: Pset #1 due today
Pset #2 out.

Today: E-cash & bitcoin

- Intro
- Electronic checks
- Bitcoin (See video by Prof Felten et al. bitcoinbook.cs.princeton.edu)
    - Version 1: GoofyCoin ← insecure
    - Version 2: ScroogeCoin ← secure but requires trusted party
    - Final version: Bitcoin. ← decentralized versio[n] of Scrooge coin.

Intro:

What properties can/should electronic money have

- What does "possessing val" mean?
- How can we tranfer val?

With physical money:

- hard to generate
- only one person at a time "owns" the money.

with bits:
- Easy to generate
- Bits can be copied ⇒ double spending.

Simple Example: Electronic checks

- Bank has PK_B, SK_B
- User has PK_u, SK_u, certificate on PK_u by bank.

Check = [cert. on PK_u, signed by bank
sign(SK_u, "pay Bob \$100, date, ser #")]

- Bank deposits check just once (using ser #).
  (Usual problem of overdrawn accounts...)

This works!

Q: Can we make payments more like cash?

Desirable properties:

- Non forgeable
- Not double-spendable
- Transferability: A can pay B
- Transitivity: B can use A's payment to pay C.
- Divisibility & combinibility.
- Efficiency (esp. for small money)

- Scalability
- Anonymity
- Decentralized (no trusted party)

Double spending: Key problem, especially in schemes that offer transitivity.

Alice can give "same" coin to Chuck & to Donald!

- Prevention: seems tough (using only bits)
- Detection: requires DB with all spending records (public ledger).

- Many approaches: Micromint (Rivest & Shamir 1996)
  Peppercorn (Micali & Rivest ?)

Bit Coin: [From online lectures by Felten et. al.]

Decentralized identity management

Identity = PK

- Each PK is an actor in the system.
- To "speak for" PK, you must know a matching SK

- Each user can create many identities, (as often as they wa
  simply by creating random key pairs (PK, SK).

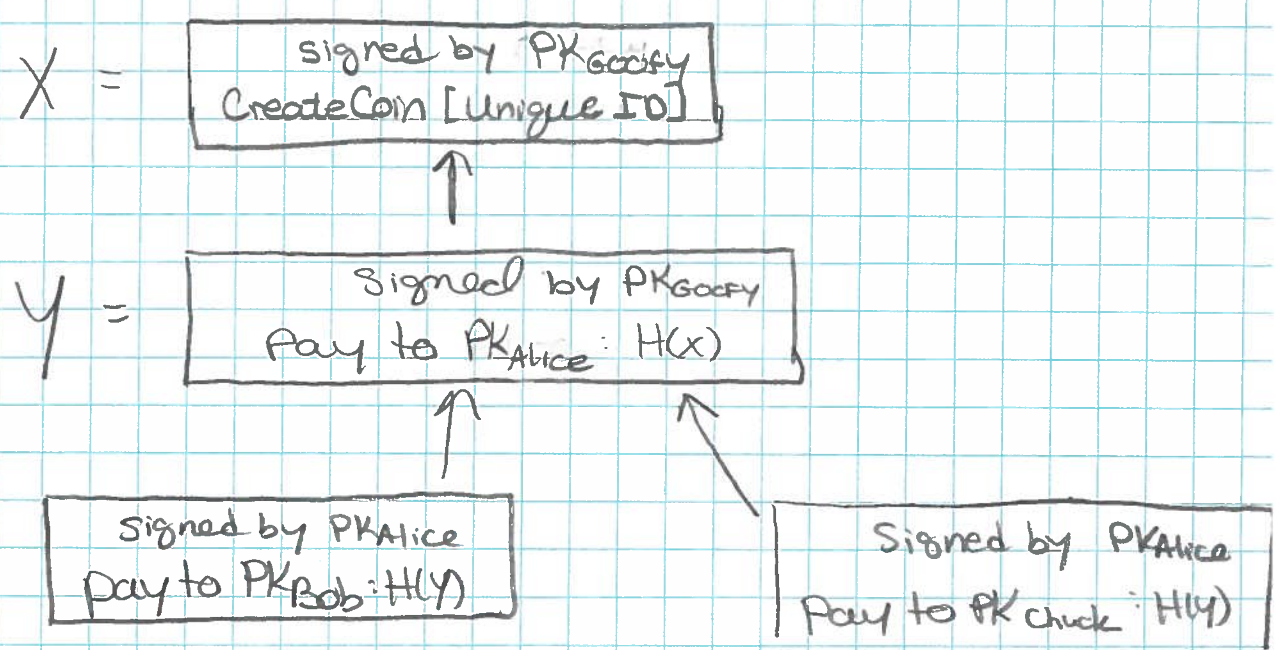
- No central place needed to register
  (no central control)

- These identities are often called "addresses".

- Is this anonymous? It is complicated.
  Depends how often you change your ID.

Version I: <u>GoofyCoin</u> (Centralized) & insecure

- Goofy can create new coins

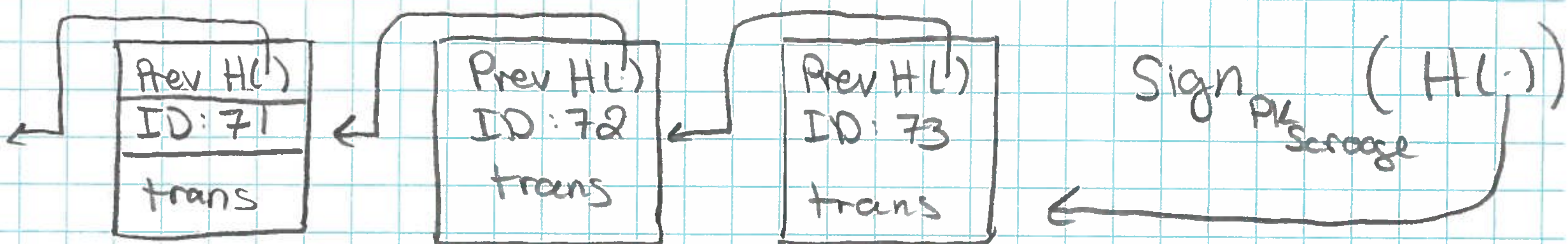


Problem: Double spending!

Version II: Scrooge Coin (Centralized) & secure

Idea: Scrooge will publish history of all the transactions that happened.

This will have the form of a block chain (as before), digitally signed by Scrooge.



Scrooge publishes this history (public ledger)

- This allows to detect double spending.

Two types of transactions:

① CreateCoins:

transID: 73 type: CreateCoins

	#	value	recipient
coinID 73(0) →	0	3.2	PK_1
CoinID 73(1) →	1	1.4	PK_2
CoinID 73(2) →	2	7.1	PK_3

Valid by def.

② PayCoins transaction :

Consumes & destroys some coins and creates new coins of same total value.

transID: 73	type: PayCoin	
consumed coins: 15(1), 42(0), 72(3)		
#	val	recipient
0	3.2	PK_1
1	1.4	PK_2
2	7.1	PK_3
signed by all owners of consumed coins		

Valid if :

- All consumed coins are valid & not already consumed
- Total value out = total val in
- signed by owners of all consumed coins.

If valid Scrooge will acknowledge & add to the public ledger & sign.

<u>Note</u> : Coins never subdivided or combined, only created & consumed.

But we get the same affect as if we are

able to combine & subdivid, by making a Paycoin transaction to that affect.

Scrooge Coin works!

<u>Core problem:</u> Scrooge!

This is a centralized solution.

- What if Scrooge becomes malicious?

Q: Can we make it decentralized?

<u>Bitcoin:</u> DeScroogify the system!

<u>Key Challenge</u>: <u>Distributed Consensus</u>

How can we provide the services that Scrooge provides in a decentralized way, where no specific party is trusted?

How can everyone agree on a single public block chain, which is the agreed upon history of which transaction happened?

Bitcoin is a peer-to-peer system:

When Alice wants to pay Bob she broadcasts the transaction to all Bitcoin nodes.

Signed by PK_{Alice}
Pay to PK_{Bob} 2 Bitcoins
H( )

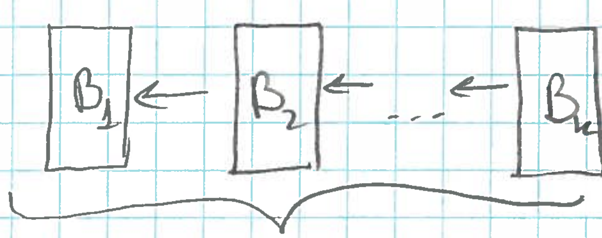
link this coin to her receipt of this coin from someone else in the past.

How could consensus work?

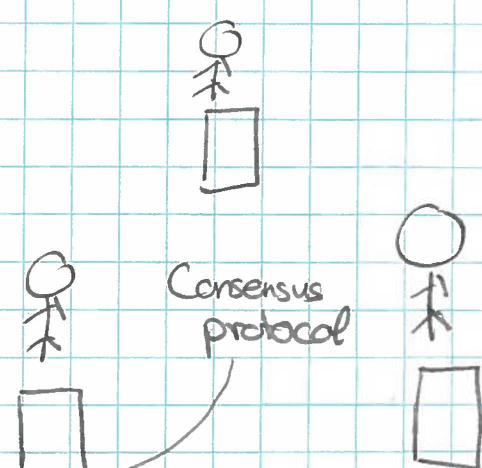
At any given time:

for efficiency

- All nodes have a sequence of blocks of transactions they have reached consensus on.
- Each node has a set of outstanding transactions it has heard about. (different nodes may have diff. versions)



agreed upon



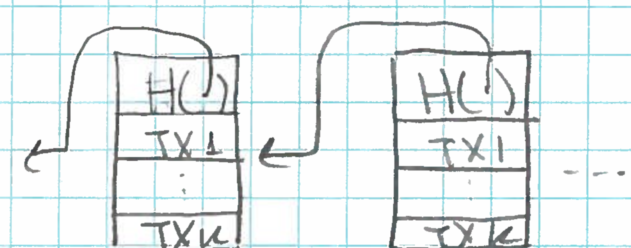
Goal: Agree on any valid block, even if proposed by only one node.

Simplifying assumption:

A random node is selected to add a block, no one else can add!

In this case:

- New transactions are broadcasted to all nodes.
- Each node collects new transactions into a block.
- for simplicity → A random node is selected to add next block. This node broadcasts his block.
- Other nodes accept block iff all transactions in it are valid (unspent & valid sigs).
- Nodes express their acceptence of the block by including its hash in the next block they create.



This is very close to Bitcoin!

Is this secure? What can a malicious node do?

① Denial of service: Adv can decide not to include transactions to Bob.

$\Rightarrow$ Bob will need to wait until next block.

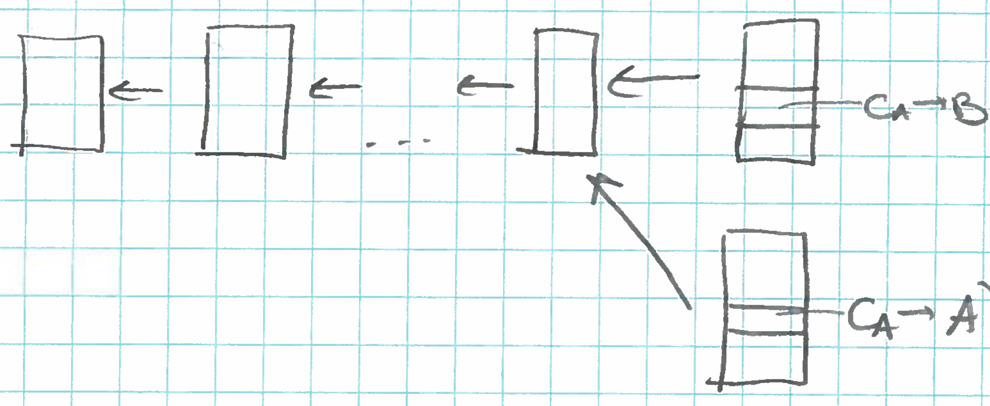
② Double spending:

Alice can give Bob a coin by generating a (signed) transaction $C_A \to B$

But can also generate and broadcast a transaction $C_A \to A'$

where $PK_{A'}$ belongs to Alice.

How can Bob protect himself?



Honest nodes always extend longest valid chain.

Bob can wait until a few more blocks are added

(recommendation: wait 6 confirmations ie., until 6 additional blocks are added)

Bitcoins removes simplifying assumption and "selects random node" using:

Proof of Work

Instead of picking random node, we "approximate" selecting a random node:
Select nodes in proportion to a resource (we hope) no one can monopolize: computing power (or proof of work)

Hash puzzles

To create a block, must find nonce s.t.

$$H(\text{nonce} \,\|\, \text{prev\_hash} \,\|\, \text{transactions of this block}) < \text{target} = (0^k \ast)$$

If hash function is secure, then the only way to succeed is to try enough nonces until you get lucky.

- Instead of selecting a random node, nodes are competing to solve a hash puzzle.

The lucky node that found a good nonce gets to propose the next block.

Completely Decentralized !!

<u>PoW Property 1</u>: Difficult to compute.

As of Aug. 2014: $K \approx 10^{20}$.

- Only a few nodes bother to compete – miners.
- A lot of concentration of power in mining echo system (even though technically anyone can be a miner). ← undesirable

<u>PoW Property 2</u>: Parameterizable cost (not fixed throughout time).

Goal: Avg. time between any two succ blocks ≈ 10 min.

<u>PoW Property 3</u>: Trivial to verify.

$$H(\text{nonce} \,\|\, \text{prev\_hash} \,\|\, \text{transactions of this block}) < \text{target}.$$

Key security assumption:

Secure if majority of miners, weighted by hash power follow the protocol (i.e. honest)

* Utilize currency to incentivize nodes to behave honestly.

Incentive 1: Block reward

Creator of block gets to:

- Include special coin-creation transaction in block & choose recipient address (PK).
- Value is fixed:
  - Currently 25 BTC
  - Halves every 4 years
  - Run out in 2040

[Finite supply of 21 million BTCs]

Why does this incentivize honesty?

- Block creator gets to "collect" the reward only if block ends up on long-term consensus branch.
- Incentivizes nodes to behave in a way other nodes agree with.

Incentive 2: Transaction fees

Creator of transaction can choose to make output val less than input val:
Remainder is a transaction fee that goes to block creator.

Recap:

Identities: No need for real world identities

Transactions: Msgs broadcast in peer-to-peer network.
These are instructions to transfer a coin from one address to another.

P2P network: Its goal is to propegate new transactions and new blocks.

Block chain & consensus: Where the security of the system comes from

Hash puzzles & mining: Miners are nodes that bother to compete in creating new blocks.

* # of bits I own is subject to consensus.

Ownership of a bit coin = Other nodes think I own this bitcoin.