BIT COIN -1

Reference

Bitcoin and Cryptocurrency Technologies

By Arvind Narayanan, Joseph Bonneau, Edward Felten,

Andrew Miller, Steven Goldfeder

References

- Main reference: *Bitcoin and Cryptocurrency Technologies,* By Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder
- Slides are mainly taken (or adapted) from the book and from slides of the authors of the text

Methods of payments in Internet

- Most payments in Internet require a credit (debit) card
- There are other forms (e.g. Paypal) that are based on a trusted authority
- Payments are expensive
- Not anonymous

BITCOIN

Introduction and key aspects

Paying in the Web: SSL

SSL and credit card are used for paying

- Simple, no need of specialized software
- compliant with credit card mechanisms
- most used method for paying in the web

Problems

- malicious sellers have info on clients
- clients can in principle refuse to pay (there is no signature)
- many disputes (20%- 60%)
- expensive method for the shop
- not anonimous

Alternative: PAYPAL (reduces complaints, need a cetificate)

Secure Electronic Transactions (SET)

SET developed in 1996 by Mastercard, Visa etc to protect Internet credit card transactions

- not a payment system
- rather a set of security protocols & formats
  - secure communications amongst parties
  - trust from use of X.509v3 certificates
  - privacy by restricted info to those who need it

Problems:

- All users must have a certificate (complicated)
- Anonimity between clients and merchant but there is a Central authority: knows everything
- Costly (you have to pay credit card companies)

Money: properties

- Recognizable
- Divisible
- Transportable
- Transferable
- Scarce
- Hard to counterfeit
- Stable supply
- Durable
- Stable value
- Anonymous

Money: properties

- Recognizable
- Divisible
- Transportable
- Transferable
- Scarce
- Hard to counterfeit
- Stable supply
- Durable
- Stable value
- Anonymous

Bitcoin
additional properites

- Decentralized: no central authority
- Immune to sovereign censorships, confiscation

Bitcoin

- Bitcoin is a combination of several things: a currency, a payment system, and a collection of algorithms and software implementations.
- The goal of bitcoin is to enable payments with low transaction costs. Bitcoin can also sometimes provide anonymity.
- One bitcoin (BTC) is worth about $1141 (march 2017)
- Approximately 15.2 million bitcoins have been created (mined) to date, for a total value of approximately $ 17.34 billion.

bmm

How to use Bitcoin

- Download software to create a Bitcoin wallet (see https://bitcoin.org/en/choose-your-wallet)

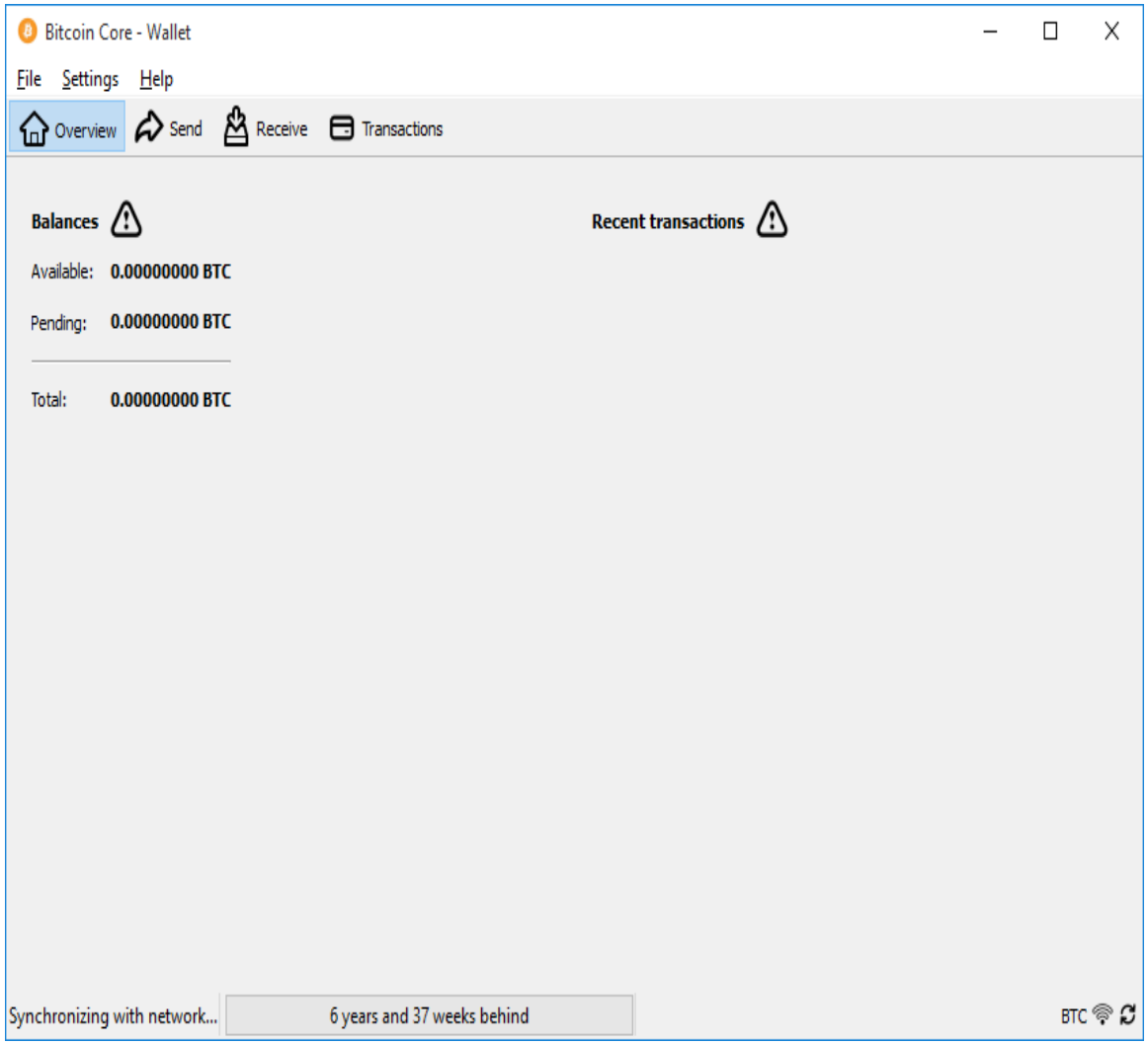
- The wallet holds the private keys you use to prove you own specific Bitcoins.

- The software creates public/private key pairs for you as needed. For each pair, there is a corresponding bitcoin address, which is a 160-bit hash of the public key. Bitcoins are sent to addresses.

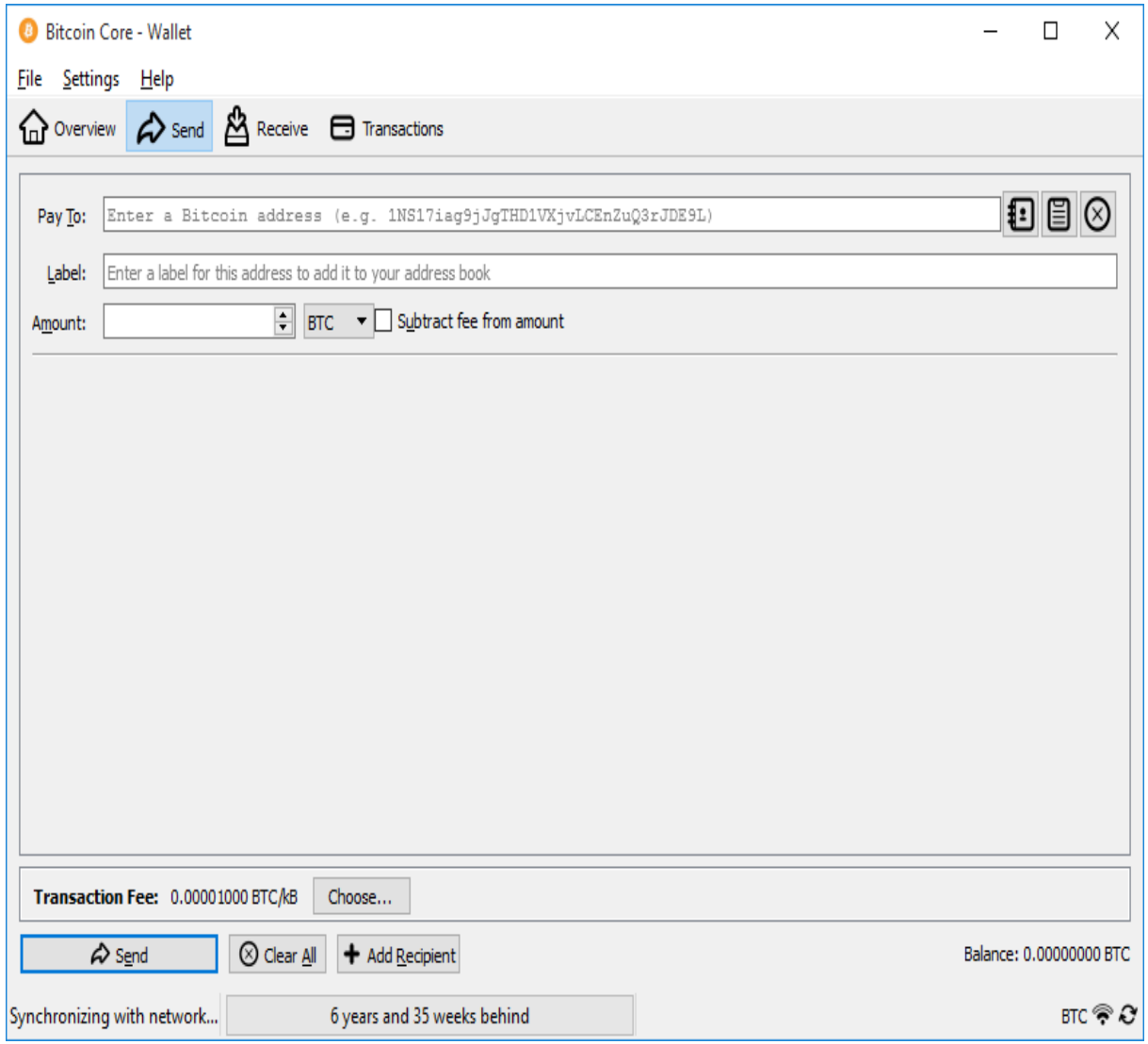
- The wallet also contains software that allows you to send and receive bitcoins. You send bitcoins by registering your payments in the block chain, which is bitcoin's public ledger containing all transactions since the beginning of bitcoin.

-

Bitcoin Core (original) wallet on first start-up



Send (pay)


Bitcoin Core - Wallet
File Settings Help
Overview Send Receive Transactions
Pay To: Enter a Bitcoin address (e.g. 1NS17iag9jJgTHD1VXjvLCEnZuQ3rJDE9L)
Label: Enter a label for this address to add it to your address book
Amount: BTC Subtract fee from amount
Transaction Fee: 0.00001000 BTC/kB Choose...
Send Clear All Add Recipient
Balance: 0.00000000 BTC
Synchronizing with network...
6 years and 35 weeks behind
BTC

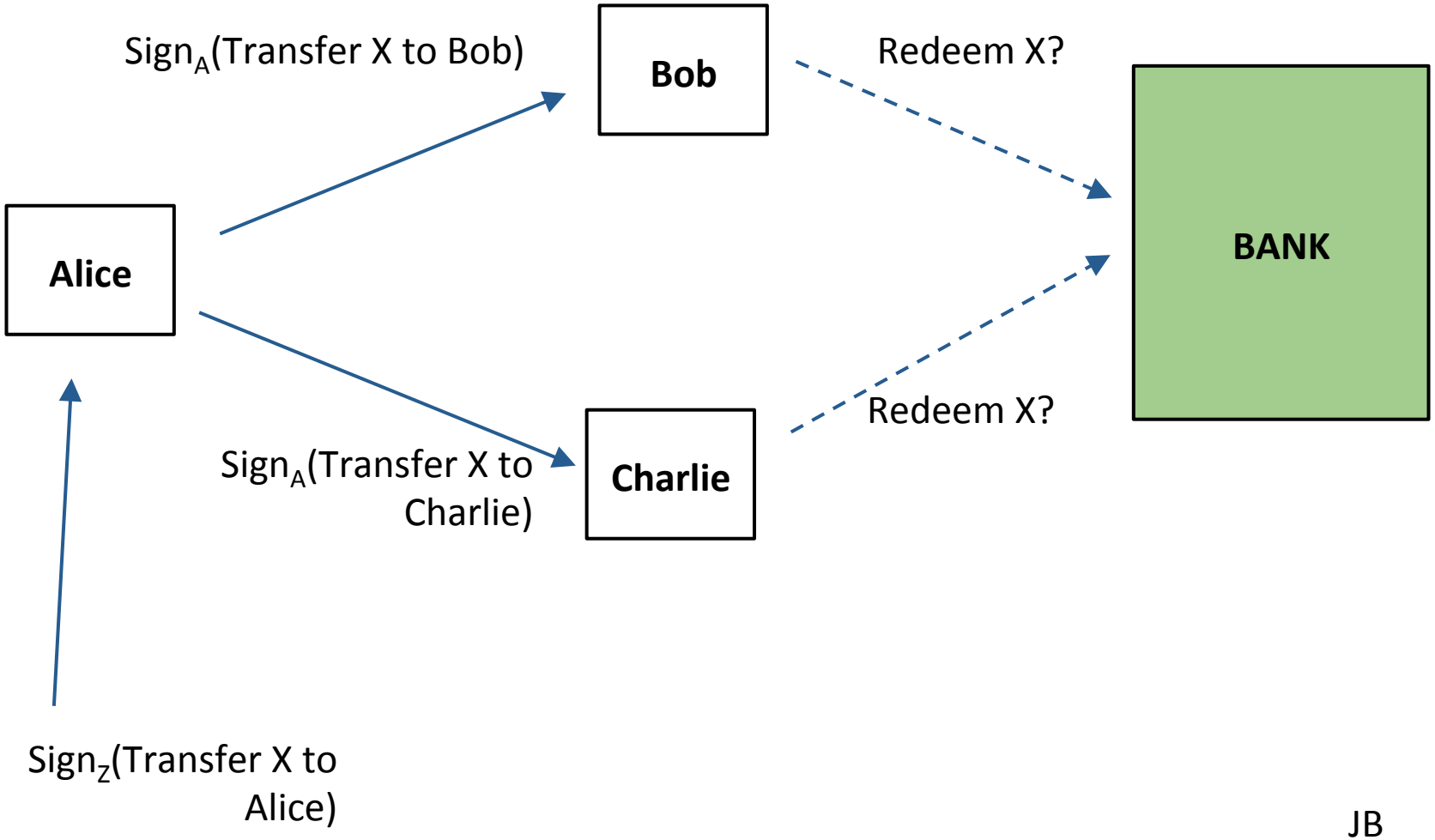
bmm

Spending Bitcoin

- A transaction is of the form “send these Bitcoins from address Y to address Z”
- Specific Bitcoins are described as outputs of previous transactions.
- The transaction is signed with the private key of address Y and broadcast, along with the public key of Y, to the payment network
- A transaction might also include a transaction fee, to be described later.

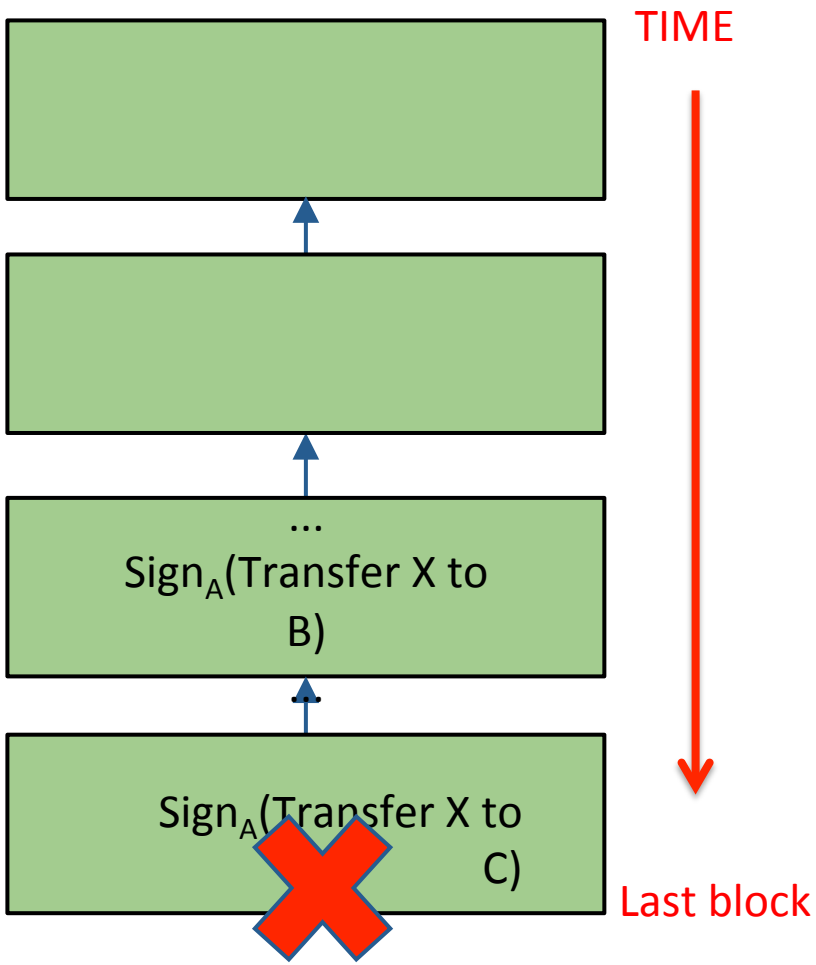
bmm

Double spending: why ecash is hard



JB

Solution: Maintain a global public append-only log



*The **block chain** - a public ledger of **all transactions**.*

In Bitcoin, the log is extended in increments of **blocks**, each of which may contain thousands of transactions. (A new block is created every ~10 minutes)

Here the last block is cancelled because contains transfer of X by A that is not the owner anymore

Bitcoin: Make the bank a global log

Questions

- Who stores the ledger?
- How is a new block created?
- How is a transaction verified?
- How to avoid double spending?
- How to reward those who maintain the public ledger of all transactions?
- How secure is Bitcoin?
- Why does Bitcoin have value?

Preliminaries

Bitcoin uses three cryptographic tools

- Hash functions
- Digital signatures
- Public key

Hash functions

Properties

- Its input can be any string of any size
- It produces a fixed size output; as an example SHA-256 (256-bit output size) (However, our discussion holds true for any output size as long as it is sufficiently large.
- It is efficiently computable: computing the hash of a

  n -bit string should have a running time that is O(n).

Cryptographic hash functions

A Hash function that verifies the following Properties

- **Collision-resistance**: A hash function H is said to be collision resistant if it is infeasible to find two values, x and y, such that $x \neq y$, yet $H(x)=H(y)$.
- **Hiding**. A hash function H is hiding if: when a value r is chosen from a probability distribution that has high min-entropy, then given $H(r \mid x)$, it is infeasible to find x.
- **Puzzle friendliness**. A hash function H is said to be puzzle-friendly if 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 $H(k \mid x) = y$ in time significantly less than 2^n.

***High min-entropy**: 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.

Cryptographic hash functions

Collision-resistance: A hash function H is said to be collision resistant if it is infeasible to find two values, x and y, such that $x \neq y$, yet $H(x)=H(y)$

- The input is a string of any lenght and the output is 256 bits, hence there are many x,y s.t. H(x)=H(y) -pigeonhole principle
- Birthday paradox: if the output is 256 requires ~2^{255} to have 50% probability to find a collision by repeatedly trying a random string (10,000 hashes per second, it would take more than one octillion (10^{27}) years to calculate 2^{128} hashes
- The cryptographic hash functions that we use in practice are just functions for which people have tried really, really hard to find collisions and haven't yet succeeded.
- Message digest: we upload a large file M with y=H(M), and we want to be able to verify later that the file M' we download is the same (M=M') it is sufficient to check that H(M')=y=H(M)

Commitment scheme

A commitment scheme consists of two algorithms:

- (com, key) := commit(msg)

  The commit function takes a message as input and returns two values, a commitmeny – com - and a key.

- isValid := verify(com, key, msg)

  The verify function takes a commitment, key, and message as input. It returns true if the com is a valid commitment to msg under the key, key. It returns false otherwise.

We require that the following two security properties hold:

- Hiding: Given com, it is infeasible to find msg
- Binding: For any value of key, it is infeasible to find two messages, msg and msg' such that msg ≠ msg' and verify(commit(msg), key, msg') == true

Commitment scheme

- To use a commitment scheme, one commits to a value, and publishes the commitment com. This stage is analogous to putting the sealed envelope on the table.
- At a later point, if they want to reveal the value that they committed to earlier, they publish the key, key and the value, msg. Now, anybody can verify that msg was indeed the value committed to earlier. This stage is analogous to opening up the envelope.

Commitment scheme

Consider the following commitment scheme that uses a hash fiunction:

- commit(msg) := (H(key | msg), key)

  where key is a random 256-bit value
- verify(com, key, msg) := true if H(key | msg) = com; false otherwise

In this scheme, to commit to a value, we generate a random 256-bit value, which will serve as the key.

Search Puzzle

A search puzzle consists of

- a hash function, H
- a value, id (which we call the puzzle-ID), chosen from a high min-entropy distribution
- and a target set Y

A solution is a value x such that $H(id \mid x) \in Y$.

Puzzle friendliness. A hash function H is said to be puzzle-friendly if 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 $H(k \mid x) = y$ in time significantly less than 2^n

Intuition: if H has an n-bit output, then it can take any of 2^n values.

Solving the puzzle requires finding an input so that the output falls within the set Y, which is typically much smaller than the set of all outputs.

The size of Y determines how hard the puzzle is:

- If Y is the set of all n-bit strings the puzzle is trivial
- if Y has only 1 element the puzzle is maximally hard (hiding property)
- If Y is the set of all strings ending with 0; then I expect that if I try a random x I have 0.5 probability that the last bit of $H(id \mid x)$ is 0.

We want that there is no way to improve significantly 0.5

Search Puzzle

A search puzzle consists of

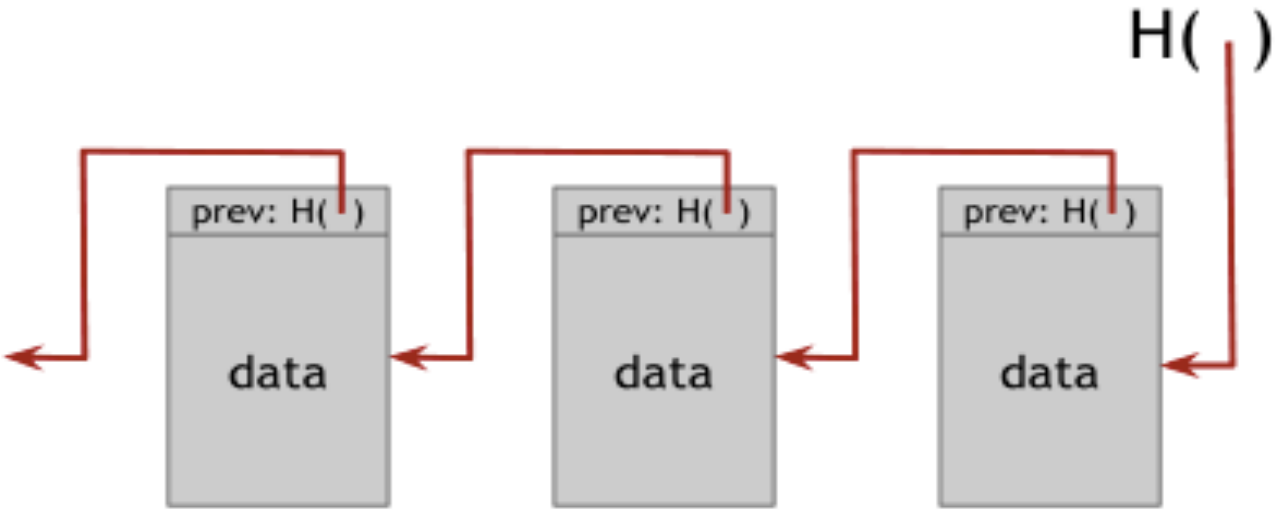
- a hash function, H
- a value, id (which we call the puzzle-ID), chosen from a high min-entropy distribution
- and a target set Y

A solution is a value x such that $H(id \mid x) \in Y$.

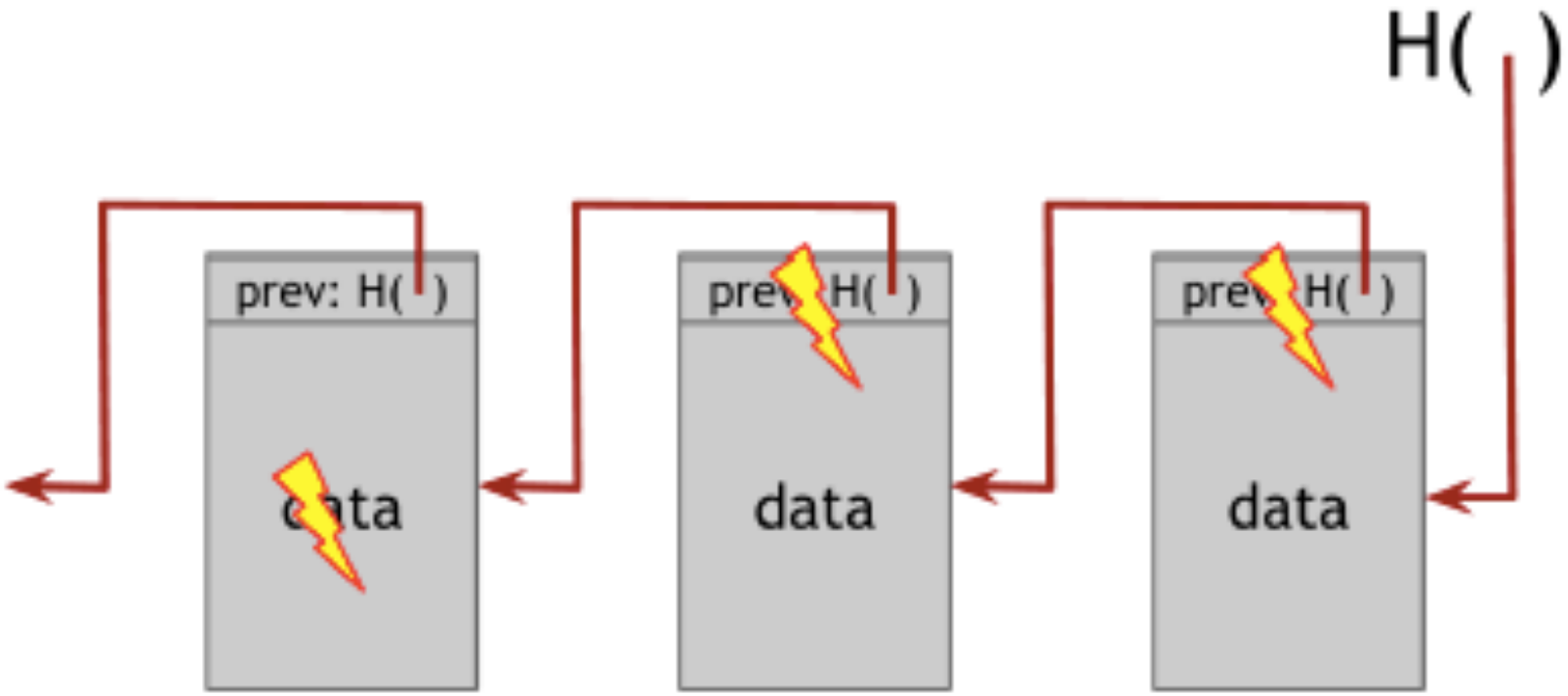
Solving the puzzle requires finding an input so that the output falls within the set Y, which is typically much smaller than the set of all outputs.

- The size of Y determines how hard the puzzle is.
- If Y is a string of 256 bits and cardinality of is k then if we try a random x we expect that the probability that $H(id \mid x) \in Y$ is ~ $k / 2^{256}$
- If a search puzzle is puzzle-friendly, this implies that there's no solving strategy for this puzzle which is much better than just trying random values of x.

BLOCK CHAIN



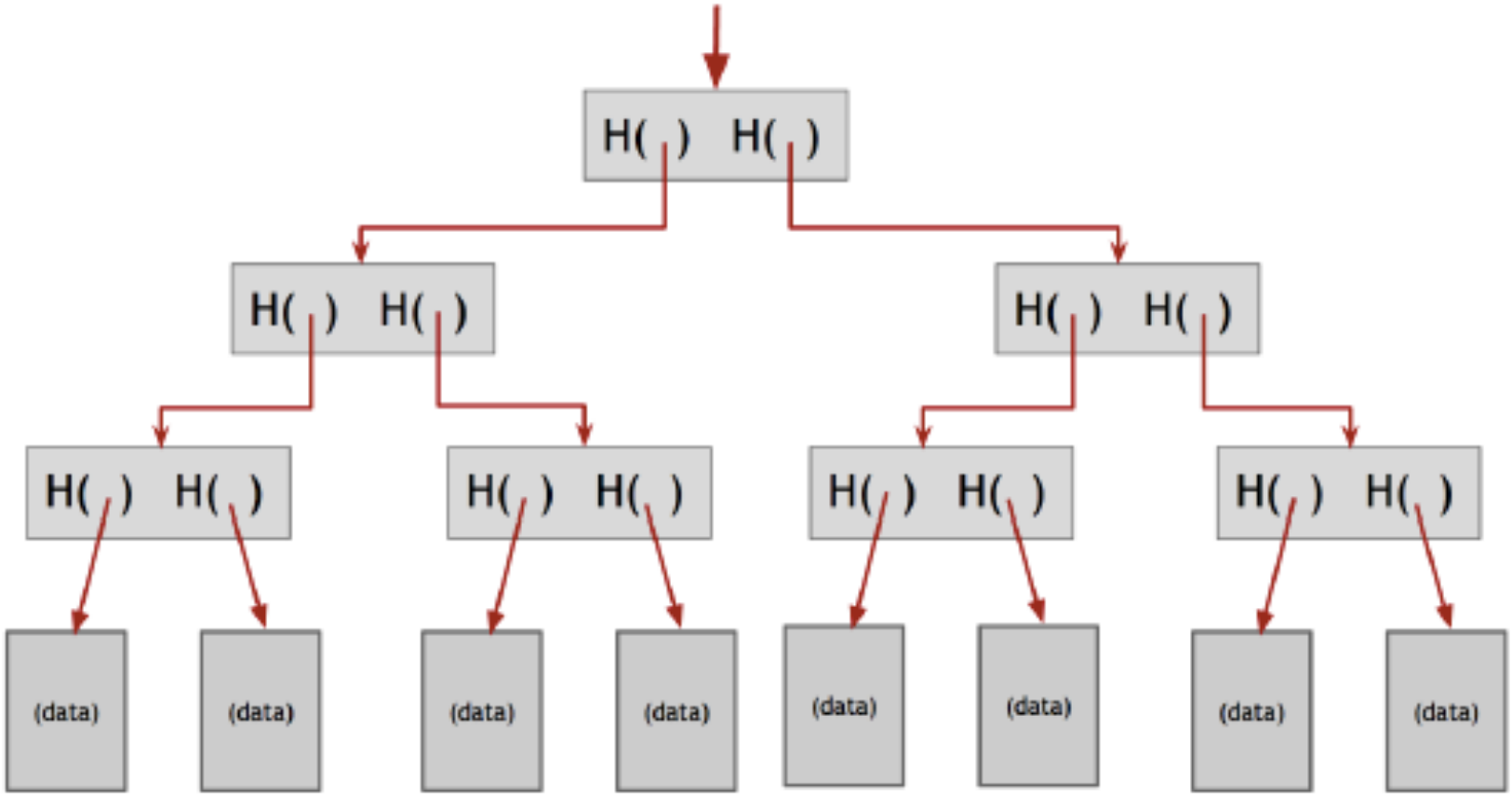
A use case for a block chain is a tamper-evident log:
a log data structure that stores a lot of data, and allows us to append data onto the end of the log.
But if somebody alters data that is earlier in the log, we're going to detect it.



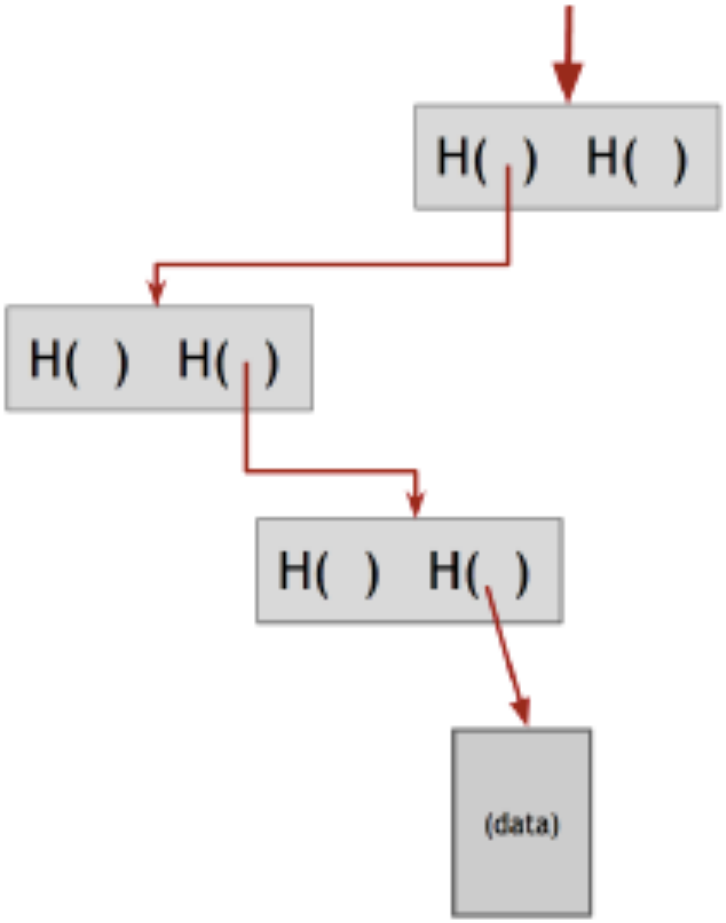
If an adversary modifies data anywhere in the block chain, it will result in the next hash pointer being incorrect.

If we store the head of the list, then even if the adversary modifies all of the pointers to be consistent with the modified data, the head pointer will be incorrect, and we will detect the tampering.

Merkle tree



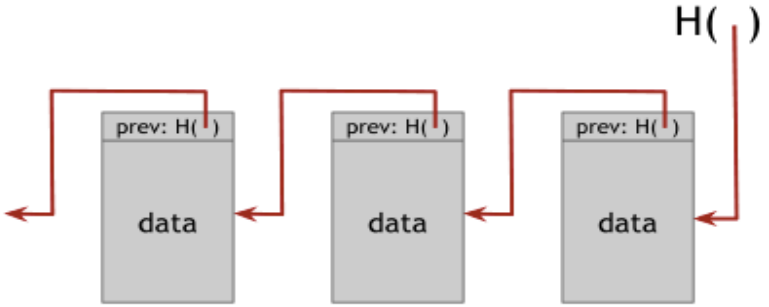
Merkle tree: proof of membership



To prove that a data block is included in the tree, one only needs to show the blocks in the path from that data block to the root.

Note: in the case of a list you might search and check all blocks

BLOCK CHAIN Summary



- The blockchain is a data structure that stores transactions that is similar to a linked list. data is split into containers - the blocks.
- A block consists of a header, and the transactions contained. Inside the block, a Merkle tree is used.
- Each block is connected with its predecessor with a cryptographically secured reference. This makes the data structure tamper-evident, changes to old blocks are easy to detect and dismissed.
- Development and maintenance of a public blockchain are expensive, but digital money can justify this overhead.
- Blockchains don't guarantee truth; they preserve truth & lies from later alteration, allowing one to securely analyze them and be more confident in uncovering the lies.

Digital signatures

A digital signature scheme consists of three algorithms:

- (sk, pk) := generateKeys(keysize) The generateKeys method takes a key size and generates a key pair. The secret key sk is kept privately and used to sign messages. pk is the public verification key that you give to everybody.
- sig := sign(sk, message) : takes a message, msg, and a secret key, sk, as input and outputs a signature for the msg under sk
- isValid := verify(pk, message, sig) : takes a message, a signature, and a public key as input. It returns a boolean value, isValid, that will be true if sig is a valid signature

We require that the following two properties hold:

- Valid signatures must verify

  verify(pk, message, sign(sk, message)) == true

Signatures are existentially unforgeable

Decentralized identity management

Rather than having a central authority that you have to go in order to register as a user in a system, you can register as a user all by yourself.

You don’t need to be issued a username nor do you need to inform someone that you’re going to be using a particular name. If you want a new identity, you can just generate one at any time, and you can make as many as you want.

.

Decentralized identity management

EXAMPLE:

- If you prefer to be known by five different names, no problem! Just make five identities.
- If you want to be somewhat anonymous for a while, you can make a new identity, use it just for a little while, and then throw it away.

All of these things are possible with decentralized identity management, and this is the way Bitcoin does identity. These identities are called **addresses**, in Bitcoin jargon.

In the context of Bitcoin and cryptocurrencies an address is a hash of a public key. It's an identity that someone made, as part of this decentralized identity management scheme.

Two attempts

We now see two attempts to check the problems

1. Goofy coin: decentralized but allows double spending
2. Scrooge coin: centralized no double spending

GoofyCoin

Two rules: 1. CREATE COIN

CREATE COIN: The first rule is that a designated entity, Goofy, can create new coins whenever he wants and these newly created coins belong to him. To create a coin, Goofy generates a unique coin ID uniqueCoinID that he's never generated before and constructs the string

"CreateCoin [uniqueCoinID]".

He then computes the digital signature of this string with his secret signing key. The string, togetherr with Goofy's signature, is a coin.

Anyone can verify that the coin contains Goofy's valid signature of a CreateCoin statement, is a valid coin.

Goofy Coin

Second rule: 2. Spending

whoever owns a coin can transfer it on to someone else using digital signatures or other crypto function

Example

- Goofy wants to transfer a coin that he created to Alice. To do this he creates a new statement that says “Pay this to Alice” where “this” is a hash pointer that references the coin in question.
- Recall that identities are really just public keys, so “Alice” refers to Alice’s public key. Finally, Goofy signs the string representing the statement.
- Since Goofy is the one who originally owned that coin, he has to sign any transaction that spends the coin. Once this data structure representing Goofy’s transaction signed by him exists, Alice owns the coin. She can prove to anyone that she owns the coin, because she can present the data structure with Goofy’s valid signature.
- Furthermore, it points to a valid coin that was owned by Goofy. So the validity and ownership of coins are self-evident in the system.

GoofyCoin: double spending

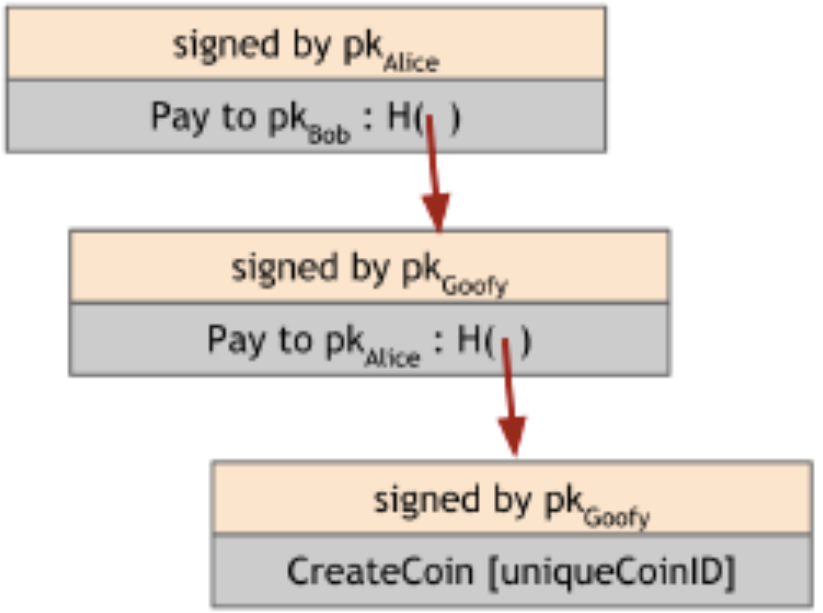
Alice can spend similarly. she creates a statement that says,

"Pay this coin to Bob's public key" where "this" is a hash pointer to tl coin that was owned by her.

Double spending

Alice signs

1. "Pay this coin to Bob's public key"
2. "Pay this coin to Charlie's public key"



Scrooge coin

A designated entity called Scrooge publishes a history of all the transactions that have happened. To do this he uses a block chain, which is digitally signed by Scrooge

Scrooge Coin: payment

PayCoins transaction is valid if four things are true:

1. The consumed coins are valid, that is, they really were created in previous transactions.
2. The consumed coins were not already consumed in some previous transaction. That is, that this is not a double-spend.
3. The total value of the coins that come out of this transaction is equal to the total value of the coins that went in. That is, only Scrooge can create new value.
4. The transaction is validly signed by the owners of all of the consumed coins.

Scrooge Coin: payment

If all previous conditions are met, then this PayCoins transaction is valid and Scrooge will accept it.

- Scrooge writes it into the history by appending it to the block chain, after which everyone can see that this transaction has happened.
- Double spending is avoided: It is only when Scrooge publisehs the transaction that the participants can accept that the transaction has actually occurred, it might be preempted by a douple-spending transaction even if it is otherwise valid by the first three conditions.

Scrooge Coin: payment

- Coins in this system are immutable — they are never changed, subdivided, or combined. Each coin is created, once, in one transaction and later consumed in some other transaction
- To subdivide a coin, Alice create a new transaction that consumes that one coin, and then produces two new coins of the same total value. Those two new coins could be assigned back to her.
- So although coins are immutable in this system, it has all the flexibility of a system that didn't have immutable coins.

Scrooge coin: the problem

ScroogeCoin allows people to see which coins are valid. It prevents double-spending, because everyone can look into the block chain and see that all of the transactions are valid and that every coin is consumed only once.

The problem is Scrooge — he has too much influence.

- He can't create fake transactions, because he can't forge other people's signatures.
- But he could stop endorsing transactions from some users, denying them service and making their coins unspendable. If Scrooge is he could refuse to publish transactions unless they transfer some mandated transaction fee to him.
- Scrooge can also of course create as many new coins for himself as he wants. Or Scrooge could get bored of the whole system and stop updating the block chain completely.
- The problem here is centralization. Although Scrooge is happy with this system, we, as users of it, might not be.

Scrooge coin: the problem

The problem is Scrooge — he has too much influence.

- He can't create fake transactions, because he can't forge other people's signatures.
- But he could stop endorsing transactions from some users, denying them service and making their coins unspendable. If Scrooge is he could refuse to publish transactions unless they transfer some mandated transaction fee to him.
- Scrooge can also of course create as many new coins for himself as he wants. Or Scrooge could get bored of the whole system and stop updating the block chain completely.
- The problem here is centralization. Although Scrooge is happy with this system, we, as users of it, might not be.

QUESTION: Can we have a cryptocurrency that operates like ScroogeCoin in many ways, but doesn't have any central trusted authority? Answer: BITCOIN

Bitcoin Distributed Consensus

Bitcoin

- Personal identities based on public key to guarantee anonimity (as in Goofy and Scrooge protols)
- Cryptography to guarantee authentication of transactions (as in Goofy and Scrooge protoc.)
- There is no central authority (as in Scrooge prot.)
- The network of users keep many copies of the **ledger** (=accounting book) consistently (i.e. there is an agreement of the correct one)
- The correct transactions are those contained in the majority of ledgers

Decentralization

Decentralization is not all or nothing; almost no system is purely decentralized or purely centralized.

Example, e-mail is fundamentally a decentralized system based on a standardized protocol, SMTP, and anyone who wishes can operate an email server of their own.

Yet, what has happened in the market is that a small number of centralized webmail providers have become dominant. Similarly,

Bitcoin protocol is decentralized, services like Bitcoin exchanges, where you can convert Bitcoin into other currencies, and wallet software, or software that allows people to manage their bitcoins may be centralized or decentralized to varying degrees.

Bitcoin achieves decentralization

Five specific questions:

1. Who maintains the ledger of transaction?
2. Who has authority over which transactions are valid?
3. Who creates new bitcoins?
4. Who determines how the rules of the system change?
5. How do bitcoins acquire exchange value?

We consider the first three questions that reflect the technical details of the protocol

Distributed Consensus Protocol

There are n nodes that each have an input value.

Some of these nodes are faulty or malicious.

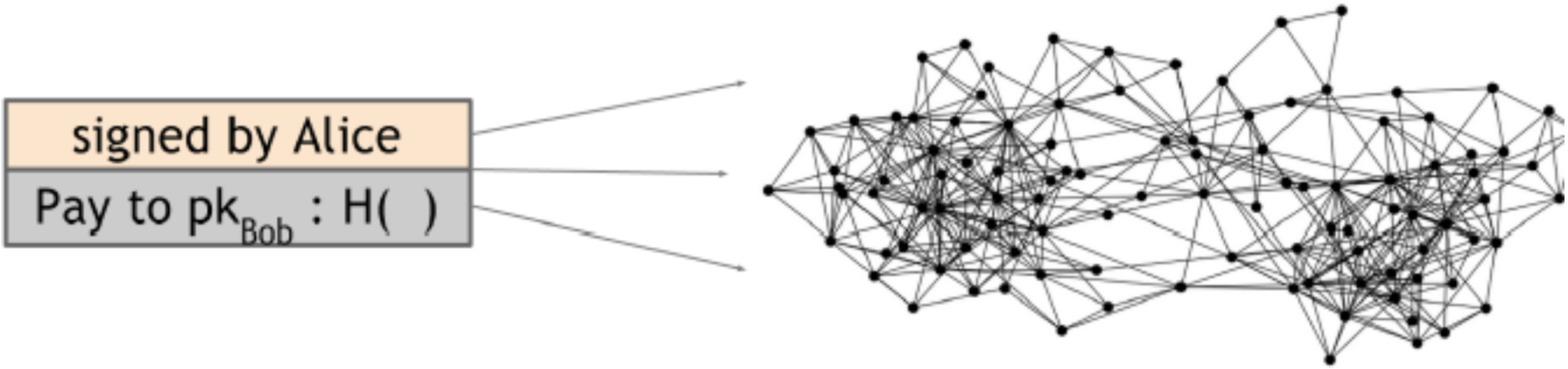
A distributed consensus protocol has the following two properties

- It must terminate with all honest nodes in agreement on the value
- The value must have been generated by an honest node

Distributed consensus has various applications, and it has been studied for decades in computer science. The traditional motivating application is reliability in distributed systems.

Distributed consensus in Bitcoin

Bitcoin is a peer-to-peer system: When Alice wants to pay Bob, she broadcasts a transaction to all Bitcoin nodes that comprise the peer-to-peer network.



Bob is one node of the network, but where is Bob?

Recall Bob public key is Bob's identity, nothing else is known

Which Consensus in Bitcoin network?

- Many users broadcast transactions; nodes must agree on exactly which transactions were broadcast and the order in which these transactions happened.
- Many ledgers: At any given point, all the nodes in the peer-to-peer network have a ledger consisting of a sequence of blocks, each containing a list of transactions, that they've reached consensus on.
- The protocol is able to keep consistency (with time delay). All nodes agree on a single, global ledger (=accounting book) for the system. As in ScroogeCoin, Bitcoin for optimization, groups transactions into blocks: consensus on a block-by-block basis.

Transactions and block

A **transaction** contains Alice's signature, an instruction to pay to Bob's public key, and a hash.

- The hash represents a pointer to a previous transaction output that Alice received and is now spending. The pointer references a transaction that was included in some previous block in the consensus chain.

Blocks (set of transactions) include a hash pointer to the previous block that they're extending. Extending a block takes a random time and significant computational effort

Ledger (set of blocks) contains all blocks with all Bitcoin transactions

How nodes reach consensus on a block

Block = set of transactions

- At regular intervals each node with a ledger proposes a new block to be added containing its own outstanding transaction set
- Then all nodes execute some consensus protocol, where each node's input is its own proposed block.
- Some nodes may be malicious and put invalid transactions into their blocks, but we might assume that other nodes will be honest.
- If the consensus protocol succeeds, a valid block will be selected as the output.

How nodes reach consensus on a block

One possible approach

- Each node proposes a block and then run a consensus protocol. If the consensus protocol succeeds, a valid block will be selected as the output
- Even if the selected block was proposed by only one node, it's a valid block (recall property of consensus protocol)
- There may be some other valid outstanding transactions that did not get included in the block, but this is not a problem. If some valid transaction didn't make it into this particular block, it could just wait and get into the next block

How nodes reach consensus on a block

The previous approach needs to be adapted; two types of obstacles:

- imperfections in the network, such as latency and nodes crashing
- Malicious nodes that deliberate try to subvert the process.

High latency implies that there is no notion of global time. Therefore not all nodes can agree to a common ordering of events simply based on observing timestamps.

Distributed consensus might be hard

Byzantine Generals Problem

- The Byzantine army is separated into divisions, each commanded by a general. The generals communicate by messenger in order to devise a joint plan of action
- Some generals may be traitors and may intentionally try to subvert the process so that the loyal generals cannot arrive at a unified plan

Goal: reach agreement among all loyal generals: they must agree on the same plan without the traitorous generals being able to cause them to adopt a bad plan.

It has been proven that this is impossible to achieve if one-third or more of the generals are traitors.

Consensus in Bitcoin

Bitcoin differs from traditional models for consensus

1. Introduces the idea of incentives (=money), which is novel for a distributed consensus protocol
2. Bitcoin's consensus algorithm uses randomization

Use of randomization implies that consensus is reached over a long period of time (no time guarantee)

- In about an hour most nodes reach consensus
- but even at the end of that time, nodes can't be certain that any particular transaction or a block has made it into the ledger
- as time goes on, the probability that your view of any block will match the eventual consensus view increases with times

Two difficulties

In Bitcoin there is no central authority to assign identities to participants and verify that they're not creating new nodes every 5 minutes

1. Sybil attack: Sybils are just copies of nodes that amalicious adversary can create to look like there are a lot of different participants, when in fact

all those pseudo-participants are really controlled by the same adversary.

2. **Anonimity** of all users is a goal of Bitcoin: nodes have no identities

Transactions and block

A **transaction** is data that contains Alice's signature, an instruction to pay to Bob's public key, and a hash.

- This hash represents a pointer to a previous transaction output that Alice received and is now spending.

  The pointer references a transaction that was included in some previous block in the consensus chain.

Blocks (set of transactions) include a hash pointer to the previous block that they're extending.

We say that a **transaction included in a block is confirmed**

Ledger: list of all blocks (organized as a Merkle tree)

NOTE: Adding a block to the ledgers takes a random time (e.g. 10 minutes) and significant computational effort

Bitcoin consensus algorithm (simplified)

Assume the ability to select a random node in a manner that is not vulnerable to Sybil attacks.

Consensus is reached as follows

1. New transactions are broadcast to all nodes
2. Each node collects new transactions into a block
3. In each round a random node gets to broadcast its block
4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures)
5. Nodes express their acceptance of the block by including its hash in the next block they create
6. If there are more possibilities the node chooses the longest chain

Bitcoin: stealing bitcoins

Stealing Bitcoins.

Can Alice simply steal Bitcoins belonging to another user at an address she doesn't control?

No. Even if it is Alice's turn to propose the next block in the chain, she cannot steal other users' bitcoins.

Doing so would require Alice to create a valid transaction that spends that coin: That is to forge the owners' signatures which she cannot do if a secure digital signature scheme is used.

Denial of service attack.

Assume Alice really dislikes some other user Bob.

- Alice can then decide that she will not include any transactions originating from Bob's address in any block that she proposes to get onto the block chain: she's denying service to Bob.
- While this is a valid attack that Alice can try but with minor effect: if Bob's transaction doesn't make it into the next block that Alice proposes, he will just wait until an honest node gets the chance to propose a block and then his transaction will get into that block.

Double spending

Alice may try to launch a double-spend attack.

- Assume that Alice is a customer of some online that must pay Bob for some service
- Alice creates a Bitcoin transaction from her address to Bob's and broadcasts it to the network.

We assume a majority of honest nodes and that some honest node creates the next block, and includes this transaction in that block

So there is now a block that was created by an honest node that contains a transaction that represents a payment from Alice to the merchant Bob.

Double spending

The latest block was generated by an honest node and includes a transaction in which Alice pays Bob.

- Upon seeing this transaction included in the block chain, Bob concludes that Alice has paid him.

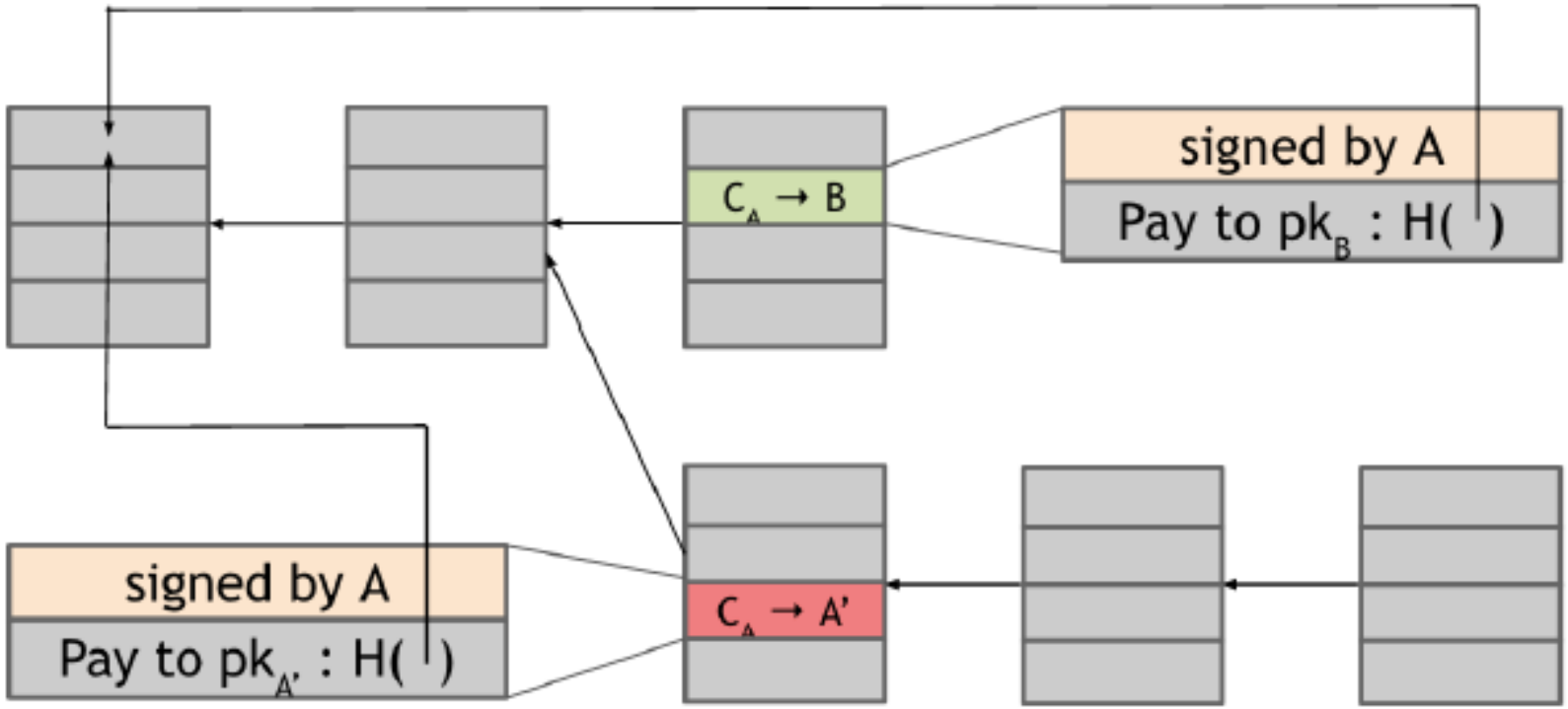
Suppose the next random node that is selected in the next round happens to be controlled by Alice

- Alice proposes a block that ignores the block that contains the payment to Bob and instead contains a pointer to the previous block and includes a transaction that transfers the coins that she was sending to Bob to a different address that she herself controls.

Double spend attempt is successful?

Alice double spends

- There are two blocks that contradict each other: the one with the Alice → Bob transaction and the one with the Alice → Alice transaction.
- They will not be pulished together by the same node. Which of the two block the network will reach consensus?
- Honest nodes follow the policy of extending the longest valid branch, so which branch will they extend? We do not know!!
- At the beginning the two branches have the same length (they only differ in the last block and both of these blocks are valid). The node that chooses the next block then may decide to build upon either one of them, and this choice will largely determine whether or not the double-spend succeeds.



Alice creates two transactions with same Bitcoin C_A: one to Bob and one to Alice; they spend the same Bitcoin, but only one of these transactions can be included in the block chain.
The arrows are pointers from one block to the previous block that it extends including a hash of that previous block within its own contents.

Double spending

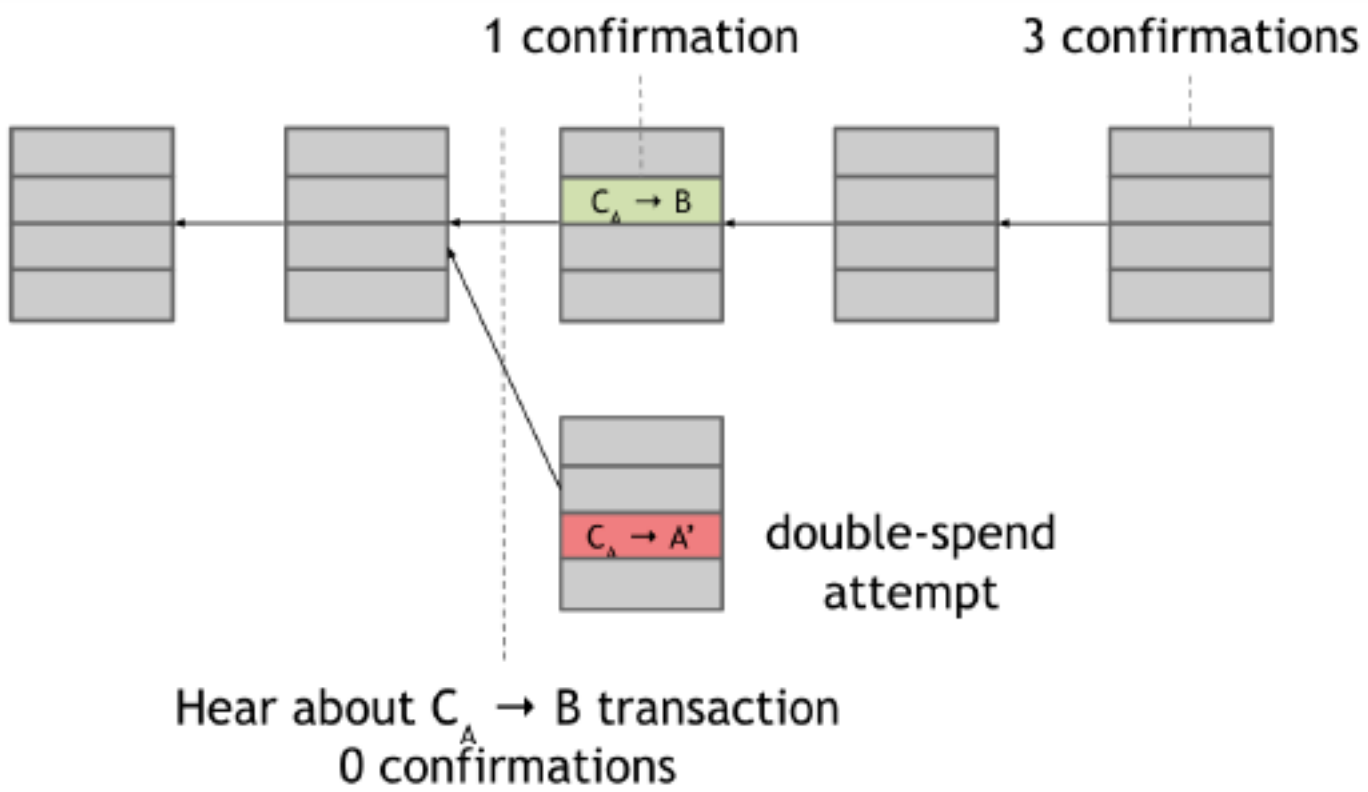
In a distributed system the two transactions look identical -we do not know which is the legitimate one

Nodes try to extend the transaction they are aware: **the accepted transaction is the one that will be the longest**; time to extend is random

The accepted transaction is decided within some time window: more time more certainty

The block containing the short transaction to Bob is ignored by the network, and this is now called an **orphan block**.

Double spending



- The number of confirmations provides confidence in the probabiity of the next accepted transaction (high confirmations more realiable)
- In fact, the double-spend probability decreases exponentially with the number of confirmations.

Hash puzzles

How a node computes a new block?

- the node that proposes a block is required to find a number, or nonce, such that the when you concatenate the nonce, the previous hash, and the list of transactions that comprise that block and take the hash of this whole string, then that hash output should be a number that falls into a target space that is quite small in relation to the the much larger output space of that hash function.
- Bitcoin defines such a target space as any value falling below a certain target value:

  H(nonce|| previous _hash||tx||tx||…||tx) < target

- If the hash function is good then this is computationally hard; hence this requires computation power and time.

Hash puzzles: 1. difficult to compute

Three important properties of hash puzzles. The first is that they need to be quite difficult to compute

- Difficulty varies with time: end of 2014, the difficulty level is about 10^{20} hashes per block. (i.e. the size of the target space is only $1/10^{20}$ of the size of the output space of the hash function). This is a lot of computation.
- Because of this, only some nodes even bother to compete in this block creation process. This process of repeatedly trying and solving these hash puzzles is known as Bitcoin mining, and we call the participating nodes miners.
- Even though technically anybody can be a miner, there's been a lot of concentration of power in the mining ecosystem due to the high cost of mining.

Hash puzzles: 2.parameterizable cost

The second property is that the cost to be parameterizable, not a fixed cost forever

- To do this the nodes in the Bitcoin network automatically recalculate the **target**, (as a fraction of the output space of the hash function[s]), every 2016 blocks.
- They recalculate the target in such a way that the average time between successive blocks produced in the Bitcoin network is about 10 minutes.
- With a 10-minute average time between blocks, 2016 blocks works out to two weeks. In other words, the recalculation of the target happens roughly every two week

Hash puzzles: 2.parameterizable cost

Why do we want to maintain 10-minute invariant?

- If blocks were to come very close together, then there would be a lot of inefficiency, and we would lose the optimization benefits of being able to put a lot of transactions in a single block. There is nothing magical about the number 10, and if you went down 5 minutes, it is ok

- Many attacks on Bitcoin are infeasible if the majority of miners are honest and follow the protocol
  - This is true because if a majority of miners, weighted by hash power, are honest, the competition for proposing the next block will automatically ensure that there is at least a 50 % chance that the next block to be proposed comes from an honest node.

Hash puzzles: 2.parameterizable cost

Solving hash puzzles is probabilistic: nobody can predict which nonce is going to result in solving the hash puzzle.

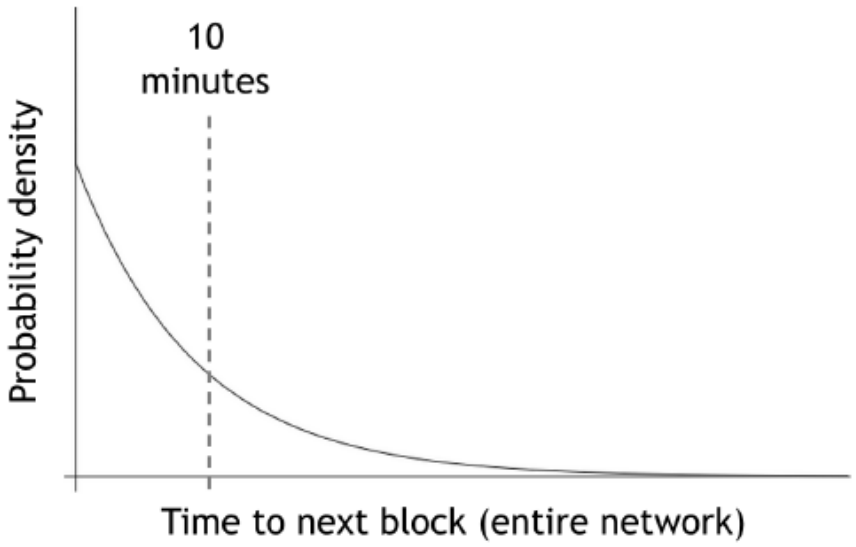
- Mathematically, this process is called Bernoulli trials.

How long it will take a miner to find a block?

Average time=

10 minutes/ fraction Hash power

A miner with 0.1 percent of the total network hash power, find a block once every 10,000 minutes, (about a week)



Hash puzzles: 3. trivial to verify

It is trivial to verify that a node has computed proof of work correctly.

- the nonce that makes the block hash fall below the target must be published as part of the block. It is thus trivial for any other node to verify that the output is less than the target.

This allows us to get rid of centralization. We don't need any centralized authority verifying that miners are doing their job correctly.

- Any node or any miner can instantly verify that a block found by another miner satisfies this proof-of-work property.

Block Reward for miners

The node that creates a block includes a special transaction in that block:

- A coin-creation transaction, (analogous to CreateCoins in Scroogecoin), and the node can choose the recipient address of this transact.
- This is the payment to the node in exchange for the service of creating that block to goes on the consensus chain
- Today: the value of the block reward is 25 Bitcoins. But it actually halves every 210000 blocks. Based on the rate of block creation this means that the rate drops roughly every four years. We're now in the second period
- Halving every 210000 blocks gives a geometric series, and you might know that it means that there is a finite sum. It works out to a total of 21 million bitcoins

Block Reward for miners

- Every ten minutes, one lucky Bitcoin miner earns a reward for extending the block chain by one block.
- In 2009, the reward was 50 BTC. Since May 2020 is 6.25 BTC. (See https://blockchain.info to issue queries about the block chain.); every 210000 blocks the reward halves
- **Mining is the only mechanism for creating new bitcoins.** The total number of Bitcoins will never exceed 21M.
- The rewarded miner also receives all (optional) transaction fees in the block.

Block Reward for miners

CLAIM: A node gets the block reward regardless of whether it proposes a valid block or behaves maliciously. So there is no incentive to be honest. **This is not true!**

- A node "collect" its reward only if the block in question ends up on the long-term consensus branch
- In fact, the coin-creation transaction will only be accepted by other nodes if it ends up on the consensus chain
- Comment: block reward is a subtle and very powerful trick: It incentivizes nodes to behave honestly

Transaction fees for miners

The creator of a transaction can make the total value of the transact. outputs less than the total value of its inputs.

- Whoever creates the block that first puts that transaction into the block chain gets the difference - a transaction fee. (if you're a node that's creating a block that contains, 200 transactions, then the sum of all those 200 transaction fees is paid to the address that you put into that block).
- The transaction fee is purely voluntary, but we expect, that as the block reward starts to run out, it will become more and more important, almost mandatory, for users to include transaction fees in order to get a reasonable quality of service.
- This is already starting to happen now. But it is unclear precisely how the system will evolve; it really depends on a lot of game theory which hasn't been fully worked out yet.

Reward for Miners: cost vs profit

- Cost for miners

  Hardware cost + electricity cost

- Miners have incentives if

  Cost < Transaction fee + Block reward

Reward for Miners: Proof of work

How Bitcoin solves the problem of honest behaviour by users?

- Miners have incentives to be honest (block reward and transaction fees)
- The next block is proposed by selecting a random node in proportion to a resource that we hope that nobody can monopolize (computational power)
- A different solution: the next block is chosen by a random node; then we do not have incentives to be honest
- NOTE: block reward will finish around 2040; will the end of Bitcoin? There is another incentive: transaction fees

Full node and light node

- Any computer that connects to the Bitcoin network is called a **node**.
- Nodes that fully verify all of the rules of Bitcoin are called **full nodes and download the blockchain**.
- A full node does not propose new blocks but stores and answers queries from other nodes
- A light Node does not download the complete blockchain.

Full node and light node

- Any computer that connects to the Bitcoin network is called a **node**.
- Nodes that fully verify all of the rules of Bitcoin are called **full nodes**.
- A full node does not propose new blocks but stores and answers queries from other nodes

Lightweight (or simply light) node

- A light Node does not download the complete blockchain. Instead, it downloads the block headers only to validate the authenticity of the transactions.
- Because of this reason light nodes are easy to maintain and run. Lightweight nodes use a method called Simplified payment verification (SPV) to verify transactions.
- Lightweight nodes are served by full nodes to connect to the Bitcoin network. They are effectively dependent on the full nodes to function.

Lightweight (or simply light) node

Lightweight node wallets have limitations.

- **Validation:** Lightweight wallets do not validate the rules of bitcoin. If somebody pays a lightweight wallet user with fake bitcoins, then the wallet will accept them and the user will be left out of pocket.
- **Privacy:** Lightweight wallets typically send addresses to a trusted third party and receive wallet balance and history. This allows the trusted third party to spy on all the users past and future transactions.

These downsides can all be avoided by configuring a lightweight node wallet to connect only to your own full node.

What makes a full node?

Full nodes download every block and transaction and check them against Bitcoin's consensus rules.

Examples of consensus rules, (there are many more):

- Blocks may only create a certain number of BTC (now 6.25)
- Transactions must have correct signatures for the bitcoins being spent.
- Transactions/blocks must be in the correct data format.
- Within a single block chain, a transaction output cannot be double-spent.

If a transaction or block violates the consensus rules, then it is rejected, even if every other node on the network thinks that it is valid.

What makes a full node?

Full nodes download every block and transaction and check them against Bitcoin's consensus rules.

Examples of consensus rules, (there are many more):

- Blocks may only create a certain number of BTC (now 6.25)
- Transactions must have correct signatures for the bitcoins being spent.
- Transactions/blocks must be in the correct data format.
- Within a single block chain, a transaction output cannot be double-spent.

If a transaction or block violates the consensus rules, then it is rejected, even if every other node on the network thinks that it is valid.

What makes a full node?

- **Security**: Running a full node is the only way you can use Bitcoin in a trustless way. You will know for sure that all the rules of Bitcoin are being followed, (e.g. no bitcoins are spent not belonging to the owner, no coins were spent twice, etc.)
- **Privacy**: Full nodes are currently the most private way to use Bitcoin, with nobody else learning which bitcoin addresses belong to you.
- The most popular software implementation of full nodes is called Bitcoin Core

Full node: Economic strenght

Full nodes enforce the consensus rules no matter what.

- Lightweight nodes do not do this: they do whatever the majority of mining power says. Therefore, if most of the miners got together to increase their block reward, for example, lightweight nodes would blindly go along with it.
- If this ever happened, the network would split such that lightweight nodes and full nodes would end up on separate networks, using separate currencies: People using lightweight nodes would be unable to transact with people using full nodes.
- In practice, miners are unlikely to attempt anything like the above scenario as long as full nodes are prevalent because the miners would lose a lot of money: if this happens bitcoin would loose valure.

Full node: Economic strenght

- In practice, miners are unlikely to attempt anything like the above scenario as long as full nodes are prevalent because the miners would lose a lot of money.
- But the incentives completely change if everyone uses lightweight nodes. In that case, miners definitely *do* have an incentive to change Bitcoin's rules in their favor.
- Therefore, it is critical for Bitcoin's survival that the great majority of the Bitcoin economy be backed by full nodes, not lightweight nodes.
- To contribute to Bitcoin's economic strength, you must actually use a full node for your real transactions (or use a lightweight node connected to a full node that you personally control). Just running a full node on a server somewhere does not contribute to Bitcoin's economic strength.

Full node and light node: conclusions

- **Economic strength:** it is critical for Bitcoin's survival that the great majority of the Bitcoin economy be backed by full nodes, not lightweight nodes.
- **Full node:** better for privacy, security; bad for running costs (keep the blockchain)
- **Light node:** worse for privacy, security; easy to realize (you use other nodes); should trust a full node for verifying

Conclusions

Protection against invalid transactions is based on cryptography and it is enforced by consensus

- if a node does attempt to include a cryptographically invalid transaction, then the transaction won't end up in the long-term consensus chain is because a majority of the nodes are honest and won't include an invalid transaction in the block chain.

Protection against double-spending is by consensus (no crypto): two transactions that represent a double-spending attempts are both valid from a cryptographic perspective; consensus determines which one will end up on the long-term consensus chain.

- you're never 100 percent sure that a transaction you're interested in is on consensus branch.
- the exponential probability guarantee is rather good. After about six transactions, there's virtually no chance that you're going to go wrong

Consensus is based on incentives for miners and on services of full nodes