Beginnings in Blockchain

DevCon 2021

Andrew Nash
www.linkedin.com/in/a-nash

UCC Data and Analytics Society

https://discord.com/invite/EEBBatKw6n

From whence has it come? Why?

- Consider the conventional methods for conducting any transaction between parties - monetary, trade, donation
- One person/group gives something to another, and may or may not get something in return
- You trust that the government doesn’t suddenly devalue the currency
- You trust that the seller, or your bank doesn’t erase all record of the sale, after it has been made
- You trust that you can’t spend money that doesn’t exist
- Etc
- But, if there is no central authority, arbitrator, or enforcer to keep people trustworthy, what will happen?

The Solution

- Enter blockchain
- A *decentralised transaction ledger*
- A network of machines, that store and track transactions, in an immutable fashion - once on the blockchain, neither incompetent nor malicious actors can erase records
- This needs to be extremely robust - and, applied correctly, can be more secure and authoritative than conventional centralised ledgers
- The fundamental building block on which blockchain is built, is the *cryptographic hash*

What is a hash?

Examples

MD5

SHA-1

SHA-2

Approved, and likely in to become widespread in a few years…..

SHA-3

A (speedy) run through of how SHA-1/2 works

1. Take the binary data to be hashed, append a 1 followed by 0s, until its length mod 512 is 448
2. Add a 64 bit integer to the end of the data, representing the length of the original data (note 448+64 = 512)
3. There are 8, 32 bit hard-coded constants that form part of the algorithm (derived from the fractional parts of prime roots)
    1. `6a09e667`
    2. `bb67ae85`
    3. `3c6ef372`
    4. `a54ff53a`
    5. `510e527f`
    6. `9b05688c`
    7. `1f83d9ab`
    8. `5be0cd19`

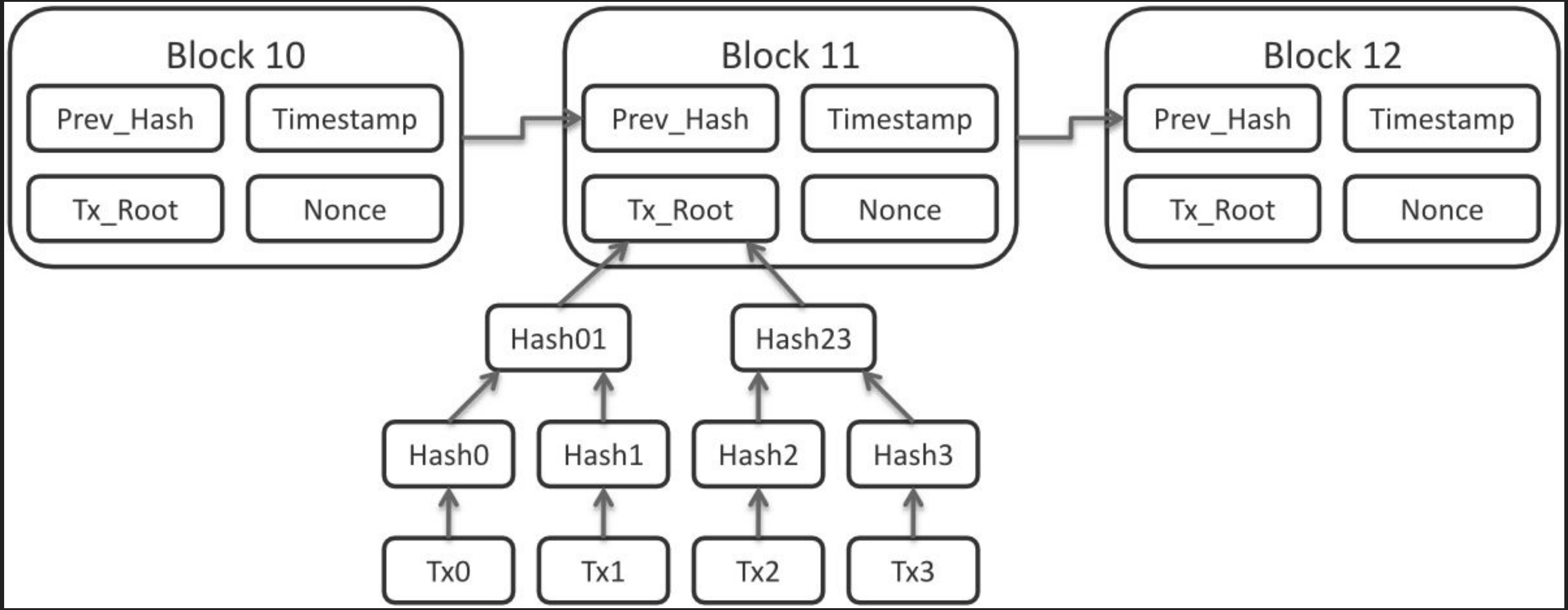
- There are 64 more of these (derived from prime cube roots)
- The padded data is split into 512 bit chunks
- Each of these is split into 16, 32-bit, words
- For each chunk, do the following numerous times
    - Add 48 more 16-bit words, where each new word i is a sum (mod 2**37) of
        - The XOR of all of the i - 15th word rotated 7 and 18 bits right, and shifted 3 bits right
        - The XOR of all of the i - 2th word rotated 17 and 19 bits right, and shifted 10 bits right
        - The i -16th word
        - The i-7th word
- Perform compression on this 64 word block
    - This uses the initial 8 values we defined as well as 64 values above
    - A similar process of rotations, shifting, XOR and addition is used
    - 8 new hash values are obtained, which are added to the 8

When this is all completed, the 8 has values are concatenated - the hash

Why does this matter?

- The fundamental building block on which a blockchain is based
- Will eventually be replaced by SHA-3 - of critical importance as computational capabilities increase

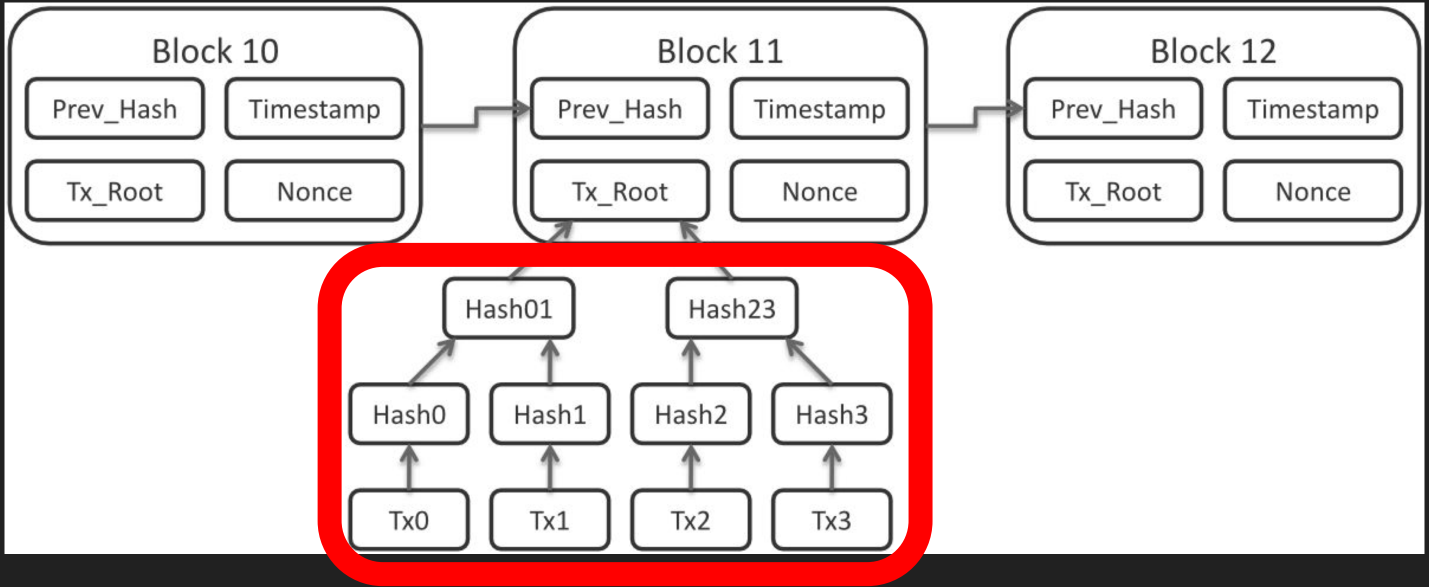
Blockchain - a chain of blocks



Lots Going on here!

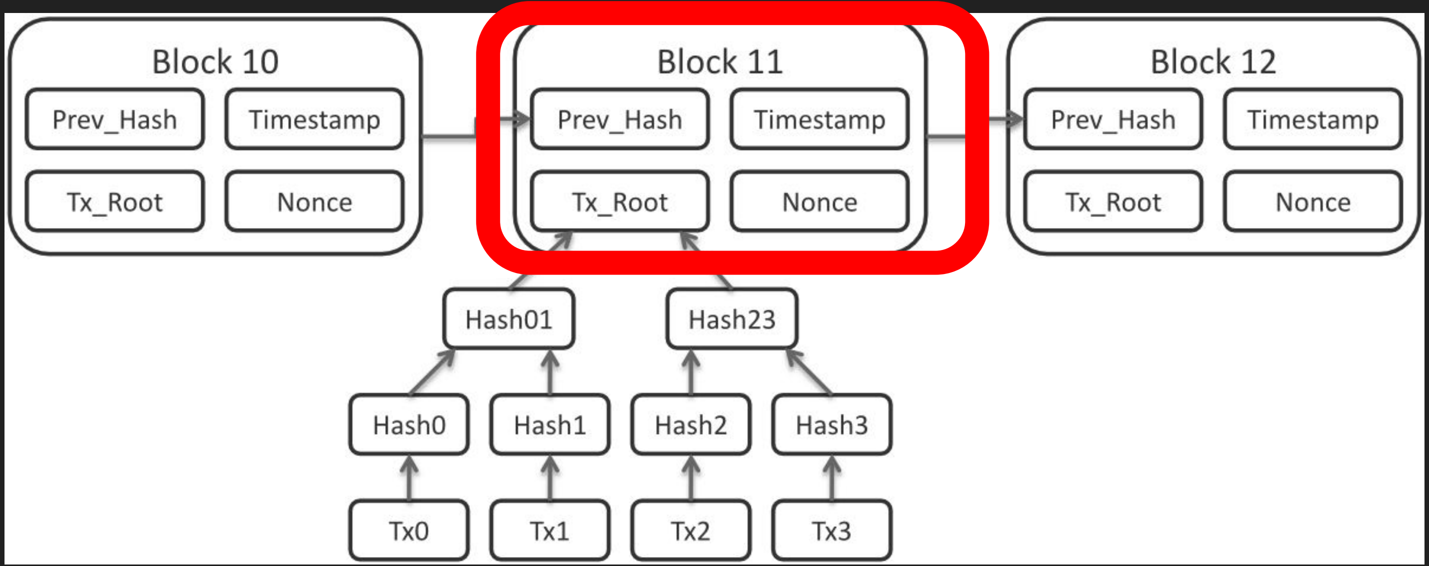
Let’s break it down

1. Obviously a block must contain transactions



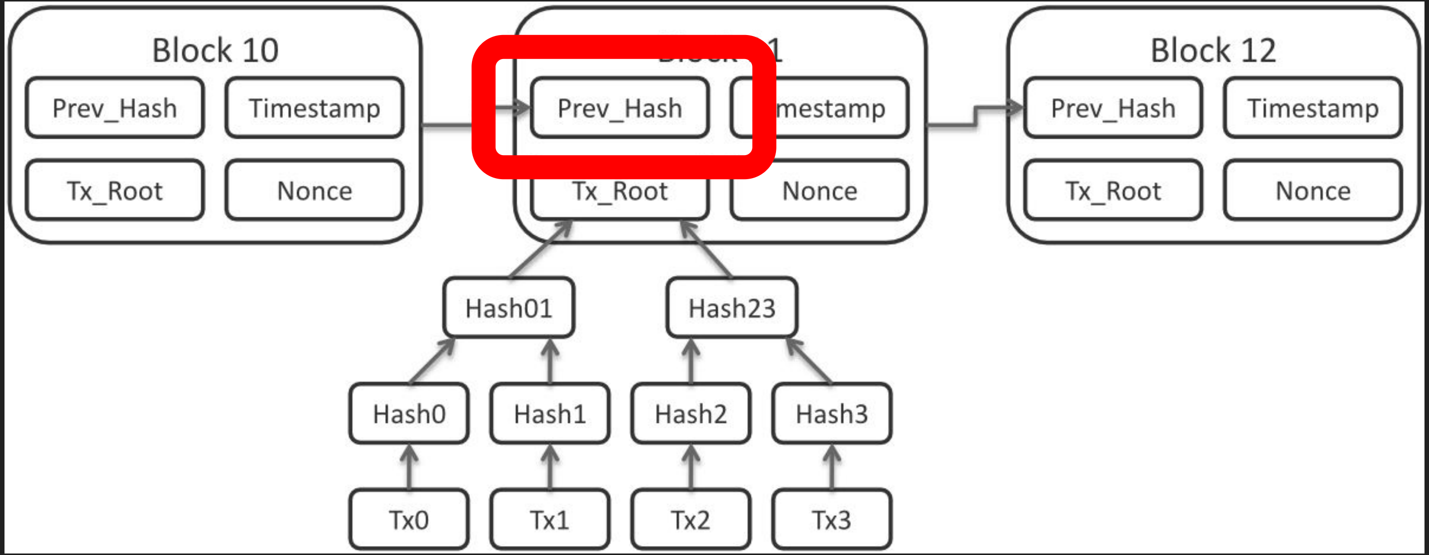
Lots Going on here!

2. Associated with each block of transactions, is a block header



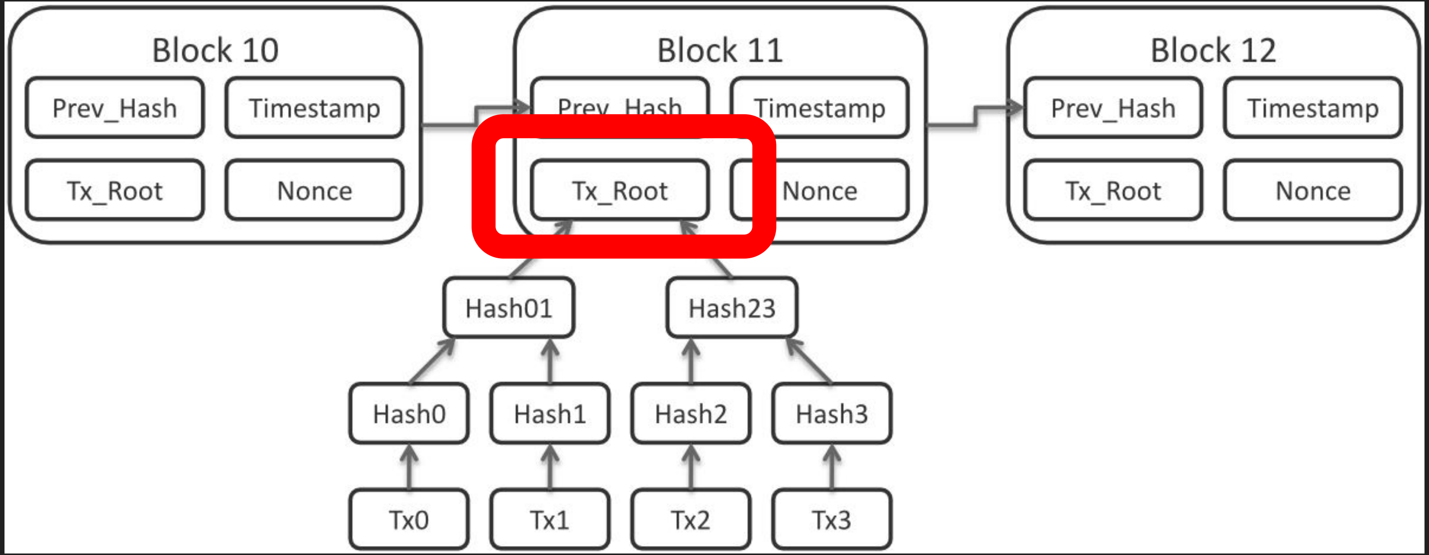
Lots Going on here!

i. And what makes it a “chain” of blocks, is that the header of each block contains an SHA-256 hash of the previous block header



Lots Going on here!

ii. This is a hash of all the transactions (often organised in a Merkel tree for efficiency) - much more space efficient, keeps the size of the header small

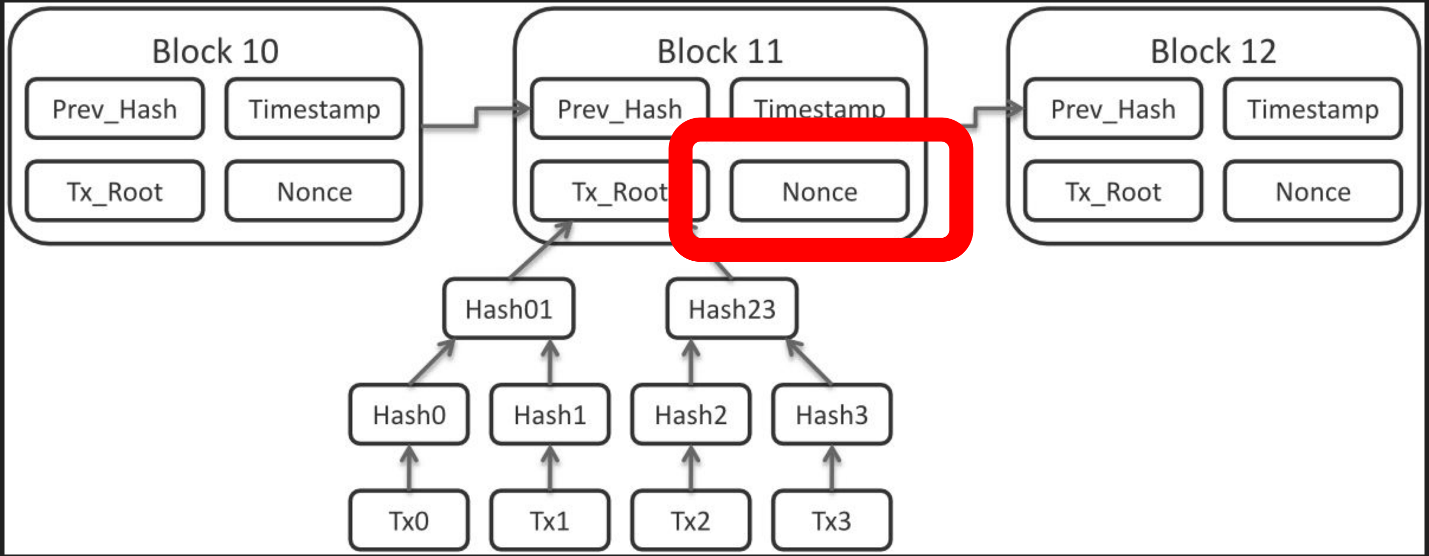


But how does this make things immutable?

- Consider this “chaining” of blocks
- If you want to check if a blockchain is valid, start from the current block, hash the previous block, and compare to the value in the current header
- Iterate all the way to the start of the chain, up to the “genesis block”

But what about the nonce ….. ?

The nonce is a part of the mechanism used to add blocks on to the chain



Time for some code!

We are going to implement different components of the blockchain in different classes

1. Transaction - a class to represent transactions
2. Block - a block will contain a header (prev_hash, timestamp, nonce, and hash of all transactions), and a list of transactions
3. Blockchain - this will perform validity checks, handle forks and maintain a pool of unverified transactions
4. Client - this will handle initiating transactions, creating wallets

Adding blocks to the chain

- This is the most critical aspect of any blockchain
- This needs to be done in such a way that malicious actors can’t retrospectively create a whole new modified chain, or start seizing control of adding to the chain with malicious intent

Consensus algorithm

- There are multiple independent actors making transactions that need adding to the blockchain - but there are multiple, potentially other, actors who are all trying to update the blockchain with these transactions
- We need a way for everyone involved to periodically agree and calibrate what the state of the blockchain is
- This is known as a consensus algorithm
- This is the most critical aspect of any blockchain
- This needs to be implemented in such a way that malicious actors can't retrospectively create a whole new modified chain, or start seizing control of adding to the chain with malicious intent

Proof of Work

- The proof of work algorithm involves an extra component, known as a nonce
- When a group of transactions are presented to be added to the chain, that need to be added, the following process is applied
    a. The transactions are validated
    b. The hash of the previous block header is computed
    c. A nonce is calculated which is defined as a value that when hashed with the rest of the header, results in a hash that starts with “many” 0s
- This process of finding a valid nonce is extremely costly - i.e. computing one is proof that on average, a certain amount of computational “work” has been done

PoW Ctnd

- The difficulty of finding a nonce scales exponentially with the number of 0s required
- A blockchain can define this difficulty however it wants
- Bitcoin, for example, is designed to process one block every 10 minutes, so the difficulty of finding a nonce is increased/decreased dynamically depending on the status of the network

But why bother?

- As an incentive, it is standard for PoW blockchains to reward “miners”, who approve transactions onto the blockchain with a payment
- This takes the form of allowing them to include place an extra transaction on their block, originating from some special address, bestowing a specific amount
- This is the infamous “mining” process (that is proven to be ruining the planet, availability of consumer electronics, and arguably hamstringing the technical development of blockchains and transition to better consensus algorithms …….)

But, what if two people mine a block

- Any consensus algorithm has to have a mechanism for dealing with conflicting/competing “forks”, or versions of the chain caused by miners competing in parallel.
- To do this, there has to be clearly defined rules, that nodes follow, to allow a “consensus” to be reached as to which of competing chains is correct
- In PoW, this correct chain is defined as the one that involved most “work”, i.e. most overall difficult nonces involved.
- This is typically computed as the average number of hashes required to compute each chain
- Hashes = $16^{\#\text{ zeros}}$, if the number with leading 0s is written in hex

Hint: remember, the nonce still has a numerical value

Transaction signing

- There is a problem with the implementation as I have shown it so far
- Anyone can submit a transaction to the chain, how do we know that it originated from the actual sender?
- Digital signing - a well known algorithm, the very same principle that underpins SSL certs
- This is based on public-key encryption - a technology in itself is at the bedrock of technological security

The RSA algorithm

- Initially conceived at GHCQ, finally “solved” by Ron Rivest, Adi Shamir and Leonard Adleman at MIT
- Key idea: you have a private key, that you use to encrypt data and a public key that you can share with everyone, that can be used to decrypt your encrypted data. It is insanely difficult to recreate the private key from the public key

The Maths

1. Pick two (very, very big) **prime** numbers, P, Q and multiply them
2. N = P*Q is the first part of the public key, and can be released publicly (P and Q are secret)
3. Pick e, an integer (not one of P,Q) such that $1<e<(P-1)(Q-1)$. This is the final part of the public key, and is released with N
4. The private key, $d = (k*(P-1)(Q-1) + 1) / e$ for some integer, k
5. Encryption is simply: $\text{MESSAGE}^{e} \bmod n$
6. Decryption: $\text{ENCRYTED\_MESSAGE}^{d} \bmod n$

- The only effective way to break RSA encryption is to factorise N into P and Q (hence why they must be prime), which is massively costly (RSA keys are often 2048 bits or more)
- This is very limited in terms of the size of message it can operate on, the exponential operators create issues when dealing with large files
- It is perfect for signatures, however, as we will now see

Note that many blockchain implementations use ECDSA algorithms rather than RSA, as they can provide the same security with keys of fewer bits. The mechanics of these are radically different to RSA, but the idea of public/private keys is analogous.

Signing a Transaction on a blockchain (or in general)

- Create an RSA public and private key pair
- The public key will represent your id, or origin address on the chain - think of it like an IBAN, but on a decentralised system
- Hash the details of a transaction you want to make (your address, the address of the recipient, the amount, timestamp, etc), using SHA-256
- Encrypt this with your private key. Bundle this encrypted data with a copy of the public key, and the details of the exact RSA protocol used. This is your signature
- Think about how given this signature, the reader can be 100% confident that the data it accompanies is from you or not

Cryptocurrency Wallets

- A wallet is simply a software or hardware solution that handles this process for you on a major blockchain - it creates and stores the keys, generates signatures, and posts them to miners to include on the chain

Let’s implement it!

But, how do we make sure someone can actually make a transfer?

- The most obvious way, is to scan the blockchain, and keep track of the net balance of an address
- If an account has balance greater than that of the amount of that transaction, the transaction can be processed, otherwise not
- This is known as the “account transaction model”
- Small fees are included with each transaction - this is a discrepancy between the amount paid and received, that the miner can claim

Unspent Transaction Output (UTXO)

- There is an alternative model, used by many blockchains - including Bitcoin
- Any received transaction is stored at the recipient's address, and is considered an unspent transaction
- The balance of an address is the sum of all its unspent transactions
- To make a transaction, you have to send whole UTXO(s), and receive whole new UTXO(s) as change - inputs/outputs
- Transaction fees increase with the number of UXTO(s) - not necessarily the value involved
- If a transaction is put in the pool, before the input UTXO that it references, it is an "orphan transaction", and is placed in a separate, orphan transaction pool

[1]

References

https://www.oreilly.com/library/view/mastering-bitcoin/ - a good source on much of the material here

https://github.com/adilmoujahid/blockchain-python-tutorial/blob/master/blockchain_client/blockchain_client.py - a basis for much of the code

https://bitcoin.org/bitcoin.pdf The original whitepaper that proposed Bitcoin

https://ethereum.org/en/learn/ The ethereum blockchain is far better roadmapped than bitcoin, and has excellent documentation for details of almost every aspect of the current blockchain, and plans for future development (check out their PoS documentation in particular)

Someone asked about NFTs ...

- Nothing more than a fancy term for a special form of Smart Contract

https://ethereum.org/en/developers/docs/standards/tokens/erc-721/