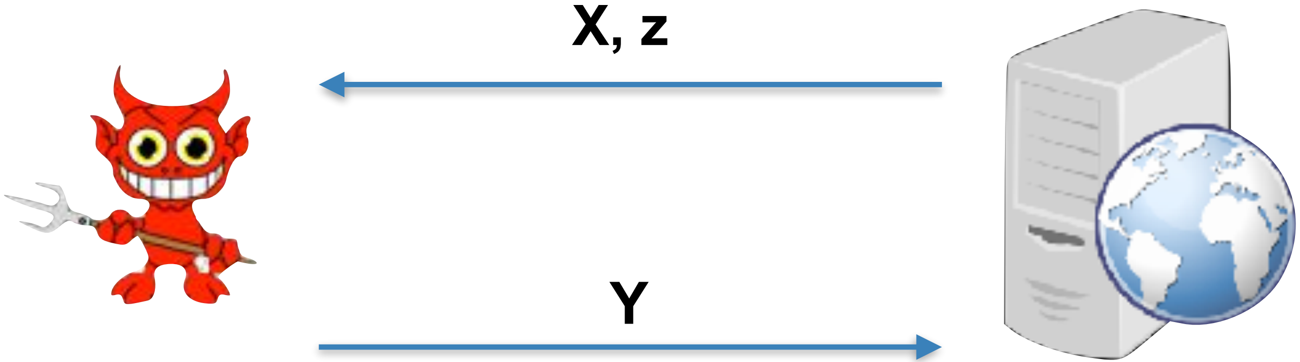
Verify that H(X | Y) = 0^z…

Assume H: $\{0,1\}^* \text{ -> } \{0,1\}^k$

Example (hash) puzzle (PoW)

X is a random bit string (128 bits)

z is an integer between 0…k



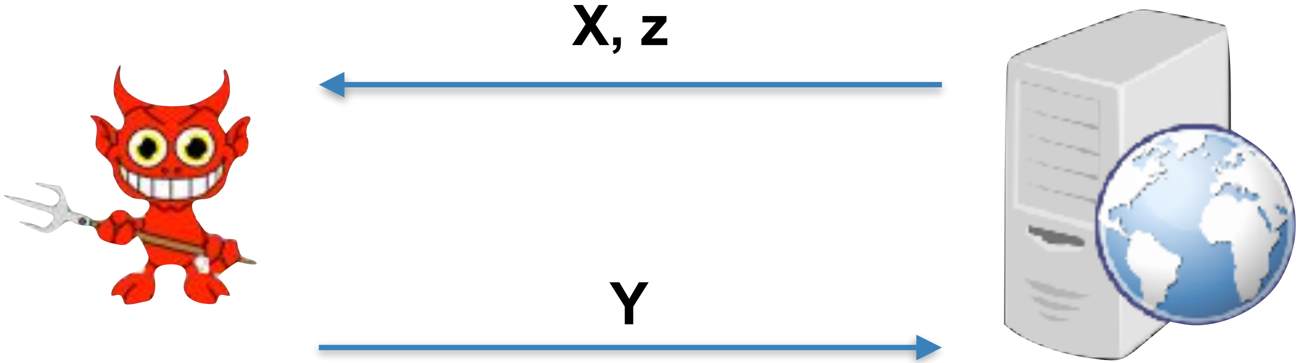
Verify that $H(X \mid Y) = 0^z$…

Q: How does the client find Y?

Example (hash) puzzle (PoW)

X is a random bit string (128 bits)

z is an integer between 0…k



For **Y** = 0 to infinity, do:
  if H(**X | Y**) leads with **z** "0" bits, return Y
  else loop

Verify that H(X | Y) = 0^z…

Q: How does the client find Y?
A: Brute force search

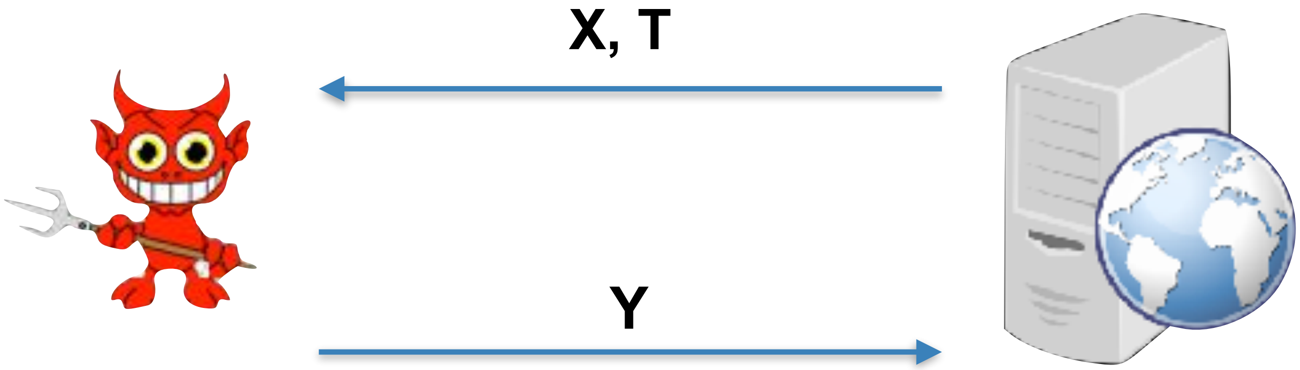
Example (hash) puzzle (PoW)

- Assuming that H is a random oracle:
  - Each hash is a Bernoulli trial
  - How many times does the client need to evaluate H() to find a solution for a given z?
  - What’s does the variance look like?

Hash puzzle variant (PoW)

X is a random bit string (128 bits)

T is a k-bit (unsigned) integer



Verify that uint[H(X | Y)] < T

This generalizes the previous puzzle, and allows fine-grained difficulty adjustment

Hash power broker Nicehash denies that it enables bad actors to use its hash renting platform to launch 51% attacks on blockchain networks. The broker insists that it does not have any way of monitoring or determining which blockchain is benefitting from a particular algorithm hash data. Only buyers of hashrate know this, as well as pools that receive such hashpower.

The comments by Nicehash are in response to pointed allegations from Ethereum Classic (ETC) Labs developers stating that the hash power used to initiate the 51% attacks on their network was purchased from the broker. To further support their allegations, the ETC devs claim an unnamed Nicehash cofounder has already been convicted in Slovenia on similar offenses.

Digital Signatures

What we want from signatures

- Only you can sign, but anyone can verify
- Signature is tied to a particular document

    (*can't be cut-and-pasted to another doc)*

- Even if one can see your signature on some documents, he cannot "forge" it

Digital signatures

Security parameter

- (sk, pk) ← keygen(1^k)

  sk: secret signing key

  pk: public verification key

randomized algorithm

- sig ← sign(sk, message)

Typically randomized

- isValid ← verify(pk, message, sig)

Requirements for signatures

- Correctness: “valid signatures verify”
  - verify(pk, message, sign(sk, message)) == true

- Unforgeability under chosen-message attacks (UF-CMA): “can’t forge signatures”
  - adversary who knows pk, and gets to see signatures on messages of his choice, can’t produce a verifiable signature on another message

UF-CMA Security

(sk, pk) ← keygen(1^k)

pk →

← m_0

sign(sk, m_0) →

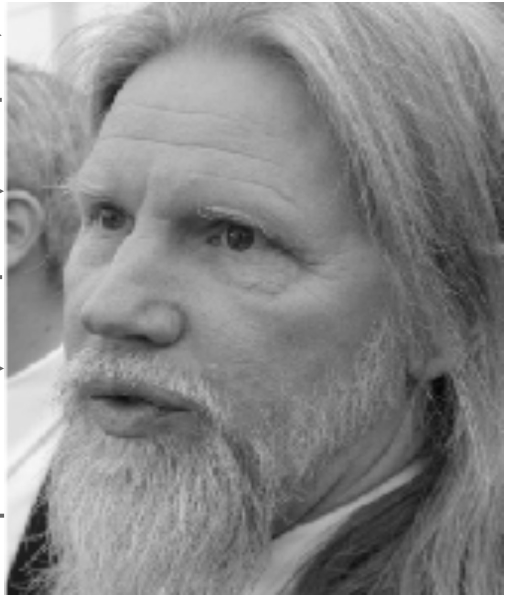
← m_1

sign(sk, m_1) →

• • •

← M, sig

M not in { m_0, m_1, … }



Challenger

verify(pk, M, sig)

ifValid, attacker wins

Adversary

Definition: A signature scheme (keygen,sign,verify) is UF-CMA secure if for every PPT adversary A, there exists a negligible function n(k) s.t. Pr[A wins in above game] = n(k)

Notes

- Signatures can be shorter than message: sign Hash(message) rather than message
- Algorithms are randomized: need good source of randomness. Bad randomness may reveal the secret key
- fun trick: sign a hash pointer. signature “covers” the whole structure

Notes…

- Bitcoin uses Elliptic Curve Digital Signature Algorithm (ECDSA)
- ECDSA is a close variant of Schnorr Signature scheme over Elliptic curves