(Brief) Discussion

1. What is the double spending problem?
2. What is a byzantine fault tolerant network?
3. Describe proof of work

Blockchain at Michigan

W3: Bitcoin

2022

Previously at BAM...

Double spending problem is a significant problem in **digital, decentralized** systems.

This is the **main motivation** behind Bitcoin.

Blockchains solve this problem using robust **consensus mechanisms**.

In particular, Bitcoin uses **proof of work**.

Proof of Work (Recap)

Summary

We force nodes to **work** to contribute to the blockchain. This disincentivizes evil actors.

In particular, this work involves **hashing** a block repeatedly, changing the **nonce** each time, until a particular hash is found.

Trying to find this nonce is called **mining**.

Today’s Goals

To be able to answer:

1. What does it mean to “own” a bitcoin?
2. What exactly is stored in a block?
3. How exactly are blocks added to the chain?

To be able to read the Bitcoin whitepaper!

(spoilers: this is the reading this session)

Warning!

This is a **dense** session.
Please ask questions.
Go through these slides again later.

“Owning” a bitcoin

What are you actually buying when you buy 1 BTC?

"Owning" a bitcoin

What are you actually buying when you buy 1 BTC?

Bitcoin is purely digital.

"Owning" 1 BTC means that there **exists a transaction** on the distributed ledger (blockchain) that states that **1 BTC was transferred** from *address A* to *address B*.

This assumes you have the **private key** for *address B*

“Owning” a bitcoin

Technically speaking, **nobody “owns” anything**.

Knowing the **private key** of an address means that you can do stuff with the BTC in that address.

If you lose the private key, the BTC in that address is lost forever.

What are you actually buying when you buy 1 BTC?

Bitcoin is purely digital.

“Owning” 1 BTC means that there **exists a transaction** on the distributed ledger (blockchain) that states that **1 BTC was transferred** from *address A* to *address B*.

This assumes you have the **private key** for *address B*

What Are Keys?

But first, some cryptography basics.

Cryptography

Encryption

Encrypt: plaintext → cyphertext

Decrypt: cyphertext → plaintext

Hashing is a method of encryption.

Q: Is hashing a “good” method of encryption?

Cryptography

Yes and no.

SHA 256 is an excellent asymmetric algorithm.

SHA 256 is a terrible symmetric algorithm.

Note: symmetric hashing algorithms exist (more on this in a bit...)

Encryption

Encrypt: plaintext → cyphertext

Decrypt: cyphertext → plaintext

Hashing is a method of encryption.

Q: Is hashing a “good” method of encryption?

Cryptography

Caesar Cipher

Plaintext: This is a sentence

Key = 0: This is a sentence

Key = 1: Uijt jt b tfoufodf

Key = 2: Vjku ku c ugpvgpeg

Key = 13: Guvf vf n fragrapr

Cryptography

Is this a good cipher?

Caesar Cipher

Plaintext: This is a sentence

Key = 0: This is a sentence

Key = 1: Uijt jt b tfoufodf

Key = 2: Vjku ku c ugpvgpeg

Key = 13: Guvf vf n fragrapr

Cryptography

Is this a good cipher?

NO

Caesar Cipher

Plaintext: This is a sentence

Key = 0: This is a sentence

Key = 1: Uijt jt b tfoufodf

Key = 2: Vjku ku c ugpvgpeg

Key = 13: Guvf vf n fragrapr

Cryptography

How do we go from
“Guvf vf n fragrapr”
to
“This is a sentence”?

Encryption

Algorithm + key = ciphertext

Hash + nonce = encrypted block

How do we **decrypt** ciphertext?

Cryptography

How do we go from
“Guvf vf n fragrapr”
to
“This is a sentence”?

Encryption

Algorithm + key = ciphertext

Hash + nonce = encrypted block

How do we **decrypt** ciphertext?

We need the **key**!

Cryptography

encrypt("This is a sentence", key=13) =
"Guvf vf n fragrapr"

decrypt("Guvf vf n fragrapr", key=13) =
"This is a sentence"

How do we go from
"Guvf vf n fragrapr"
to
"This is a sentence"?

Encryption

Algorithm + key = ciphertext

Hash + nonce = encrypted block

How do we **decrypt** ciphertext?

We need the **key**!

Cryptography

What if we could have **two keys**?

One only for **en**cryption and

one only for **de**cryption?

Cryptography

What if we could have **two keys**?

“**Public**” key ⟶ One only for **en**cryption and

“**Private**” key ⟶ one only for **de**cryption?

This is **symmetric** encryption

Asymmetric	= one way
Symmetric	= both ways

Cryptography

What if we could have **two keys**?

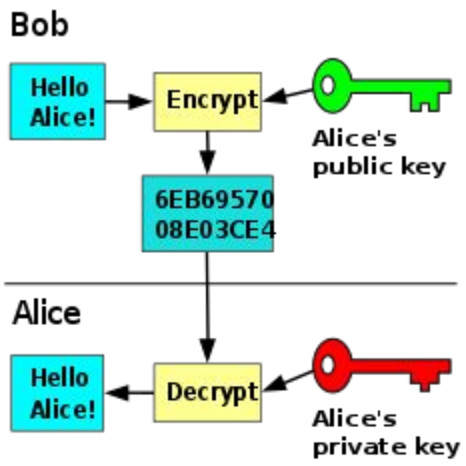
"**Public**" key → One only for **en**cryption and

"**Private**" key → one only for **de**cryption?

This is **symmetric** encryption

This is done all the time on the Internet, another network where security is important

(extra info on public/private keys)



(Reference)

Questions?

Ownership

Claim: private key = ownership

But first:

ownership = ability to send BTC that is in a given address

Ownership

Claim: private key = ownership

But first:

ownership = ability to send BTC that is in a given address

Private key gives you **control over an address** that has BTC stored in it. You could say you “own” the address, rather than the BTC.

Ownership

Q: What if someone guesses your private key?

Claim: private key = ownership

But first:

ownership = ability to send BTC that is in a given address

Private key gives you **control over an address** that has BTC stored in it. You could say you "own" the address, rather than the BTC.

Ownership

Q: What if someone guesses your private key?

A: Then they own the BTC in the address as much as you do.

But this is **extremely** unlikely…

Claim: private key = ownership

But first:

ownership = ability to send BTC that is in a given address

Private key gives you **control over an address** that has BTC stored in it. You could say you "own" the address, which stores some BTC.

Quick Facts

SHA 256 was developed by the NSA.

Bitcoin uses SHA 256 (part of the SHA2 family).

Ethereum uses SHA3.

SHA3 was released in 2015.

SHA 256 produces an output of **256 bits**.

256 bits can represent

2^256 ~= 10^76 numbers.

There are “between 10^78 to 10^82” atoms in the universe.

Chances of guessing your private key are similar to finding a specific atom you own in the entire universe.

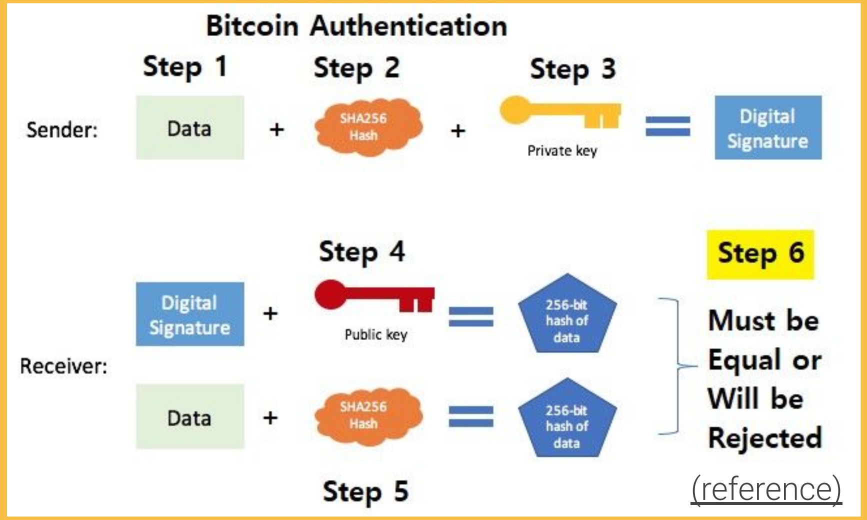
Ownership

So a **private key** gives you access to an address.

An **address** “stores” BTC, which just means that there is a transaction somewhere on the blockchain that states that some BTC was sent to that address.

So how do you send money in that address? And why do keys matter?

Ownership

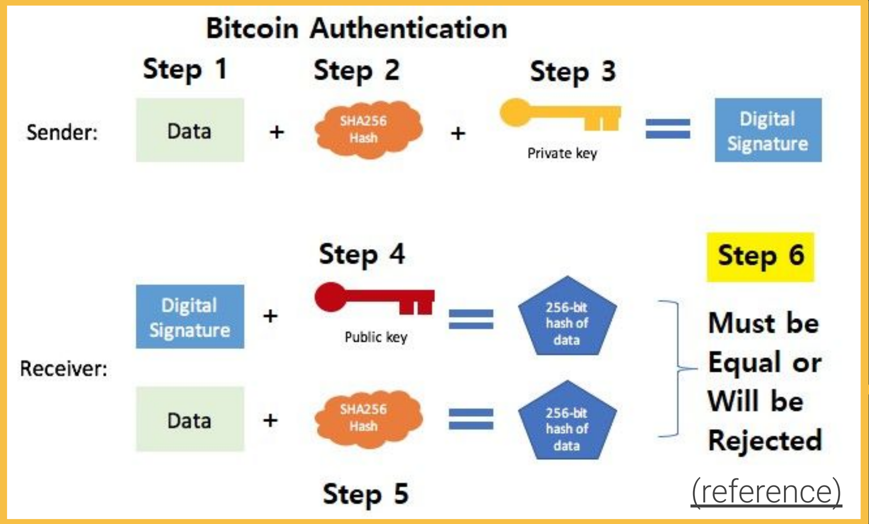


So a **private key** gives you access to an address.

An **address** "stores" BTC, which just means that there is a transaction somewhere on the blockchain that states that some BTC was sent to that address.

So how do you send money in that address? And why do keys matter?

Ownership

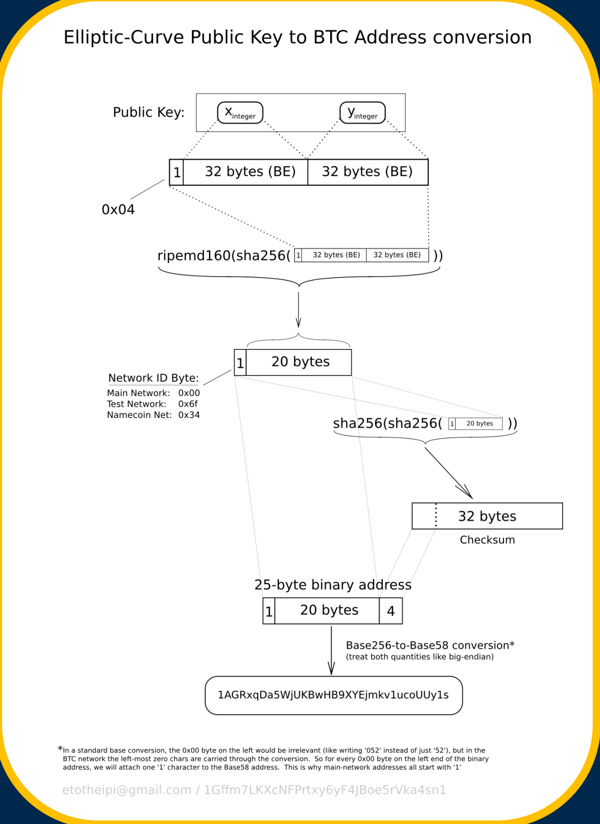


The sender and receiver must agree for a transaction to be "broadcast" (more on this in a bit).

The receiver can verify the sender using the **sender's public key**, because **only the sender** could have created that **digital signature**.

Questions?

Wallets



(Bitcoin specific)

A **wallet** is a collection of private keys.

Specifically, **100 private keys**.

Each private key generates a public key.

Each public key generates an address.

Generated using **asymmetric** encryption.

(source + more info)

(further reading)

Wallets

You have a private key to access a wallet (which private keys to access an address).

Think of the wallet key as a “masterkey”.

Creating a wallet **generates 100 random** private keys, from which the public keys and addresses are calculated (using math and algos).

Wallets

You have a private key to access a wallet (which private keys to access an address).

Think of the wallet key as a “masterkey”.

Creating a wallet **generates 100 random** private keys, from which the public keys and addresses are calculated (using math and algos).

Q: What if a randomly generated key conflicts with an existing key?

Wallets

Collisions:

*"Since Bitcoin addresses are basically random numbers, **it is possible, although extremely unlikely**, for two people to independently generate the same address. This is called a collision. **If this happens, then both the original owner of the address and the colliding owner could spend money sent to that address.** It would not be possible for the colliding person to spend the original owner's entire wallet (or vice versa).*

*But because the space of possible addresses is so astronomically large **it is more likely that the Earth is destroyed in the next 5 seconds, than that a collision occur in the next millenium.**" - reference*

You have a private key to access a wallet (which private keys to access an address).

Think of the wallet key as a "masterkey".

Creating a wallet **generates 100 random** private keys, from which the public keys and addresses are calculated (using math and algos).

Q: What if a randomly generated key conflicts with an existing key?

Wallets

You have a private key to access a wallet (which private keys to access an address).

Q: Why do you need 100 addresses (or more)?

Wallets

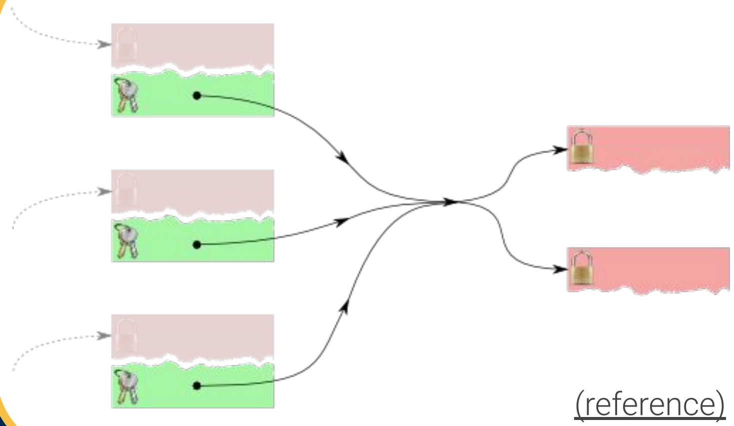
You have a private key to access a wallet (which private keys to access an address).

Q: Why do you need 100 addresses (or more)?

Bitcoin **never reuses an address**.

- Makes you virtually untraceable

Wallets



You cannot “partially” use the funds in a given address.

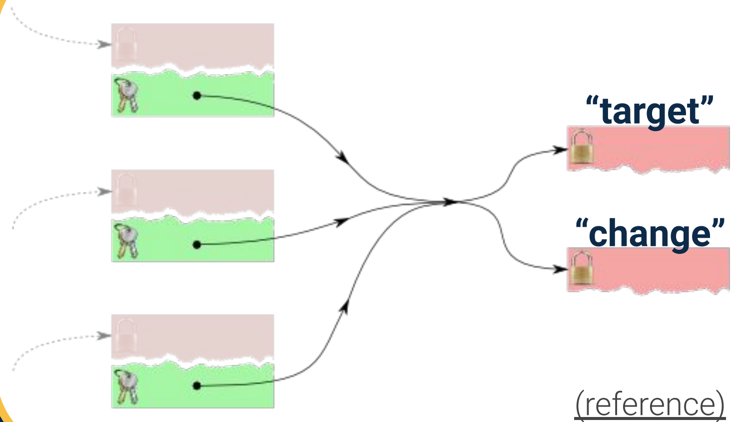
You have a private key to access a wallet (which private keys to access an address).

Q: Why do you need 100 addresses (or more)?

Bitcoin **never reuses an address**.

- Makes you virtually untraceable

Wallets



(reference)

You cannot “partially” use the funds in a given address.

You have a private key to access a wallet (which private keys to access an address).

Q: Why do you need 100 addresses (or more)?

Bitcoin **never reuses an address**.

- Makes you virtually untraceable

(reference)

Transactions

How does "sending BTC" work?

Remember: **sending BTC** means adding a "**transaction**" on a **block** on the **blockchain** that says that BTC was sent to a certain address.

Transactions

How does "sending BTC" work?

Big picture:

1. Transaction is **broadcast** to miners.

2. Miners **add (or don't add)** the transaction to the block they are mining.

3. First miner to find proof of work for their block **broadcasts** block to nodes.

4. Nodes **confirm** whether block is valid (transactions are not double spent).

(Continued)

5. Miners show approval by **choosing** to extend the chain from this new block (or show disapproval by forking off).

Transactions

In Satoshi's words:

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Questions?
/quick break

Validation

How does "sending BTC" work?

Big picture:

1. Sender **broadcasts** transaction to miners.

2. Miners **add (or don't add)** the transaction to the block they are mining.

3. First miner to find proof of work for their block **broadcasts** block to nodes.

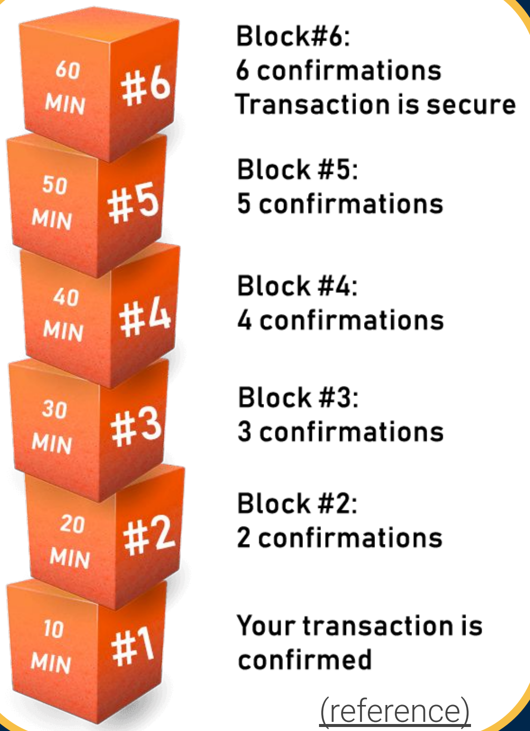
4. Nodes **confirm** whether block is valid (transactions are not double spent).

(Continued)

5. Miners show approval by **choosing** to extend the chain from this new block (or show disapproval by forking off).

How???

Validation



(reference)

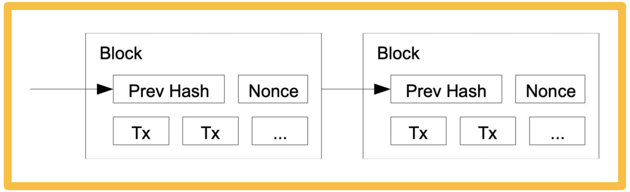
Validation vs confirmation?

Valid: transaction contains no double spending

Confirmed: enough blocks have been chained to the block containing the transaction such that it is now in the longest chain

Confirmation

Note: blocks are **chained** and **immutable**.



Confirmation

p = probability an honest node finds the next block
q = probability the attacker finds the next block
q_z = probability the attacker will ever catch up from z blocks behind

$$q_z = \begin{Bmatrix} 1 & \text{if } p \le q \\ (q/p)^z & \text{if } p > q \end{Bmatrix}$$

To get the probability the attacker could still catch up now, we multiply the Poisson density for each amount of progress he could have made by the probability he could catch up from that point:

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{Bmatrix} (q/p)^{(z-k)} & \text{if } k \le z \\ 1 & \text{if } k > z \end{Bmatrix}$$

Rearranging to avoid summing the infinite tail of the distribution...

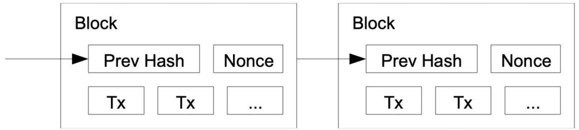
$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

Gambler's Ruin Problem

Note: blocks are **chained** and **immutable**.

Running some results, we can see the probability drop off exponentially with z.

```
q=0.1
z=0     P=1.0000000
z=1     P=0.2045873
z=2     P=0.0509779
z=3     P=0.0131722
z=4     P=0.0034552
z=5     P=0.0009137
z=6     P=0.0002428
z=7     P=0.0000647
z=8     P=0.0000173
z=9     P=0.0000046
z=10    P=0.0000012
```




```
q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
z=45    P=0.0000024
z=50    P=0.0000006
```

← This is the main point!

Confirmation

Question from last session:

What if two blocks are created simultaneously?

Nodes always consider the longest chain to be the correct one and will keep working on extending it. If two nodes broadcast different versions of the next block simultaneously, some nodes may receive one or the other first. In that case, they work on the first one they received, but save the other branch in case it becomes longer. The tie will be broken when the next proof-of-work is found and one branch becomes longer; the nodes that were working on the other branch will then switch to the longer one.

Validation

So how can we validate a block?

Validation

So how can we validate a block?

Intuitive answer: Go through the **entire blockchain** to check that the BTC is legitimate.

This works, but it expensive (storage and speed)!

Full nodes do this.

- Store all transactions in a given block
- Hash entire block to compare with "prev hash" of next block

Validation

> **The problem**
>
> Suppose I have a Bitcoin mobile wallet; my wallet tells peers only to broadcast transactions for my wallet address (known as *bloom filtering*). A peer broadcasts a transaction to me claiming that it is within a particular block. The block pointer may be perfectly valid, but the transaction could have been tampered with. In order to ensure the transaction is valid for that block I need to validate it, and I could do this by repeatedly hashing each transaction in the block (including the hash of the transaction before it) and check that the result matches the block header. Producing a block pointer to validate the transaction requires me to be sent all of the other transactions in that block so I can generate this hash value. It is computationally infeasible to produce the correct hash from a different set of transactions, so this ensures that the transaction is, indeed, in the block (other steps will check to ensure the block itself is valid).
>
> (reference)

So how can we validate a block?

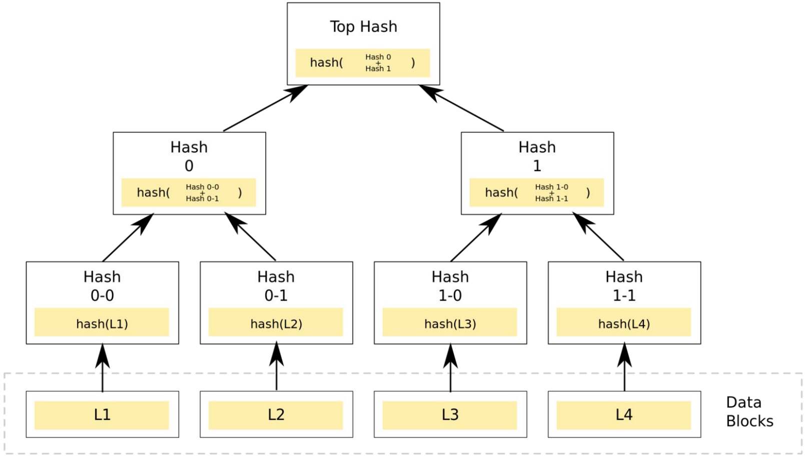
Intuitive answer: Go through the **entire blockchain** to check that the BTC is legitimate.

This works, but is expensive (storage and speed)

Full nodes do this.

- Store all transactions in a given block
- Hash entire block to compare with “prev hash” of next block

Validation



[(reference)]

Merkle Trees

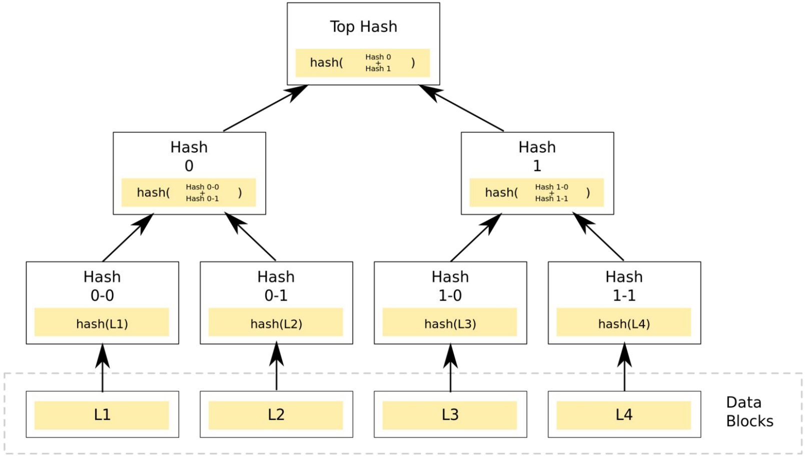
A Certified Digital Signature, Ralph Merkle (1989)

Tree: Data structure made of nodes and edges

Merkle Tree: A type of tree that uses hashes to "summarize" large amounts of data efficiently

* Also used in Ethereum!

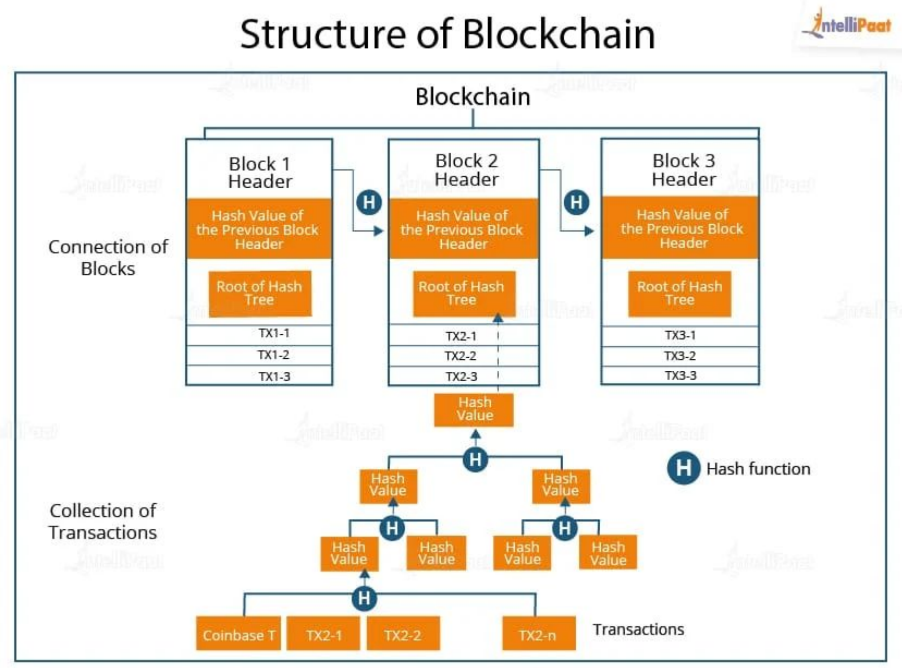
Validation



(reference)

But what really is a block?

Block = header + list of transactions



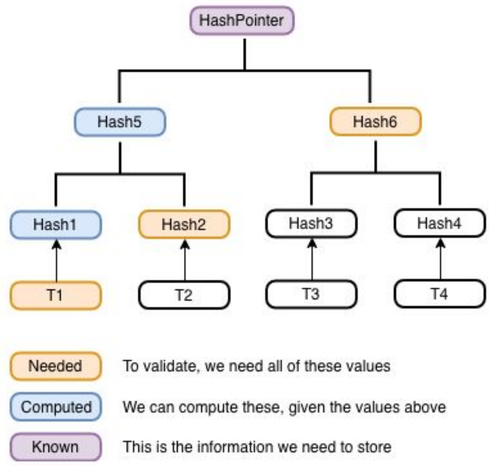
Validation

The problem

Suppose I have a Bitcoin mobile wallet; my wallet tells peers only to broadcast transactions for my wallet address (known as *bloom filtering*). A peer broadcasts a transaction to me claiming that it is within a particular block. The block pointer may be perfectly valid, but the transaction could have been tampered with. In order to ensure the transaction is valid for that block I need to validate it, and I could do this by repeatedly hashing each transaction in the block (including the hash of the transaction before it) and check that the result matches the block header. Producing a block pointer to validate the transaction requires me to be sent all of the other transactions in that block so I can generate this hash value. It is computationally infeasible to produce the correct hash from a different set of transactions, so this ensures that the transaction is, indeed, in the block (other steps will check to ensure the block itself is valid).

(reference)

we wished to validate. With a Merkle tree the only information we need is the hash of the sibling of the transaction, and the hash of each sibling up the branch to the hash pointer. We need only recompute this branch to produce a hash pointer than matches the one we have stored, not the entire tree. This means we no longer need all of those other transactions.

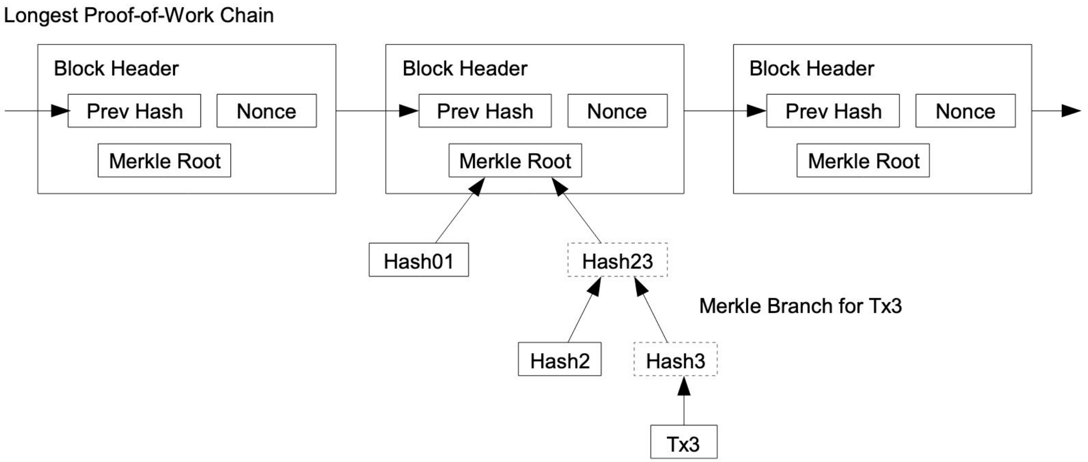


(reference)

Validation

8. Simplified Payment Verification

It is possible to verify payments without running a full network node. A user only needs to keep a copy of the block headers of the longest proof-of-work chain, which he can get by querying network nodes until he's convinced he has the longest chain, and obtain the Merkle branch linking the transaction to the block it's timestamped in. He can't check the transaction for himself, but by linking it to a place in the chain, he can see that a network node has accepted it, and blocks added after it further confirm the network has accepted it.



Full Node: stores entire blockchain

Light Node: checks merkle root for validation

Questions?

Block Size

Q: How many transactions in one block?

Block (1 MB):

- Header (80B)
- List of transactions (999.92 kB)

Header (80B):

- Bitcoin version number (4B)
- Previous block hash (32B)
- Merkle root (32B)
- Timestamp (4B)
- Difficulty (4B)
- Nonce (4B)

Block Size

Q: How many transactions in one block?

A: 1500~3000 (reference)

Visa handles around 1700 tps, which means 1.02 million every 10 mins.

Block (1 MB):

- Header (80B)
- List of transactions (999.92 kB)

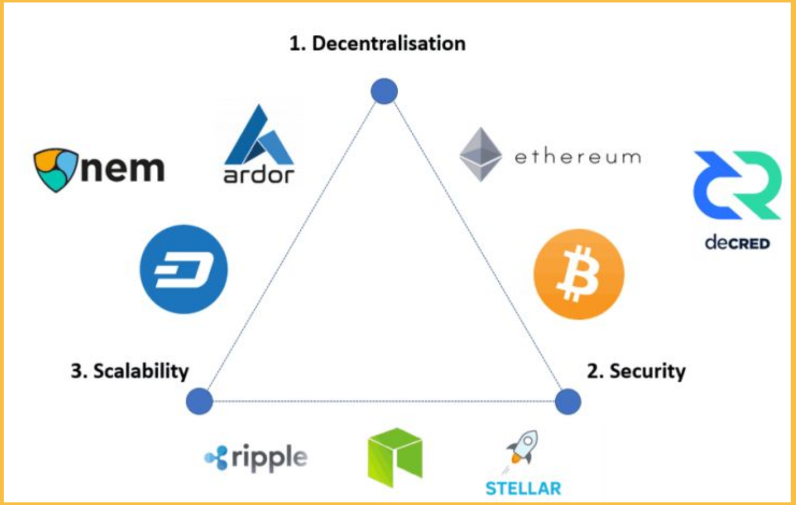
Header (80B):

- Bitcoin version number (4B)
- Previous block hash (32B)
- Merkle root (32B)
- Timestamp (4B)
- Difficulty (4B)
- Nonce (4B)

Blockchain Trilemma

Larger = better?

Blockchain Trilemma



Larger = better? Maybe.

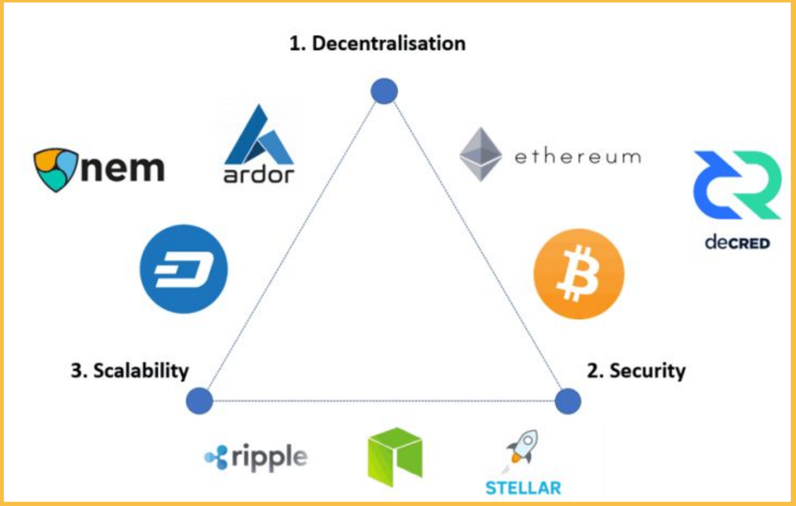
Pros:

- More transactions in one block = faster
- Lower transaction fees (higher "supply" of transactions per block)

Cons:

- Less decentralized
  - Larger block → more hash power needed to solve proof of work

Blockchain Trilemma



Big debate ⇒ Hard fork to **Bitcoin Cash**

Larger = better? Maybe.

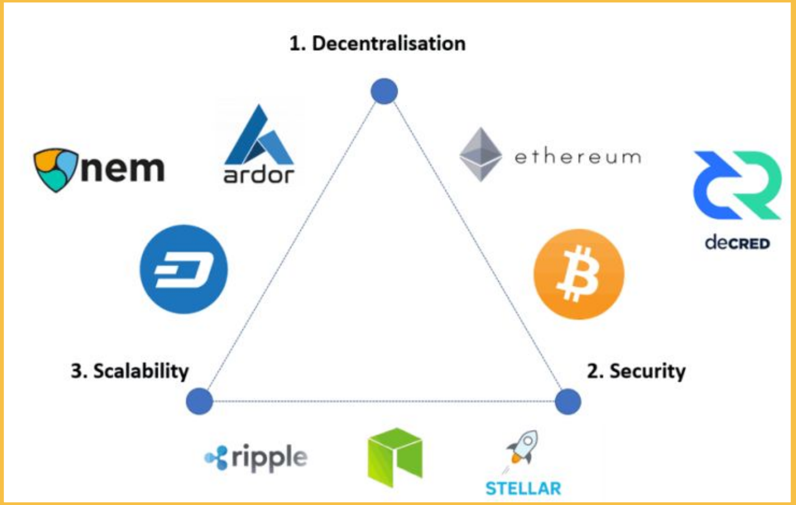
Pros:

- More transactions in one block = faster
- Lower transaction fees (higher “supply” of transactions per block)

Cons:

- Less decentralized
  - Larger block → more hash power needed to solve proof of work

Blockchain Trilemma



Changing block size is an **on chain** solution.

Contrasts with **off chain** and **layer 2** solutions.

Main idea: resolve transactions “outside” the main blockchain.

For the curiously inclined.

More on this in a later session…

Almost there...

Summary

Satoshi's last words… (in the whitepaper)

12. Conclusion

We have proposed a system for electronic transactions without relying on trust. We started with the usual framework of coins made from digital signatures, which provides strong control of ownership, but is incomplete without a way to prevent double-spending. To solve this, we proposed a peer-to-peer network using proof-of-work to record a public history of transactions that quickly becomes computationally impractical for an attacker to change if honest nodes control a majority of CPU power. The network is robust in its unstructured simplicity. Nodes work all at once with little coordination. They do not need to be identified, since messages are not routed to any particular place and only need to be delivered on a best effort basis. Nodes can leave and rejoin the network at will, accepting the proof-of-work chain as proof of what happened while they were gone. They vote with their CPU power, expressing their acceptance of valid blocks by working on extending them and rejecting invalid blocks by refusing to work on them. Any needed rules and incentives can be enforced with this consensus mechanism.

Quick Shoutout

Bitcoin was **not** the first attempt.

eCash, David Chaum (1982)

HashCash, Adam Back (1997)

B-Money, Wei Dai (1998)

Reusable Proof of Work, Hal Finney (2004)

Bitcoin, Satoshi Nakamoto (2008)

Reading:

1. Bitcoin Whitepaper, by Satoshi Nakamoto

*There will be a discussion next session

PS: Enjoy your spring break and good luck for midterms :)