Proof of Work

CS 5158/6058 Data Security and Privacy

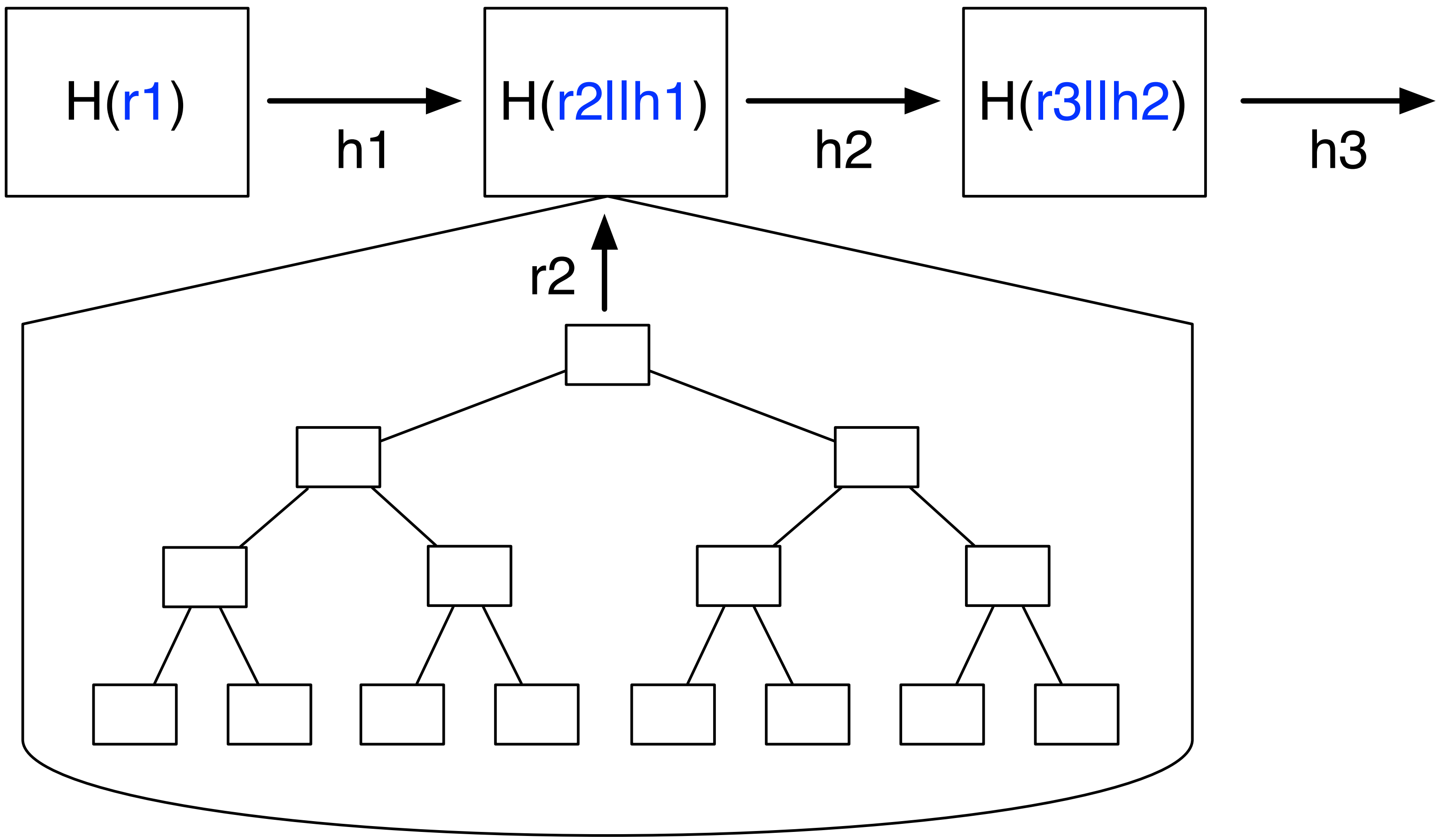
Spring 2018

Instructor: Boyang Wang

The Blockchain

- Bitcoin network has a large number of transactions.
  - About 196,000 trans per day
  - Bitcoin uses the blockchain to maintain integrity

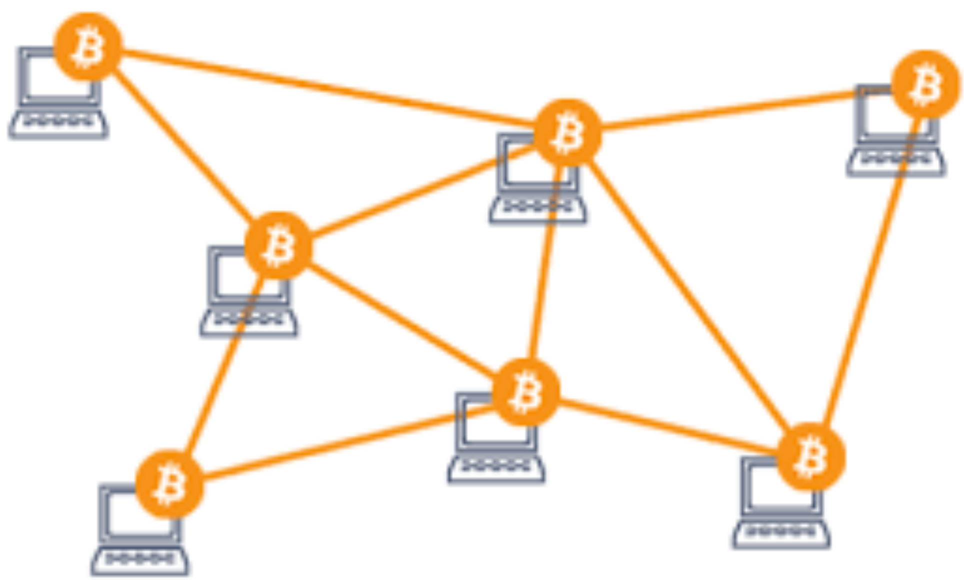
- Blockchain includes Hash chain + Merkle tree
  - Each block has 1MB limit (1000~2000 trans)
  - Trans in one block form one Merkle tree
  - All the root hash values form a hash chain
  - Total blocks: 512,169 (as of 3/5/2018)



- Example: Assume each block has 8 transactions
  - 3 blocks, 24 transactions in total
  - Hash value **h3** proves all the 24 transactions

Bitcoin Network

- Bitcoin network is a peer-to-peer network
  - No central authority
  - All nodes are equal
  - Forget non-responding nodes after 3 hrs



Join Bitcoin Network

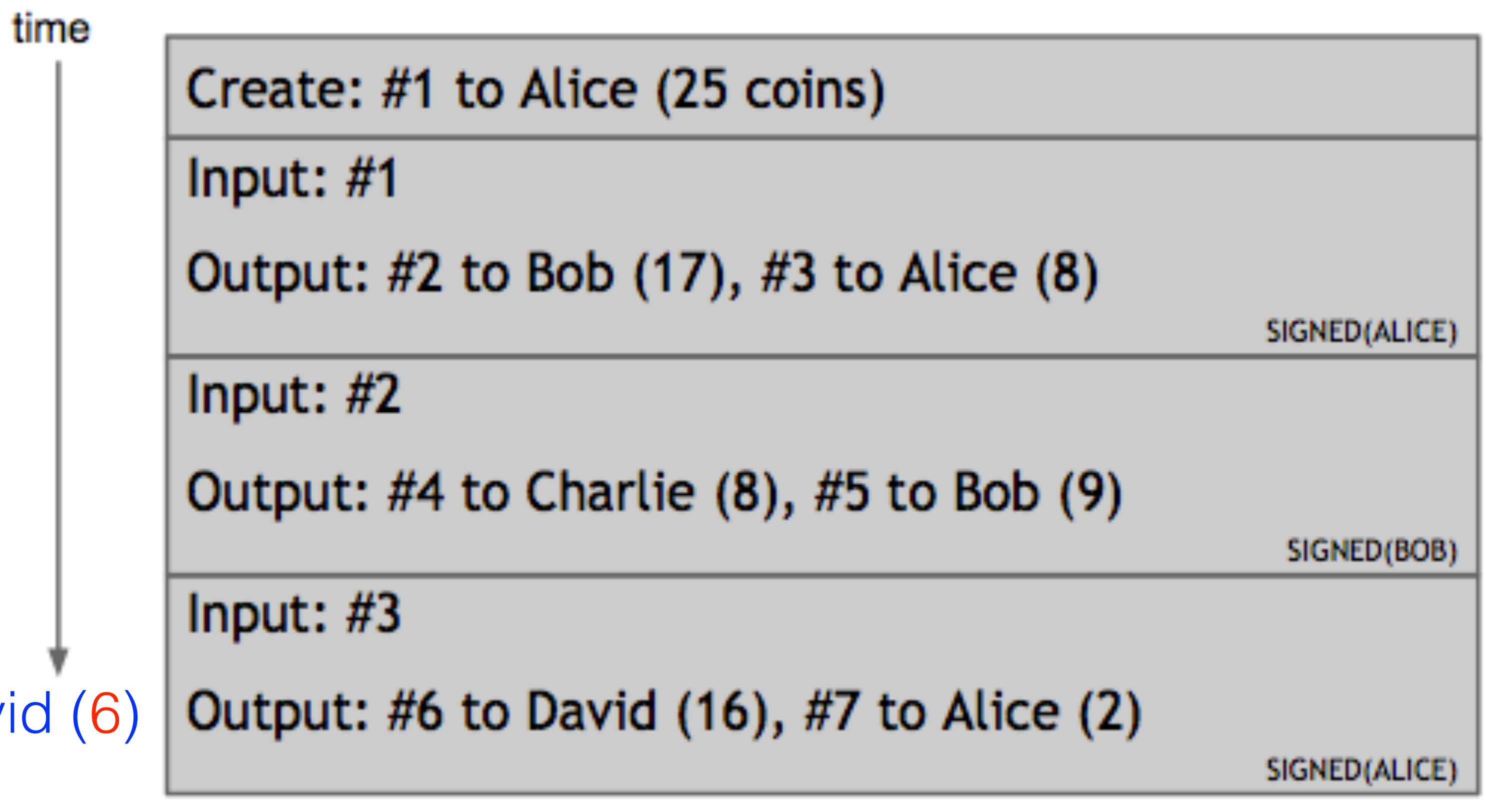
- Two types of nodes
  - Bitcoin users (use Bitcoins)
  - Miners (use & generate Bitcoins)
- Each node has a public key and a private key
- Use hash value of public key as account number (e.g., routing number on your check)
  - SHA256: H(pk), then convert to base56

1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2

1JBonneauruSSoYm6rH7XFZc6Hcy98zRZz

Bitcoin Transaction

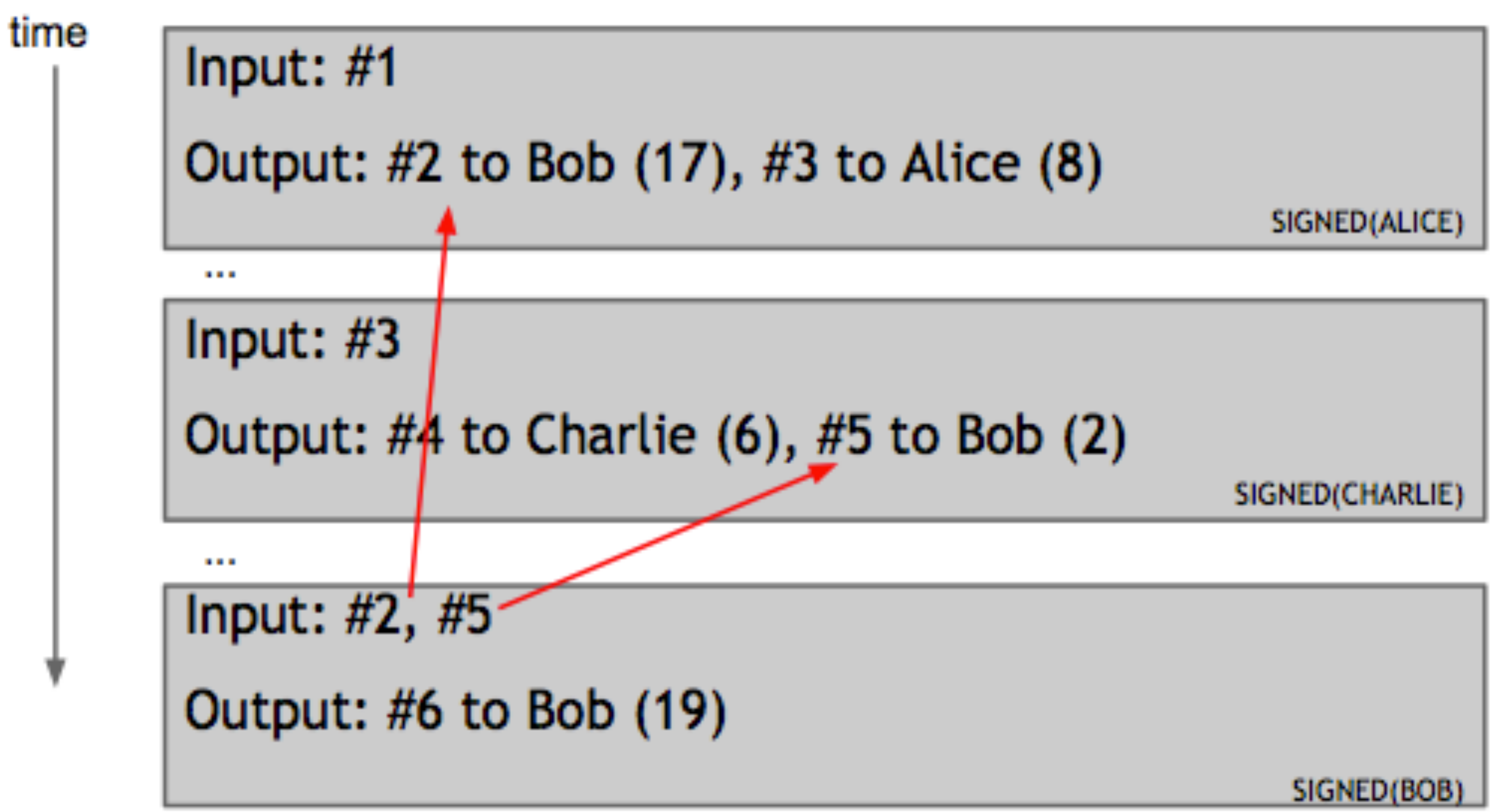
- Transactions are coin-based
  - E.g., coin #1 —> coin #2 & coin #3



#6 to David (6)

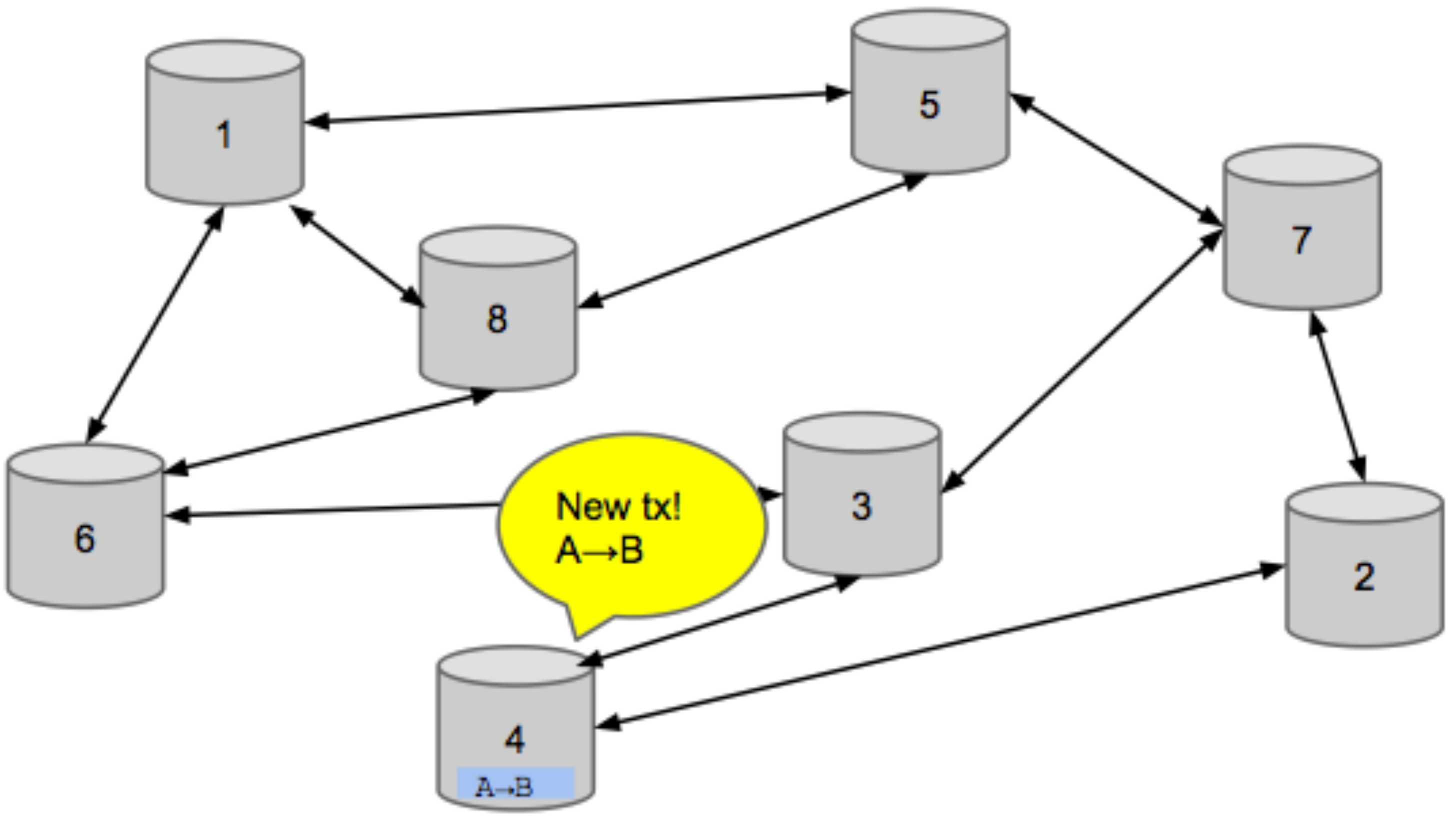
Bitcoin Transaction

- Bob can merge two coins into 1 coin
  - E.g., coin #2 and coin #5 —> coin #6



Transaction Propagation

- Each trans will be propagated to (almost) the entire network



Pending Transactions

- Alice pays Bob 17 bitcoins, this is a pending trans
- Alice will broadcast this pending trans
- Almost every miner will receive this pending trans
- Eventually, this pending trans will be included in a block with some other pending trans by a miner, and this block will be added to the current blockchain.
- This trans will be confirmed once it is in the blockchain (i.e. Bob will receive 17 bitcoins)

Pending Transactions

- There are many pending trans in Bitcoin network
- Someone needs to add those to a block, and add a new block to blockchain.
- Does not have a central authority, who will do it?

- A miner will add a new block to blockchain
  - Miner gets transaction fee: E.g., Alice pays 17 bitcoins to Bob, Bob gets 16.95, miner gets 0.05
  - Miner gets new bitcoins: 12.5 bitcoins/block

Mining

- Adding new blocks to blockchain is called mining
- There are many miners in the network, how to decide which miner add the next block?
  - Each miner is given a same "Puzzle"
  - Each miner selects its own block from current pending trans pool
  - Who finds a solution for "Puzzle" first based on its block will add its block to blockchain

Mining

- A miner cannot provide a fake solution, since others can verify
  - Finding a solution of “Puzzle” takes some effort, but given a solution, it is easy to verify
- Once a solution is verified, a corresponding block is added to the blockchain, everyone will start to find a solution for the next block.

- Bitcoin uses Proof of Work as “Puzzle”

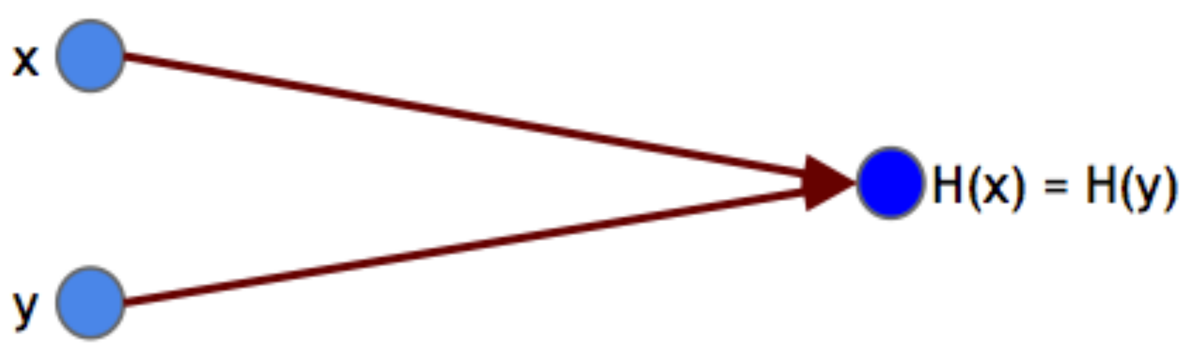
Proof of Work

- Alice wants to prove to Bob that she did certain amount of works (e.g., no. of operations or time)
  - Bob gives Alice a <u>target</u>
  - Alice finds a <u>solution</u> for this target
  - Bob verifies this solution based on this target

- Alice needs certain time to find a solution (time consuming, but still possible); With a solution and target, it is very easy for Bob to verify.

Hash Function Review

- Hash Function (e.g., SHA256):
  - An arbitrary-length input —> a fixed-length output
  - Deterministic function
  - Efficient to compute, but hard to invert
  - A collision: x != y, but H(x) == H(y)
    - A collision must exist, but hard to find



Proof of Work

- Assume hash func.'s outputs: uniformly distributed

- Alice has a message m, wants to give it to Bob
- Bob tells Alice to find a nonce (random) n s.t.
  - The output of H(m||n) is a valid hash value;
  - Otherwise he will not accept message m
- Alice takes any nonce, and sends (m, n) to Bob
- Bob verifies H(m||n) = h is a valid hash value, accepts message m

Proof of Work

- Assume the maximum hash value is $x = 2^L$
  - E.g., x has L bits, and all bits are 1s

- Alice has a message m, wants to give it to Bob
- Bob tells Alice to find a nonce n s.t.
  - $H(m||n) <= x$
  - Otherwise he will not accept message m
- Alice takes any nonce, and sends (m, n) to Bob
- Bob verifies $H(m||n) <= x$, accepts message m

H(*)	000	001	010	011	100	101	110	111

- Example:
    - A hash value has 3 bits, maximum value x = 111

- Alice has a message m, wants to give it to Bob
- Bob tells Alice to find a nonce n s.t.
    - H(m||n) <= x = 111
- Alice takes any nonce, sends (m, n) to Bob
    - n = uc; n = cincinnati; n = ohio; ……
- Bob verifies H(m||n) <= x, accepts message m

Proof of Work

- Assume the maximum hash value is $x = 2^L$
  - E.g., x has L bits, and all bits are 1s

- Alice has a message m, wants to give it to Bob
- Bob tells Alice to find a nonce n s.t.
  - $H(m||n) <= x/2$
  - Otherwise he will not accept message m
- Alice cannot choose any nonce in this case
  - Some hash values are $> x/2$; some are $<= x/2$

Proof of Work

- Bob tells Alice to find a nonce n s.t.
  - H(m||n) <= x/2;
- Alice chooses a random n,

```
while (true) {
  if (H(m||n) <= x/2), return n;
  else n = n + 1;
}
```

- <u>Enumerating all possible nonces is the only way to find a solution</u>

Example

- Bob tells Alice to find a nonce n s.t.
  - H(m||n) <= x/2;
- Example: Given m = **`uc`**, Alice chooses n = **`a`**
  - H(**`uc`**||**`a`**) > x/2, **`a`** is not a solution;
- Alice tries n = **`b`**
  - H(**`uc`**||**`b`**) > x/2, **`b`** is not a solution;
- Alice tries n = **`c`**
  - H(**`uc`**||**`c`**) <= x/2, **`c`** is a solution, return

Example

- Bob tells Alice to find a nonce n s.t.
  - H(m||n) <= x/2;

- Example: Given m = **`uc`**, Alice chooses n = **`1`**
  - H(**`uc`**||**`1`**) > x/2, **`1`** is not a solution;
- Alice tries n = **`2`**
  - H(**`uc`**||**`2`**) > x/2, **`2`** is not a solution;
- Alice tries n = **`3`**
  - H(**`uc`**||**`3`**) <= x/2, **`3`** is a solution, return

Proof of Work

- Alice finds nonce n, and sends (m, n) to Bob
- Bob verifies $H(m||n) <= x/2$, accepts message m
  - $x/2$ is called a target, n is called a solution

- For Alice
  - Assume outputs are uniformly distributed
  - Given a random n, the probability that
    - $Pr[H(m||n) <= x/2] = 1/2$
  - Find a solution after 2 different nonces (average)

H(*)	000	001	010	011	100	101	110	111

- Example:
  - A hash value has 3 bits, maximum value x = 111

- Bob tells Alice to find a nonce n s.t.
  - H(m||n) <= x/2 = 011

- Alice on average needs 2 different nonces to find n
- Bob verifies H(m||n) <= x/2, accepts message m
  - Bob only needs 1 hash to verify

- Example: H(m||n) <= x/2 = 011
- Given m = **uc**, try different randoms
- n = 1
  - H(**uc**||1) = 110 > 011, **1** is not a solution;
- n = 2,
  - H(**uc**||2) = 111 > 011, **2** is not a solution;
- n = 3
  - H(**uc**||3) = 001 <= 011 , **3** is a solution;
- n = 4
  - H(**uc**||4) = 000 <= 011 , **4** is a solution;
- n = 5
  - H(**uc**||5) = 100 > 011 , **5** is not a solution;

- <u>Practice:</u> H(m||n) <= x/2 = 011

- Given m = **`uc`**, try different randoms
- n = **`66`**, H(**`uc`**||**`66`**) = 111
- n = **`67`**, H(**`uc`**||**`67`**) = 001
- n = **`68`**, H(**`uc`**||**`68`**) = 011
- n = **`69`**, H(**`uc`**||**`69`**) = 101
- n = **`70`**, H(**`uc`**||**`70`**) = 010

- Which randoms are solutions?
- Which randoms are not solutions?

- Practice: H(m||n) <= x/2 = 011

- Given m = **`uc`**,
- n = **`66`**, H(**`uc`**||**`66`**) = 111 > 011
- n = **`67`**, H(**`uc`**||**`67`**) = 001 <= 011
- n = **`68`**, H(**`uc`**||**`68`**) = 011 <= 011
- n = **`69`**, H(**`uc`**||**`69`**) = 101 > 011
- n = **`70`**, H(**`uc`**||**`70`**) = 010 <= 011

- 66, 69 are not solutions
- 67, 68, 70 are solutions

Proof of Work

- Bob tells Alice to find a nonce n s.t.
  - $H(m||n) <= x/4$
  - Otherwise he will not accept message m
- For Alice
  - Assume outputs are uniformly distributed
  - Given a random n, the probability that
    - $Pr[H(m||n) <= x/4] = 1/4$
  - Find a solution after 4 different nonces (average)

H(*)	000	001	010	011	100	101	110	111

- Example:
    - A hash value has 3 bits, maximum value x = 111

- Bob tells Alice to find a nonce n s.t.
    - H(m||n) <= x/4 = 001

- Alice <u>on average</u> needs 4 different nonces to find n
- Bob verifies H(m||n) <= x/4, accepts message m
    - Bob only needs 1 hash to verify

Proof of Work

- Bob tells Alice to find a nonce n s.t.
  - $H(m||n) <= x/8$
  - Otherwise he will not accept message m

- For Alice
  - Assume outputs are uniformly distributed
  - Given a random n, the probability that
    - $Pr[H(m||n) <= x/8] = 1/8$
  - Find a solution after 8 different nonces (average)

H(*)	000	001	010	011	100	101	110	111

- Example:
  - A hash value has 3 bits, maximum value x = 111

- Bob tells Alice to find a nonce n s.t.
  - H(m||n) <= x/8 = 000

- Alice <u>on average</u> needs 8 different nonces to find n
- Bob verifies H(m||n) <= x/8, accepts message m
  - Bob only needs 1 hash to verify

H(*)	000	001	010	011	100	101	110	111

- Difficulty: the no. of starting 0 bits in a target
- The no. of hash operations at Alice increases exponentially with difficulty

Target	Difficulty	Hashes at Alice	Hashes at Bob
x=111	0	1	1
x/2= 011	1	2	1
x/4=001	2	4	1
x/8=000	3	8	1

H(*)	0000	0001	0010	0011	0100	0101	0110	0111
	1000	1001	1010	1011	1100	1101	1110	1111

- Practice: maximum value x = 1111

- If difficulty is 1, what is target? Average hashes to find a solution?
  - Target: 0111, on average 2 hashes find a solution

- If difficulty is 3, what is target? average hashes to find a solution?
  - Target: 0001, on average 8 hashes find a solution

H(*)	0000	0001	0010	0011	0100	0101	0110	0111
	1000	1001	1010	1011	1100	1101	1110	1111

- <u>Practice:</u> maximum value x = 1111
  - Assume $H(m||n_1) = 0001$, $H(m||n_2) = 0010$

- Difficulty is 2, target: 0011
  - is n_1 a solution? is n_2 a solution?
  - Yes (0001 <= t); Yes (0010 <= t)
- Difficulty is 3, target: 0001
  - is n_1 a solution? is n_2 a solution?
  - Yes (0001 <= t); No (0010 > t)

- Proof of Work: message m
    - maximum hash value $x = 2^L$
    - difficulty d
    - target t
    - nonce n
    - solution s

- Given a difficulty d, target $t = x/2^d = 2^{L-d}$
    - First d bits in t are 0s and rest L-d bits are 1s
- For a message m and a nonce n,
    - $Pr[H(m||n) <= t] = 1/2^d$
    - Average no. of hashes to find a solution s is 2^d

Proof of Work

- Given difficulty d, target $t = x/2^d = 2^{L-d}$
- For message m and nonce n,
  - $Pr[H(m||n) <= t] = 1/2^d$
- Average 2^d hashes to find a solution

- Example: difficulty $d = 10$, $x = 2^{20}$, $L = 20$
  - target $t = x/2^d = 2^{20}/2^{10} = 2^{10}$
    - 00000 00000 11111 11111
  - Average $2^d = 1024$ hashes to find solution

Proof of Work

- Given difficulty d, target $t = x/2^d = 2^{L-d}$
- Average 2^d hashes to find a solution

- Practice: difficulty $d = 20$, $x = 2^{160}$, $L = 160$
  - what is target t ?
  - how many hashes to find a solution?
  - how many hashes to verify a solution?
  - $t = x/2^d = 2^{160}/2^{20} = 2^{140}$;
  - $2^d = 2^{20} = 1048576$ to find; 1 hash to verify