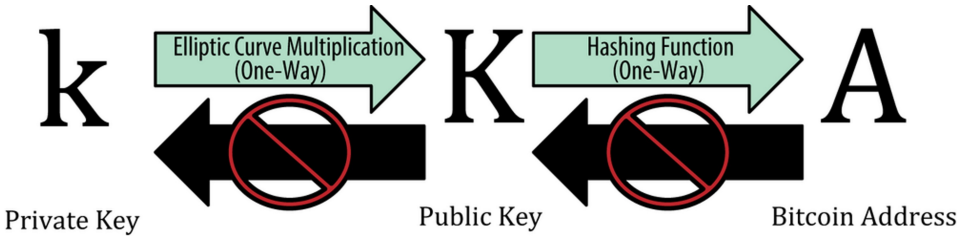
Principles of Distributed Ledger

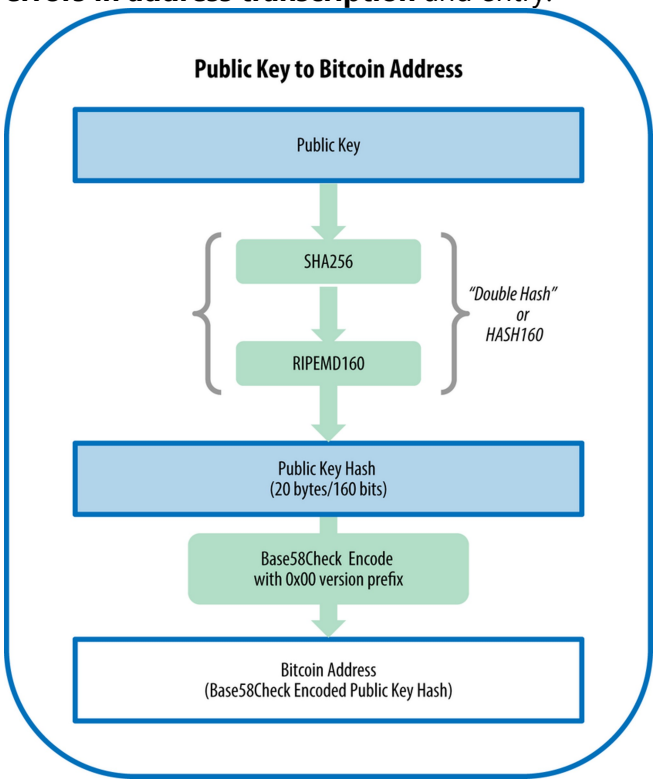
2021年1月21日 21:04

Introduction

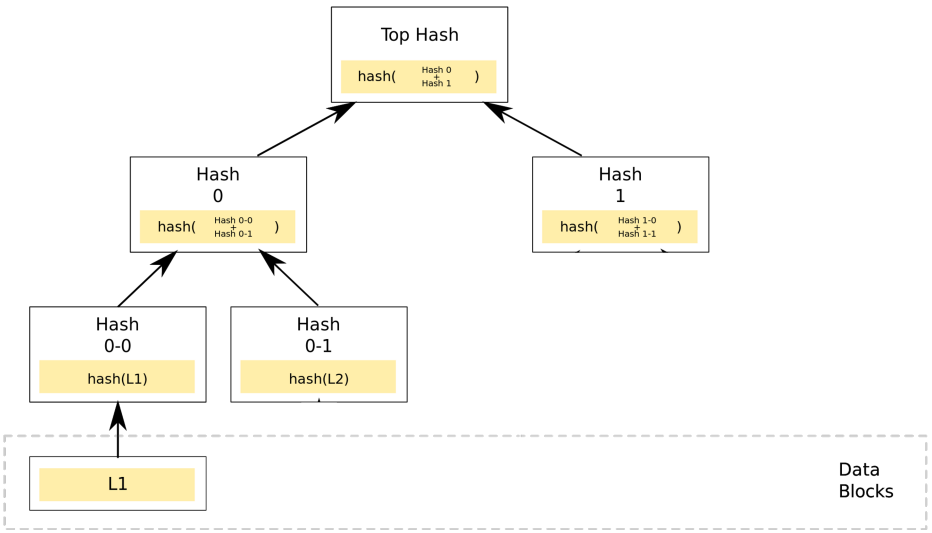
- **Privacy-Preserving Banking:**
  - eCash software on the user's local computer stored money in a digital format, cryptographically signed by a bank. The user could spend the digital money at any shop accepting eCash, without having to open an account with the vendor first, or transmitting credit card numbers.
- **Byzantine General's Problem**
  - https://www.capgemini.com/au-en/2018/07/what-is-the-blockchain-solving-the-generals-problem/
  - **Original Problem**
    - The problem lies in **how the generals communicate**. Each army uses the same messenger to communicate with each other. The only way for the messenger to move between armies is to sneak through the enemy city. This makes the **messenger untrustworthy as there is no way of ensuring that the message you receive is true and correct**. The messenger could be captured, a general could be a traitor, or you could receive no message at all.
  - Blockchain was designed as a solution to the Byzantine Generals' Problem. Satoshi Nakamoto, the inventor of Bitcoin, describes the problem where **each 'node' (a computer)** in a blockchain network can be thought of as a **general**. Each message is a **separate transaction that must be verified**, and its authenticity confirmed. Blockchain enables this verification through a process known as **consensus**. Rather than each general waiting for a single messenger to deliver its message, the generals can act in concert to verify messages broadcast to each other. Provided there are **more 'good generals' than there are 'traitors'**, then the network should be able to distill the authentic data from the bad data. This solution is known as decentralization and is critical to building blockchain networks.
  - Satoshi Nakamoto introduced the concept of **'proof-of-work' as his consensus algorithm** in the Bitcoin Blockchain.
- **Bitcoin Characteristics**
  - Peer-to-peer decentralized currency
  - **NO** trusted third parties
  - Based on distributed database
    - Transactions and Blocks are stored in a distributed fashion
- **Data Types**
  - **Cryptographic Hash Function**
    - **Pre-image Resistance:** Given the value of the hash, it is hard to find the original value (irreversibility)
      - For successfully 'breaking' preimage resistance, you get some output hash x and need to find the corresponding input m to that specific x. Unless there is a structural flaw in the hash algorithm, the **quasi-randomising effect of the hash function** dictates that the only way is to brute-force over a lot of different inputs until you find your target input m. This is extremely expensive, and intentionally so.

        - **Second Pre-image Resistance:** Given the original value x, it is hard to find another y that has the same hash value
        - **Collision Resistance**: Hard to find a pair of value x and y that has the same hash value
            - For collisions, you need to find any two distinct inputs m1,m2 that result in the same output hash. Now again you could choose a brute-force approach and try different inputs until you find a match. However you are not constrained by the fact that your inputs **must result in a specific hash**, like in both preimage cases. This makes it significantly easier to 'break' collision resistance.
            - **Easier to locate a pair of values that have the same hash**
        - **Hiding**: When add a salt, it is hard to find the original value
        - **Puzzle-friendly**:
            - In order for a hash function to be considered puzzle friendly, it must be true that there is not a strategy for solving a computational puzzle based on the hash function that's more effective than simply trying random values from a set of possible solutions until one of them works.
        - **Derive New Hash Function with different Properties**
            - http://classweb.ece.umd.edu/enee757.F2007/757Hash.pdf
    - **Elliptic Curve Signature algorithm:**
        - For example, at a security level of 80 bits (meaning an attacker requires a maximum of about 2^80 operations to find the private key) the size of an ECDSA private key would be 160 bits, whereas the size of a DSA private key is at least 1024 bits. On the other hand, the signature size is the same for both DSA and ECDSA: approximately 4t bits, where t is the security level measured in bits, that is, about 320 bits for a security level of 80 bits.
        - Only used to **sign transactions** and **verity signatures**
- **Malleable Signature**
    - A signature scheme is malleable if, on input a message and a signature, it is possible to efficiently **compute a signature on a related message**, for a transformation that is allowed with respect to this signature scheme
- **Bitcoin Address**
    - https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch04.html
    - 

    - Elliptic public key = ECDSA (Private key)
      Public key = '0x04' + Elliptic public key
      Encrypted public key = RIPEMD-160 (SHA-256 (Public key))
      Mainnet encrypted public key = '0x00' + Encrypted public key
      C = SHA-256 (SHA-256 (Mainnet encrypted public key))
      Checksum = first 4 bytes of C
      Hex address = Mainnet encrypted public key + Checksum
      Address = Base58 (Hex address)

- Bitcoin addresses are almost always presented to users in an encoding called **"Base58Check"**), which uses 58 characters (a Base58 number system) and a checksum to **help human readability, avoid ambiguity, and protect against errors in address transcription** and entry.
- 

- **Blockchain Transactions**
  - First transaction in a block is defined by the miner, so there is no input required.
  - For all following blocks, there must be an input and at least one output.
  - A **coinbase transaction** is a unique type of bitcoin transaction that can **only be created by a miner**. This type of transaction has **no inputs**, and there is one created with each new block that is mined on the network. In other words, this is the **transaction that rewards a miner with the block reward for their work.**
- **Blockchain Transaction Signature**
  - https://www.bitcoin.com/get-started/how-bitcoin-transactions-work/
  - Sending BTC requires having access to the public and private keys associated with that amount of bitcoin.
    - a **public key to which some amount bitcoin was previously sent**
    - the corresponding **unique private key** which **authorizes the BTC previously sent to the above pub-key** to be sent elsewhere
  - **Public Key => Equivalent to Bitcoin Address**
    - Public keys, also called a bitcoin addresses, are random sequences of letters and numbers that function similarly to an email address or a social-media site username.
  - **Signing a transaction**
    - In our example transaction above, Mark wants to send some BTC to Jessica. To do this, he uses his **private key to sign a message** with the transaction-specific details.
    - **Input:** the source transaction of the coins previously sent to Mark's address
    - **Amount:** some amount of BTC to be sent from Mark to Jessica
    - **Output:** Jessica's public address.
    - Anyone in the network that has the **public key** can then decrypt the message to prove that Mark indeed owns the transaction
  - **Verifying a transaction**

        - This transaction is then broadcast to the Bitcoin network where miners **verify that Mark's keys are able to access the inputs (i.e. the address(s) from where he previously received BTC) he claims to control.** This confirmation process is known as mining because it requires resource-intensive computational labor and rewards miners, in BTC, per block solved.
    - **Public-Private Key Encryption**
        - A digital signature in its simplest description is a hash (SHA1, MD5, etc.) of the data (file, message, etc.) that is subsequently encrypted with the **signer's private key.** Since that is something only the signer has (or should have) that is where the trust comes from. **EVERYONE has (or should have) access to the signer's public key.**
        - So, to validate a digital signature, the recipient
            - Calculates a **hash of the same data** (file, message, etc.),
            - Decrypts the **digital signature using the sender's PUBLIC key**, and
            - Compares the 2 hash values.
- **Scripts**
    - Stack-based Programming languages (Each input will insert into the stack, and each command will pop one most recent inputs from the stack)
    - When the script is **evaluated to true**, the transaction will be valid
    - It is crucial to make sure the script can be executed fast enough to prevent DoS attacks
    - **<Sig> <PubKey> OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OPCHECKSIG**
        - Make sure the **public key** is equal to the provided **public key hash**
        - Make sure the **public key** is used to **decrypt** the **signature**
- **Lightweight Clients (SPV Transaction Verification)**
    - 

    - In fact, there's no way to identify the peer that hold the hashes/transactions necessary. An SPV client will have to ask **each full node it is connected** to until it receives the correct information. Once the node found the hashes/transactions which match, it will know that everything is fine (as it's hard to "fake" a hash). If **no node can provide the fitting data**, the SPV client should conclude that the transaction it is verifying is in fact **not included in a block.**
    - **Blockchain Header**
        - After having a valid merkle root they build the block's header:
            - **Version** (4 bytes)

        - **Hash of the previous block**, thus making a chain of block (32 bytes)
        - **Merkle root**, the tree of transactions' reference (32 bytes)
        - **Timestamp**, number of seconds since 1970-01-01 00:00 (4 bytes)
        - **Bits**, a representation of the networks current difficulty (4 bytes)
        - **Nonce**, incremented when mining (4 bytes)
- **Blockchain Different Types of Nodes**
    - **https://www.seba.swiss/research/Classification-and-importance-of-nodes-in-a-blockchain-network**
    - Generally, there are three distinct types of nodes – **miner nodes, full nodes, and light nodes**. Miner nodes can **propose blocks and have the complete history of the blockchain**. Full nodes have the c**omplete history of the blockchain but cannot propose new blocks** and **light nodes rely on full nodes for blockchain's history.**
- **Blockchain New Block Generation Process**
    - When miners try to compute a block, they pick **all transactions** that they want to be added in the block, plus **one coinbase (generation) transaction** to their address. They may include any transaction they want to form a tree of transactions later hashed into the merkle root and referenced into the block's header.
    - It is to note that for a block to be accepted by the network it needs to contain **only valid transactions:** inputs that are not yet spent, inputs that have the valid ammount, signature that verifies ok and etc…
    - Then this where your miner enters the game. It starts with the **nonce at 0, hash (sha-256 2x) the block's header** and then check if the hash in under the current target (if you are on a pool the share target). If not it increments the nonce and hash again. If the **hash is under the current target** YOU FOUND A BLOCK, you then transfer your **block's header and the associated transactions' tree and the network accepts it**. Because you had a coinbase transaction in your block paying to your bitcoin address those bitcoin then becomes yours.
    - It is to says that finding a block is rare so your miner will eventually **run out of nonce** to use you can then **change parameter in your block header** (more likely the timestamp) or add a extra nonce field in your transactions tree thus changing the merkle root.
    - As for which transactions are chosen to be part of a block, it is mostly the **miners choice to include one or other transaction**. But normally they would **include transaction with a miners fee** in it to gain more bitcoin, a miner could also choose not to include any transaction and only get the coinbase one. The propagation of ones transaction through the network also influence which transactions a miner do include.
- **Bitcoin Incentivize System**
    - **https://medium.com/@craig_10243/bitcoin-is-all-about-incentives-72894518f6b5**

6. Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block. This adds an incentive for nodes to support the network, and provides a way to initially distribute coins into circulation, since there is no central authority to issue them. The steady addition of a constant of amount of new coins is analogous to gold miners expending resources to add gold to circulation. In our case, it is CPU time and electricity that is expended.

The incentive can also be funded with transaction fees. If the output value of a transaction is less than its input value, the difference is a transaction fee that is added to the incentive value of the block containing the transaction. Once a predetermined number of coins have entered circulation, the incentive can transition entirely to transaction fees and be completely inflation free.

The incentive may help encourage nodes to stay honest. If a greedy attacker is able to assemble more CPU power than all the honest nodes, he would have to choose between using it to defraud people by stealing back his payments, or using it to generate new coins. He ought to find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth.

    - **Coinbase Transaction**
        - Incentives for nodes to support the network, and provides a way to initially distribute coins into circulation by mining the block via Proof of work, in exchange for nodes' CPU time and electricity.
    - **Transaction Fees**
        - If greedy attackers such as Bitmain and Bitcoin.com who seek to drastically alter the protocol are able to assemble more CPU power than all the honest nodes, they would have to choose between using it to defraud people by stealing back their payments, or using it to generate new coins. The system is **secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.**
        - **(IMPORTANT) The adversary should find it more profitable to play by the rules, such rules that favour him with more new coins than everyone else combined, than to undermine the system and the validity of his own wealth**
- **Bitcoin Transaction Fees**
    - Every cryptocurrency transaction must be added to the blockchain, the official public ledger of all completed transactions, in order to be considered a successful and valid transfer.

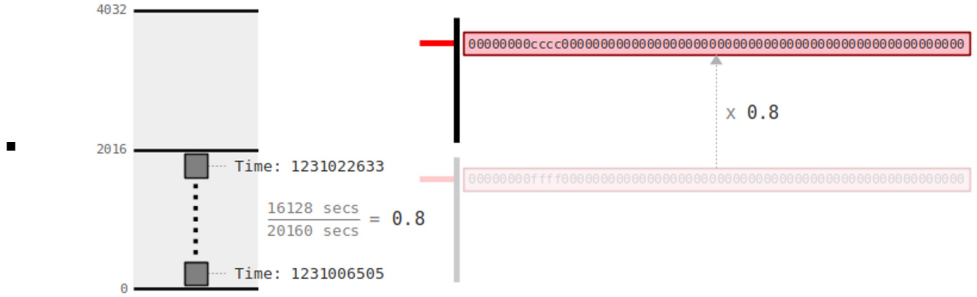
Proof of Work

- **Proof of Work**
    - Join the P2P network, listen for transactions, and **validate all incoming transactions**
        - If there is only txn 1 and txn 2 (Less than fulfilling an entire block), the miner can still try to create one (Because the bottom line is as long as it is **smaller than the block size limit**, it can be a validated block). The miner will keeps the iteration (using brute-force approach)
- **Main Chain**
    - Longest chain with the most difficulty
- **Hash Rate Calculation**
    - https://uploads-ssl.webflow.com/5d25da7a03e410dc1f3b7f36/5e6655654fda92cdd3163b27_proof%20of%20work.pdf
    - Hashrate (Hash per second, h/s) is an SI-derived unit representing the **number of**

double SHA-256 computations performed in one second in the bitcoin network for cryptocurrency mining. Hashrate is also called as hashing power. It is usually symbolized as h/s (with an appropriate SI prefix).

- **Average number of tries** is simply the difference between the total number of bits against the target bits.
- As to why it is this way is just a function of probability. If you have a range of 1-100, and your target is <= 20, there are 20 numbers that satisfy your requirement. Thus, 20 * 1/100, or 20 tries out of 100 should result in a valid number against your target.
- It works the same way, although on a much larger scale here. Your target and attempt are both 256 bits numbers, which means they range from **0 to 2^256-1**. If your target is 2^179, it means that 256-179, or 2^67 tries will, on average, **produce a valid number against the target.**

- **Blockchain Target Difficulty**
  - Each node on the network operates independently, so there is **no "single target value"** being sent around the network.
  - However, because all nodes **adopt the longest chain of blocks as their blockchain**, they will each end up **calculating the same target** as everyone else, so effectively all nodes end up sharing the same current target.
  - For example, when you run bitcoin for the first time, your node will perform the **initial block download and calculate the targets as it goes**. And because you're receiving the same blocks as everyone else, you will end up calculating the same current target too.
  - Furthermore, all nodes continually share the same view of the blockchain (as they will always adopt the longest available chain of blocks). Therefore, nodes will also continually calculate the same target value as each new block is mined.
- **Blockchain Difficulty Adjustment**
  - https://learnmeabitcoin.com/technical/target
  - https://medium.facilelogin.com/the-mystery-behind-block-time-63351e35603a
  - The difficulty is adjusted every 2016 blocks based on the time it took to find the previous 2016 blocks.
  - The average block time of the network is evaluated after n number of blocks, and if it is greater than the expected block time, then the difficulty level of the proof of work algorithm will be reduced, and if it is less than the expected block time then the difficulty level will be increased.
  - In each block, **in the header there is a parameter called, bits** — and in the genesis block the value of bits is 486604799. If we represent the same in hexadecimal it would be 1D00FFFF. This is a compact format — which can be used to find the target hash value for this (current) block. In fact **the hash of this block must be less than or equal to the target.**
  - **Reduce the difficulty when the block generation time is faster than expected**



  - **Increase the difficulty when the block generation time is slower than expected**

- 

    - **Purpose of Difficulty Adjustment**
        - If blocks are being mined **faster** than they can be broadcast across the network, it will result in **miners regularly building competing chains**. Only one of these will become the longest, so some miners will end up **wasting energy working to build on top of a competing chain** only for it to be left behind due to a chain reorganization.
        - Therefore, this time delay between blocks allows them to propagate the network so that more miners can adopt the longest available chain, which concentrates the network's mining power on extending the same chain of blocks.
    - **Difference between Bitcoin and Ethereum Difficulty Adjustment**
        - The main difference between bitcoin's level of difficulty and the ethereum's is, in bitcoin, the difficulty adjustment is done after 2016 blocks to **maintain the block time at a constant value** (even though the computational power increases)
        - In Ethereum, **based on the computational power, the block time will increase (or decrease) due to the difficulty bomb impact** (not trying to keep it at a constant value) — and the adjustment is done in **each block.** So, to keep Ethereum block time at a considerable level, the computational power also must increase with the time (to match the difficulty bomb) — if not, it will be hard to mine ethereum, which will result in a ethereum blockchain freeze — also known as ice age.
- **Pool Difficulty and Bitcoin Difficulty**
    - Pool difficulty should be slightly easier than the bitcoin difficulty, and each members in the pool will be asked to solve a easier questions and demonstrate the proof of work for receiving their proportion shares of profit
- **Merged Mining**
    - https://bitcoin.stackexchange.com/questions/273/how-does-merged-mining-work
    - **Mechanism Overview**
        - The miner (or mining controller in the case of pooled mining) actually builds a block for both hash chains in such a way that the **same hash calculation secures both blocks**. Work units based on this block are then assigned to miners. If a miner solves a block (at the difficulty level of either or both block chains) the **block is re-assembled with the completed proof of work** and submitted to the correct block chain (or both blocks are separately reassembled and each submitted to the corresponding network if it met both of their difficulty requirements).
        - Basically the idea is that you **assemble a Namecoin block and hash it**, and

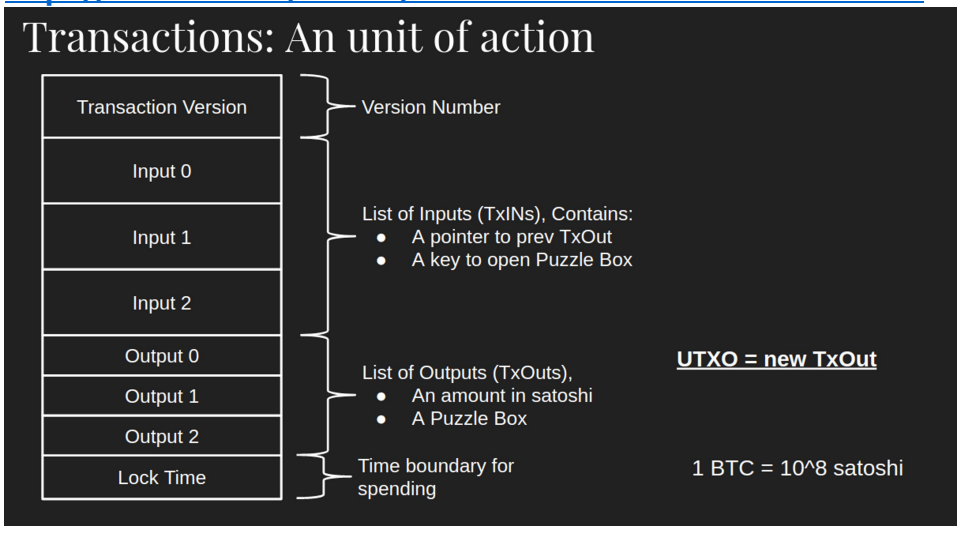
then **insert that hash into a Bitcoin block**. Now when you **solve the Bitcoin block at a difficulty level greater to or equal to the Namecoin difficulty level**, it will be proof that that amount of work has been done for the Namecoin block. The Namecoin protocol has been altered to accept a Bitcoin block (solved at or above the Namecoin difficulty level) containing a hash of a Namecoin block as proof of work for the Namecoin block. The Bitcoin block will only be acceptable to the Bitcoin network if it is at the difficulty of the Bitcoin network.

- **Extra Things in Both Network**

  The Bitcoin block chain gets a **single extra hash** when a merged mining block is accepted, and the Namecoin block chain gets a little bit more (because it includes the Bitcoin block) when a merged mining block is accepted. However, because of the Merkle Tree, the entire Bitcoin block doesn't need to be included in the Namecoin tree, just the top level hashes (so the extra bloat to the Namecoin chain is not a big problem).

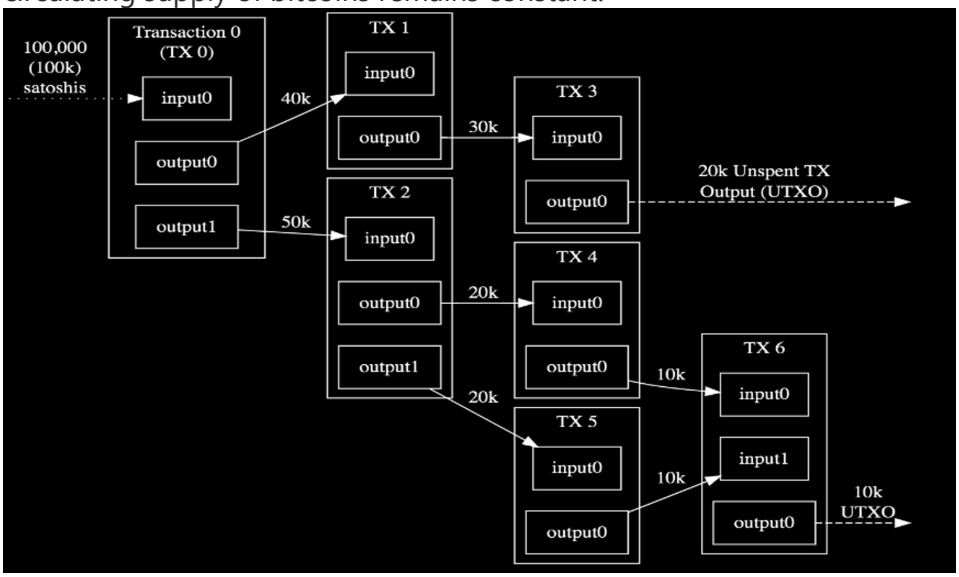
- **Example with Bitcoin and Namecoin: Namecoin supports Merged Mining**
  - First, the miner must **assemble a transaction set** for both block chains. He then assembles the final Namecoin block and hashes it. He then creates a transaction containing this hash that is valid in the Bitcoin chain and inserts it in the Bitcoin transaction set at the tip of the tree. He then assembles the final Bitcoin header with this transaction in it and sends out the work units.
  - If a miner solves the hash at the Bitcoin difficulty level, the Bitcoin block is assembled and sent to the Bitcoin network. The Namecoin hash does nothing and the Bitcoin network ignores it.
  - If a miner solves the hash at the Namecoin difficulty level, the Namecoin block is assembled. It includes the Namecoin transaction set, the Namecoin block header, the Bitcoin block header, and the hash of the rest of the transactions in the Bitcoin block. This entire "mess" is then submitted to the Namecoin system. The Namecoin system, supporting merged mining, accepts this as proof of work because it contains work that must have been done after the block header and Namecoin transaction set was built. (Because you can't build the Bitcoin transaction set containing that hash, and therefore the Bitcoin header that secures it, without that information. So it proves the work was done.)
- **Three key points to remember:**
  - The Bitcoin chain doesn't get junked up with Namecoin stuff due to merged mining. At most, **one tiny hash is inserted in the transaction tree**.
  - The two hash chains remain fully independent. The "Bitcoin stuff" that goes in the Namecoin tree is basically ignored and only used to validate the proof of work. (It will bloat the Namecoin chain a bit as it means some blocks will have an extra header and an extra hash.)
  - Lastly, **no special support is needed from Bitcoin**.
- **Disadvantages of Merged mining**
  - Miners have shown to **not always validate the blocks** they mine upon. Validating a block means to:
  - 1) Check the PoW by **hashing the header and verifying that the hash is smaller than the current target**
  - 2) Verifying that ***all*** transactions within the block are valid (e.g. don't

double spend funds etc.)
        - If miners continue mining on invalid blocks (which happened in the past), then eventually they will find out that they mined on an invalid chain and a longer fork will be provoked. Thus, such inconsistency is a major concern of merged mining.
- **Forking**
    - **https://academy.binance.com/en/articles/hard-forks-and-soft-forks**
    - **Hard Fork**
        - When a hard fork creates a new cryptocurrency, holders of the original currency can claim the same amount in the new coin in addition to the ones they hold. What this means is, when a fork occurs, you can get free coin. When a fork is announced, the original coin price can go down.
        - Hard forks are **backward-incompatible** software updates. Typically, these occur when nodes add new rules in a way that conflicts with the rules of old nodes. New nodes can only communicate with others that operate the new version. As a result, the blockchain splits, creating two separate networks: one with the old rules, and one with the new rules.
        - **All nodes need to upgrade to the latest version**
    - **Hard Fork Example**
        - An example of a hard fork was the 2017 fork that saw Bitcoin fragmented into two separate chains – the original one, Bitcoin (BTC), and a new one, Bitcoin Cash (BCH). The fork occurred after a lot of arguing over the best approach to scaling. Bitcoin Cash proponents wanted to increase the block size, while Bitcoin proponents opposed the change.
        - **An increase in block size requires modification of the rules.** This was before the SegWit soft fork (more on that shortly), so nodes would only accept blocks smaller than 1MB. If you created a 2MB block that was otherwise valid, other nodes would still reject it.
        - **Only nodes having changed their software to allow blocks exceeding 1MB in size could accept those blocks.** Of course, that would render them incompatible with the previous version, so only nodes with the same protocol modifications could communicate.
    - **Soft Fork**
        - A soft fork is a backward-compatible upgrade, meaning that the **upgraded nodes can still communicate with the non-upgraded ones**. What you typically see in a soft fork is the addition of a new rule that doesn't clash with the older rules.
        - Only previously **valid** blocks/transactions are made invalid ->Reducing functionality
    - **Soft Form Example**
        - A good real-life example of a soft fork was the aforementioned **Segregated Witness (SegWit)** fork, which occurred shortly after the Bitcoin/Bitcoin Cash split. SegWit was an update that **changed the format of blocks and transactions**, but it was cleverly crafted. Old nodes could still validate blocks and transactions (the formatting didn't break the rules), but they just wouldn't understand them. **Some fields are only readable when nodes switch to the newer software, which allows them to parse additional data.**
    - **Difference between the Hard Fork and Soft Fork**
        - Imagine the original blockchain block size if **limited to 3MB**
            - A **soft fork** example would be **reducing the size limit to 2MB**. Old

nodes without updating to the newest protocol can still process transactions as long as they limit their block size within 2 MB.
        - A **hard fork** example would be **increasing the size limit to 4MB**. So when a node that accepts new protocol produces a new block with 4 MB block size, old nodes that doesn't update to the newest protocol will consider the block as invalid.
    - Hard forks and soft forks are essentially the same in the sense that when a cryptocurrency platform's existing code is changed, an old version remains on the network while the new version is created. With a soft fork, **only one blockchain will remain valid** as users adopt the update. Whereas with a hard fork, **both the old and new blockchains exist side by side**, which means that the software must be updated to work by the new rules. Both forks create a split, but a hard fork creates two blockchains and a soft fork is meant to result in one.
- **Leader Election Mechanism**
    - **Proof of Stake**
        - https://academy.binance.com/en/glossary/proof-of-stake
        - With Proof of Stake (POS), Bitcoin miners can mine or validate block transactions **based on the amount of Bitcoin a miner holds.**
        - Proof of Stake (POS) was created as an **alternative to Proof of Work (POW)**, which is the original consensus algorithm in Blockchain technology, used to confirm transactions and add new blocks to the chain.
        - The locked funds will then act as **collateral**, meaning that **malicious validators** will most likely **lose their stake and be kicked out of the network**. On the other hand, honest validators will be rewarded as new blocks are produced (forged). Thus, we may say that a PoS blockchain achieves distributed consensus according to the economic stake that validators commit to the network.
        - Proof of Work (POW) requires huge amounts of energy, with miners needing to sell their coins to ultimately foot the bill; Proof of Stake (PoS) gives mining power based on the percentage of coins held by a miner.
        - Proof of Stake (POS) is seen as less risky in terms of the potential for miners to attack the network, as it structures compensation in a way that makes **an attack less advantageous for the miner.**
        - **PoS increases scalability while reduces centralization**
- **Bitcoin UTXO**
    - **https://medium.com/bitbees/what-the-heck-is-utxo-ca68f2651819**
    - 

    - **Transaction Output**
        - A transaction Output consists of a cryptographic lock and a value. The

outputs are somehow locked and the input provides a key to unlock them. The value is simply the amount in satoshis (sats) that is locked inside the output.

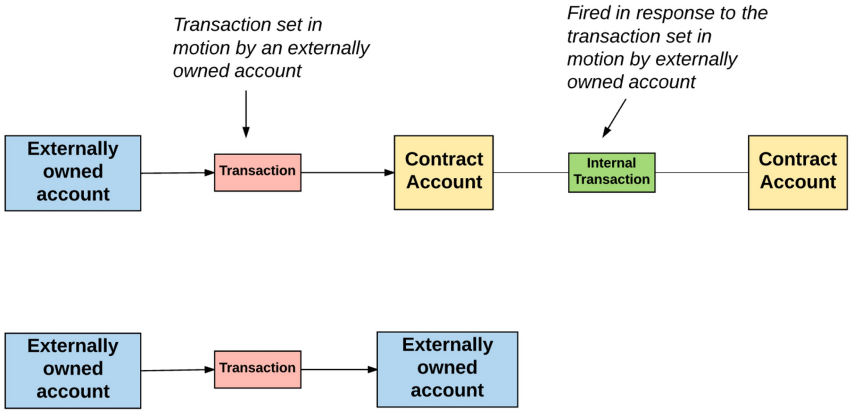
- **Transaction Input**
  - Every Transaction input consists of a **pointer** and an **unlocking key**. The pointer points back to a **previous transaction output**. And the key is used to **unlock the previous output** it points to. Every time an output is successfully unlocked by an input, it is marked inside the blockchain database as "**spent**". Thus you can think of a transaction as an abstract "action" that defines unlocking some previous outputs, and creating new outputs.
- **UTXO**
  - Once an output is **unlocked**, imagine they are removed from circulating supply and new outputs take their place. Thus the **sum of the value of unlocked outputs will be always equal to the sum of values of newly created outputs** (ignoring transaction fees for now) and the total circulating supply of bitcoins remains constant.
  - 

  - The collection of **all the UTXOs at any moment is called the UTXO set** and is being constantly maintained by every Bitcoin node. Technically they are known as the **chainstate and are stored in the chainstate data directory of a node**. The chainstate updates every time a new block is accepted in the blockchain. That block contains the **list of latest transactions**, which defines which of the **previous UTXOs are spent** and which **new UTXOs are created**. Every Bitcoin node in the network will always have the exact same copy the UTXO set in their local storage.
- **UTXO in Transaction**
  - In a transaction, UTXOs are always consumed in full, even if the required payment is of the partial value of the original UTXO. For example, referring to the figure above, suppose Alice wants to pay Bob for some goods/services which cost 6 BTC. But Alice only has a single UTXO of 10 BTC. She will create a transaction that will consume up this whole UTXO of 10 BTC and create two new UTXOs, one for Bob with 6 BTC and one for herself with 4 BTC value. The UTXO Alice pays to herself is known as the change UTXO, and her wallet software automatically tracks this UTXO to give her the final balance of 4 BTC. Every time a payment is made for a smaller value than the available UTXOs, **the wallet automatically creates the change UTXO for the user.**

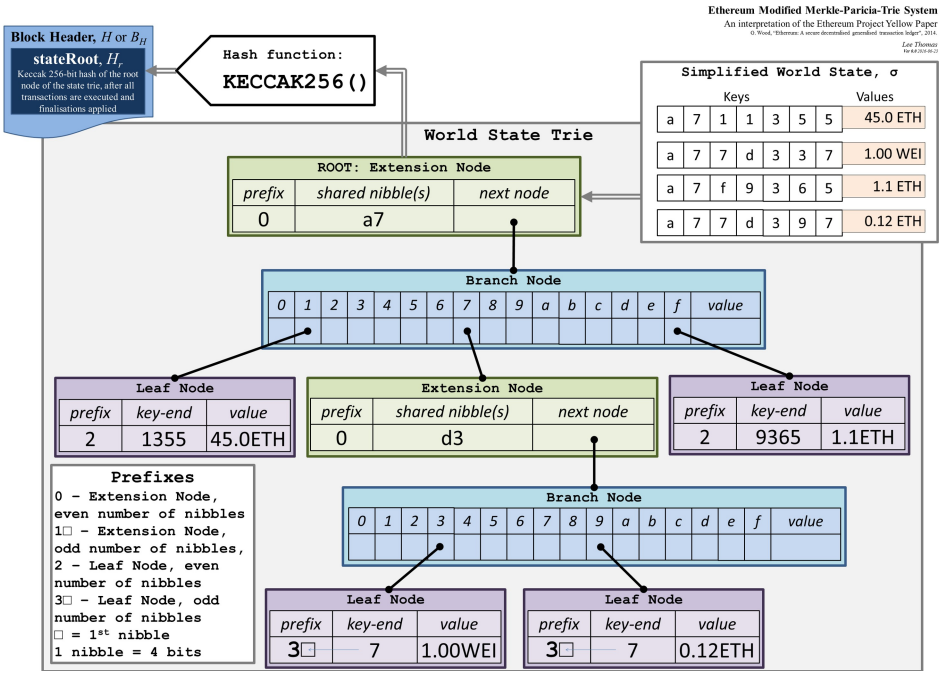
Bitcoin and Smart Contracts

- Bitcoin does have smart contracts, but without loops (i.e., implemented in Ethereum). Bitcoin does not have **high-level programming language (e.g. Solidity in Ethereum)** to compile code into byte codes
- **Smart Contract**
	Traditional	Smart
Specification	Natural language	Code
Identity & consensus	Written Signatures	Digital signatures
Dispute resolution	Judges	Decentralized platform
Nullification	Judges	?
Payment	Legally enforced	Built-in
Escrow	Trusted Third Party	Built-in
  - Smart contract refers to arbitrary code executed by all participants, with global consensus over execution. It allows automated verification and enforcement of contracts, and allow transfer of funds
- **Namecoin**
  - **Definition**
    - When Julian wants to buy wikileaks.bit, he first broadcasts a **commitment** transaction (NAME_NEW). **The transaction doesn't reveal the name he wants to buy;** it instead consists of a hash of two things: the name he wants to buy, and a secret randomly generated string (called a salt). Without knowing the salt and the name already, Keith can't determine what name Julian is buying.
    - After the **commitment transaction has received 12 confirmations (12 BLOCKS!)**, Julian broadcasts a 2nd transaction that **reveals the name and the salt**. Once this 2nd transaction is mined, Julian officially owns wikileaks.bit. The key point here is that once the 2nd transaction is broadcasted, Julian only needs **1 block to be mined** in order to obtain the name – but if Keith tries to register it, he'll need to broadcast his own commitment (now that he knows the name) and wait for 12 blocks before he can register the name.
  - **Timeline**
    - NAME_NEW (Commitment): Hash(random_nonce, 'wanted-registered-domain')
    - Wait for 12 blocks (Confirmations from other miners)
    - NAME_FIRST_UPDATE (Actual Transaction): (random_nonce(public), 'domain', 'IP)
    - NAME_UPDATE (Transaction): Update IP Address
  - **Front-running Attack**
    - Julian broadcasts a transaction, indicating that he wants to buy wikileaks.bit.
    - Keith sees Julian's transaction as soon as it gets broadcasted.
    - Keith infers that Julian places a significant value on wikileaks.bit.
    - Keith broadcasts his own transaction, trying to buy wikileaks.bit.
    - Keith bribes a miner to mine his transaction rather than Julian's. (This could

be done by using a much higher transaction fee than Julian's transaction, or it could be done via out-of-band channels, perhaps a court order.)
        - Keith now owns wikileaks.bit, not Julian, even though Julian was the first person to try to buy it.
- **Ethereum State Machine**
    - **https://medium.com/@eiki1212/ethereum-state-trie-architecture-explained-a30237009d4e**
    - **Accounts**
        - Externally owned accounts, which are controlled by private keys and have no code associated with them.
        - Contract accounts, which are controlled by their contract code and have code associated with them.
        -



    - **Account State**
        - **nonce:** If the account is an externally owned account, this number represents the **number of transactions** sent from the account's address. If the account is a contract account, the nonce is the **number of contracts** created by the account.
        - **balance:** The number of Wei owned by this address. There are 1e+18 Wei per Ether.
        - **storageRoot:** A hash of the root node of a Merkle Patricia tree. This tree encodes the hash of the storage contents of this account, and is empty by default.
        - **codeHash:** The hash of the EVM code of this account. For contract accounts, this is the code that gets hashed and stored as the codeHash. For externally owned accounts, the codeHash field is the hash of the empty string.

- 

- **Ethereum Transaction Structure**
  - https://medium.com/@eiki1212/ethereum-transaction-structure-explained-aa5a94182ad6

type	from	sig	nonce	to	data	value	gaslimit	gasprice

send	sender	sig	nonce	receiver	∅	amount	?	?

create	creator	sig	nonce	∅	code	start_bal	?	?

call	caller	sig	nonce	contract	f, args	amount	?	?

	account	contract
address	Hash(pub_key)	H(creator, nonce)
code	∅	EVM code
storage	∅	Merkle storage root
balance	ETH balance	
nonce	Number of transactions sent	

  - **- Nonce**
    - Ethereum has two types of nonce: **proof-of-work nonce** and **transaction nonce**. Transaction nonce is a sequence number of transactions sent from a given address. **Each time you send a transaction, the nonce value increased by one.** Moreover, Nonce prevents replay-attack on Ethereum blockchain.
  - **- Gas price**
    - Gas Price represents the price of Gas in Gwei. For example, 1 Gas = 10 Gwei. It is basically determined by market supply and demand. Gas

Price is used to **multiply Gas Limit to determine the final price of Gas.**

- **- Gas Limit**
  - Gas Limit is a **limit of the amount of ETH** the sender is willing to pay for the transaction. Usually, when one is talking about Gas in Ethereum, they are referring to **Gas Limit**. When transferring ETH, the sender needs to set Gas Limit. **If Gas Limit is not enough to transfer, the transfer will be canceled and Gas will be refunded to the sender.** On the other hand, if the sender sets excess Gas Limit, Gas left over will be refunded to the sender.
- **- Recipient(EOA, Contract Account)**
  - The recipient is the destination of Ethereum address. It is either EOA or contract address that represented by 20-byte.
  - **EOAs** are an account controlled by users with their **own private keys**. It has an ether balance, can send/initiate transactions, and control a private key. Differently from Contract Account, it has no associated code and is simply used for transferring money.
  - **Contracts** have addresses, just like EOAs. However, the **contract account does not have a private key**. Instead, It is owned by the logic of its smart contract code. **Any execution in the contract is triggered by transactions or messages produced by EOA.** Contract Account cannot initiate a transaction by themselves because it does not hold a private key.
- **- Value**
  - The value field represents the **amount of ether/wei from the sender to the recipient**. Value is used for both transfer money and contract execution. It is possible to construct transactions without filling value field though, it is supposed to be filled all the time.
- **- Data**
  - Data field is for contract related activities such as deployment or execution of a contract. **Data contains messages that can be conceived of as function calls.** As Ethereum has smart contract function, **a transaction needs to contain messages in order to call/execute functions**. Messages are produced by contract and execute CALL or DELEGATECALL opcodes. If Data field is empty, it means a transaction is for a payment not an execution of the contract.
- **- v, r, s(ECDSA)**
  - This field is components of an ECDSA digital signature of the originating EOA. Ethereum transactions use ECDSA(Elliptic Curve Digital Signature Algorithm) as its digital signature for verification. v indicates two things: the chain ID and the recovery ID to help the ECDSArecover function check the signature. r and s are inputs of ECDSA to generate a signature. If you want to know more details, please refer to another resource like here.

- **Ethereum Blockchain Structure**
  - **State:** has the information of all accounts in the blockchain, it is not stored in each block. The state is generated processing each block since the genesis block. Each block will only modify parts of the state.
  - **Transaction Tries:** records transaction request vectors
  - **Transaction Receipt Tries:** records the transaction outcome
  - Two tries should share the similar structure
- **Ethereum Stack, Memory and Storage**
  - Memory is zero-initialized and arranged in 256-bit words

    - **Cost of Storing**
        - fee of 20k gas to store a 256 bit word
        - Gas Price = 10 Gwei = 10^10 Wei = 10^(-8) ETH
        - 1 kilobyte --> 640k gas --> 0.0064 ETH = 6.4 USD
        - The cost of storing **1 kb is currently 6.4 USD**
    - **Storage** very expensive, the contract storage values are stoed in the blockchain forever
    - **Memory**- cheaper than storage, only accessible during contract executive, once execution is finished contents are discarder
    - **Stack**- The compiler generally uses it for intermediate values in computations and other scratch quantities.
- **Ethereum Gas System**
    - Each opcode of the EVM has a certain gas cost, used to limit computational cost and DoS attack
    - Miners can choose transactions based on GAS_PRICE, and they tend to favor transaction with larger GAS_PRICE
    - The GAS_PRICE x GAS_LIMIT should smaller than the accounts[from].balance
    - **Out of Gas Exception**
        - The state will be **reverted back to previous state**
        - Account still need to deduct the GAS_PRICE x GAS_LIMIT to compensate miners
        - For example, contract A calls contract B, whereas the contract B cause Out-Of-Gas exception, it will return back to the original state before the contract A is executed.

Classical Consensus

- **Consensus Mechanism**
    - A consensus mechanism is a fault-tolerant mechanism that is used in computer and blockchain systems to achieve the **necessary agreement** on a single data value or a single state of the network **among distributed processes** or multi-agent systems, such as with cryptocurrencies.
    - The **proof of work (POW)** is a common consensus algorithm used by the most popular cryptocurrency networks like bitcoin and litecoin. It requires a participant node to prove that the work done and submitted by them qualifies them to receive the right to add new transactions to the blockchain. However, this whole mining mechanism of bitcoin needs high energy consumption and longer processing time.
    - **Why need Proof of Work as consensus mechanism**
        - https://medium.com/coinmonks/simply-explained-why-is-proof-of-work-required-in-bitcoin-611b143fc3e0
        - Proof of work (PoW) is necessary for security, which prevents fraud, which enables trust. This security ensures that **independent data processors (miners) can't lie about a transaction**.
        - Proof of work is used to securely sequence Bitcoin's transaction history while increasing the difficulty of altering data over time. It is used to **choose the most valid copy of the blockchain in the network if there are multiple copies**. Finally, proof of work is the key to creating a distributed clock, which allows miners to freely enter and leave the network while maintaining a constant rate of operation.
- **Requirements in Blockchain Consensus (PoW, ...)**
    - Bitcoin's decentralized consensus emerges from the interplay of four processes that occur independently on nodes across the network:

    - Independent **verification of each transaction**, by every full node, based on a comprehensive list of criteria
    - Independent **aggregation of those transactions into new blocks by mining nodes**, coupled with demonstrated computation through a proof-of-work algorithm
    - Independent **verification of the new blocks by every node and assembly into a chain**
    - Independent selection, by every node, of the chain with the **most cumulative computation demonstrated through proof of work**
- **Byzantine General Problem and Proof of Work**
    - **https://www.radixdlt.com/post/what-is-proof-of-work/**
    - It empowers the distributed and un-coordinated Generals to come to an agreement:
        - The Generals **agree the first plan received by all Generals will be accepted as the plan**
        - A General solves the **PoW problem**, creating a block that is broadcast to the network so that all Generals receive it
        - Following receipt of this block, **each General verifies and works on solving the next PoW problem**, incorporating the prior solution into it, so that **their plan adds on to the previous resolution**
        - Each time a General solves a PoW problem, a block is generated and the chain begins to grow. In time, **any General working on a different solution will switch over to the longest chain**. This is the one **most Generals are contributing to and therefore has the greatest chance of success**
        - As the Generals know roughly how long a PoW solution takes to solve, after a set amount of time they will know if enough of the other Generals are also working on the same chain
    - Through this process, **the Generals can arrive at a consensus of when to attack, can estimate their chances of successfully doing so, and can prevent multiple different signals to attack being sent simultaneously.**
- **Impossibility Result from Fischer, Lynch and Patterson**
    - The problem of consensus - that is, getting a distributed network of processors to agree on a common value - was known to be solvable in a synchronous setting, where processes could proceed in simultaneous steps. In particular, the synchronous solution was resilient to faults, where processors crash and take no further part in the computation. Informally, **synchronous models allow failures to be detected by waiting one entire step length for a reply from a processor**, and presuming that it has crashed if no reply is received.
    - This kind of failure detection is **impossible in an asynchronous setting**, where there are no bounds on the amount of time a processor might take to complete its work and then respond with a message. Therefore it's not possible to say whether a processor has crashed or is simply taking a long time to respond. The FLP result shows that **in an asynchronous setting, where only one processor might crash, there is no distributed algorithm that solves the consensus problem.**
- **Timing Models**
    - **Asynchronous:** Message will be eventually delivered
    - **Synchronous:** Message will be guaranteed on the time of delivers
    - **Eventually Synchronous:** There exists a known upper bound on the delivered time which is variable
- **Broadcast Models**

    - https://dcl.epfl.ch/site/_media/education/sdc_byzconsensus.pdf
    - https://medium.com/logos-network/why-proof-of-work-is-not-viable-in-the-long-term-dd96d2775e99
    - **Consistent Broadcast:** A corrupted sender implies that not every party might terminate/deliver a request
        - In consistent broadcast, the sender first executes c-broadcast with request m and thereby starts the protocol. All parties terminate the protocol by executing c-deliver with request m.
          Consistent broadcast **ensures only that the delivered request is the same for all receivers.** In particular, when **the sender is faulty, it does not guarantee that every party terminates and delivers a request.**
    - **Reliable Broadcast**
        - A protocol for reliable broadcast is a consistent broadcast protocol that satisfies also
            - **Totality**: If some correct party r-delivers a request, then all correct parties eventually r-deliver a request.
        - Reliable broadcast additionally **ensures agreement on the delivery of the request** in the sense that **either all correct parties deliver some request or none delivers any request;**
        - Reliable broadcast supports reliability only, that is, a message that is broadcast is delivered to all alive nodes, even if failures occur in the system.
- **Known Impossibility Results**
    - In a fully asynchronous system, there is no deterministic consensus solution that tolerates one or more failures, and no algorithm can always reach consensus in **bounded time**
    - **Implication**
        - We need to have a timing assumption, and randomness in leader election is crucial
- **Consensus System for Bitcoin**
    - We do not need for a final consensus output (i.e., only the majority of the network agrees should be enough)
    - Block/transaction reward as incentive to participate
    - **Participating nodes do not need to be known upfront**
- **Bitcoin Node**
    - 115 outgoing connections and 8 incoming connections
    - **Incoming Packet** needs **signature validation, consistency and UTXO validation, denial of service protections**
- **DoS Protection Mechanism**
    - Sanity checks (lower penalty)
        - Size
        - Encoding
    - Critical points (penalty 100)
        - Signature validation
        - UTXO/double spending validation
- **Bitcoin Propagation Methods**
    - **Standard**
        - Send first the hash of an object (i.e., inv), transaction/block => **36 bytes**
        - Recipient requests the object (i.e., if not known this object before)
        - Sender transmits the object
    - **Send Headers**
        - Send first the block header (no more block hash) => **80 bytes**

        - Then the block
    - **Unsolicited Block Push**
        - Directly push the entire block to the network (i.e., usually used when a miner mines a block)
- **Bitcoin Fibre**
    - **Fibre Overview**
        - Fibre (Fast Internet Bitcoin Relay Engine) is a protocol which attempts to deliver Bitcoin blocks around the world with delays as close to the physical limits as possible.
        - **The point is that a miner can more quickly switch to a new block that has been found on the network.**
        - This is very important because Bitcoin mining is intended to be a fair lottery: If you have x% of the hashrate you should find x% of the blocks, on average. But when it takes time to communicate blocks mining acts more like a race than a lottery: the bigger miner gets an unjust share, creating a pressure to centralize.
        - This is because when a miner makes a block other miners can't work on extending it until they've heard of it, during that time they might make their own blocks. If you are a small miner and make a block this works against you because no one attempts to mine the successor to your block until later. If a larger miner mines a block it also works against you, because you've fallen behind.
        - This race behavior is the same that enables **selfish mining.**
        - The race effect can also benefit attackers trying to overpower the network, because they don't need to compete with hashrate lost to stale blocks.
        - **Improving propagation speeds reduces the race and makes mining more fair**, lowers pressure to centralize, and improves network security.
    - **Fibre Specification**
        - Fiber first sends a short block sketch: A list of **short hashes and lengths** that allow the far end to use its **mempool to lay out transactions into a block**, with **holes in positions where the transactions were missing or the hash was ambiguous**.
        - Then Fibre breaks the blocks into packets and sends **error correction data**. The error correction lets the receiver recover the block as soon as it has received **as many packets as missing chunks**, even though the sender doesn't know what's missing. Error correcting coding is also used on the initial sketch. This is similar to how Raid or PAR encoding works, but the implementation is made much more complicated by needing to be very fast.
        - All of this communication happens over UDP with pre-programmed transmission rates, so it doesn't need to ramp up slowly or wait for lost packets.
- **Double Spend Attack (General 1-n confirmation)**
    - Broadcast to the network a transaction where attacked merchant receives payment
    - Secretly mine a branch that is built upon the block before the transaction, a transaction that pays the attacker
    - Once the transaction to the merchant receives enough confirmations, and the merchant sends the product
    - Continue the secret contradictory brand until it is longer than the public transaction, and then make the blocks public. The network will identify the secret branch to be valid because it is longer than the public block, and the payment to the merchant will be replaced by the payment to the attacker.

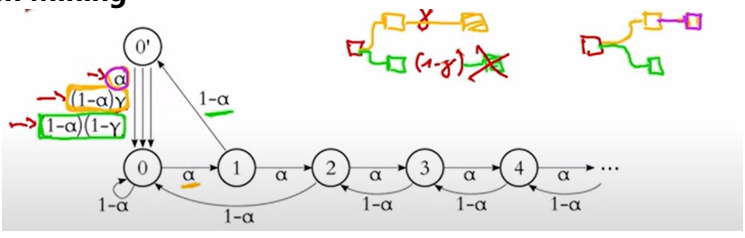
    - Most merchants wait until a number (N) of blocks have been confirmed before releasing their goods to prevent against double spend attacks. However, if enough miners have been eclipsed then an attacker can launch a double spend attack against an eclipsed merchant in spite of this. An attacker would be able to present their transaction to eclipsed miners, who add it to their ledger. This state is then shown to the merchant who cannot know better, given they too have been eclipsed and removed from the wider network. Once the merchant has sent the reciprocal goods to the attacker then the eclipsed miners' ledger is discarded and the merchant is left empty handed
    - ***We should assume there is no fiber connection between weak miner and the rest of mainstream netowkr***
- **Double Spend Attack (0-day Confirmation)**
    - Once a malicious actor has isolated a user by taking control of all outgoing connections they are able to exploit them by, for example, carrying out a 0 confirmation double spend attack. If User A is the malicious actor, User B is the isolated node and User C is another network entity, then User A would be able to send a payment to User C and then send the same transaction to User B. User B is unaware that those funds have already been spent as **all their outbound connections route through User A who is able to suppress and manipulate what information User B receives.** User B will accept the coins and only later, when they connect to the 'true' blockchain, will they find out that they have been duped and have in reality received nothing.
- **Eclipse Attack**
    - An Eclipse Attack is a means of attacking a decentralized network through which an attacker seeks to **isolate and attack a specific user(s)**, rather than attack the whole network (as in a Sybil Attack). A successful Eclipse Attack enables a would-be bad actor to isolate and subsequently **prevent their target from attaining a true picture of real network activity and the current ledger state.**
    - This attack is made possible because a decentralized network does not let all nodes simultaneously connect to all other nodes on the network. Instead, for efficiency, a node connects to a select group of other nodes, who in turn are connected to a select group of their own. For example, **a Bitcoin node has 8 incoming connections; Ethereum 13**.
    - A malicious actor would aim to **hijack all of these connections**. The effort required to achieve this varies by the construct, size and nature of a network, but generally an attacker would have to control a botnet of host nodes (each with their own IP address) and **work out** (essentially by trial and error) the **neighboring nodes of an intended victim**. The next time the victim node logs off and then rejoins the network (resetting their connections, and forcing them to find a new set of nodes to connect to) the attacker has a good chance of being in control of all of the victim's connections.
    - **Eclipse Attack Implications**
        - Double-Spend Transactions
        - Aggravated selfish mining
        - Network-wide Denial of Service Attack
            - With 6000 reachable bitcoin nodes, a DoS attack needs 450,000 TCP connections and 600 KB of advertisement/block/20 minutes
- **Eclipse Attack Timeout**

Transactions
- After 2 minutes request from other peer (FIFO)

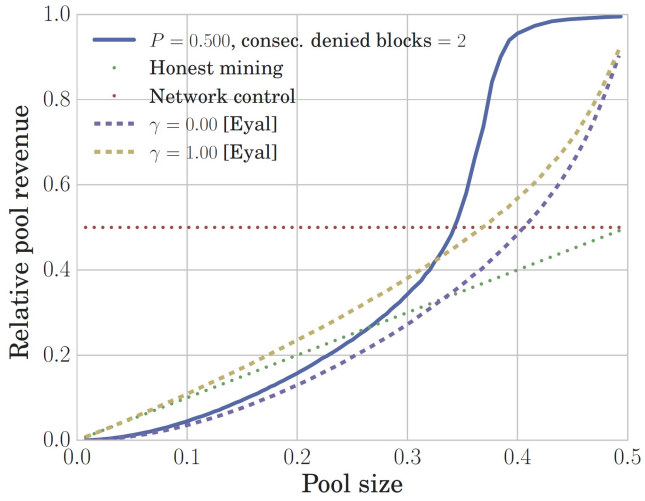
Blocks
- After 20 minutes disconnect and do nothing
- If received header, disconnect and request block from another peer

- ***BETTER TO GIVE VICTIM THE BLOCK INSTEAD OF THE HEADER ONLY!***
- **Requirement of the Victim**
  - **Not receiving the block header**
    - Under header-first synchronization, receiving the header will make the victim to request block content from others
    - **The attacker prevents the victim from receiving headers by:**
      - 1) Monopolizing the remaining open slots of the victim. The victim hence does not receive a version message and no header.
      - 2) The victim will not receive headers from existing connections (because of how the protocol works).
  - **Not receiving the version number**
    - Similar to the block sequence number. When the victim is connecting to another peer, it might realize the delayed block sent from the adversary is outdated
- **Requirement of the Adversary**
  - Be the first node that shares the newest block to the victim
  - **Connection depletion** -> Occupy all 8 incoming connections of the victim

- **P2P Network Hardening Overlay (Eclipse Attack Mitigation Techniques)**
  - **Broadcast header instead of hash(header) / inv**
    - They can verify the PoW, and know whether a chain's PoW is valid or not. As such, they know immediately if they're behind or up to date regarding floating blocks. Having said that, you're right that they still need access to the transaction to fully validate the block contents.
  - **Filtering by IP address / Randomly choosing sender**
    - Creates difficulty for adversary to control all neighboring nodes
  - **Dynamic Timeouts**
    - No 20 mins hardcoded block timeout
- **Selfish Mining**



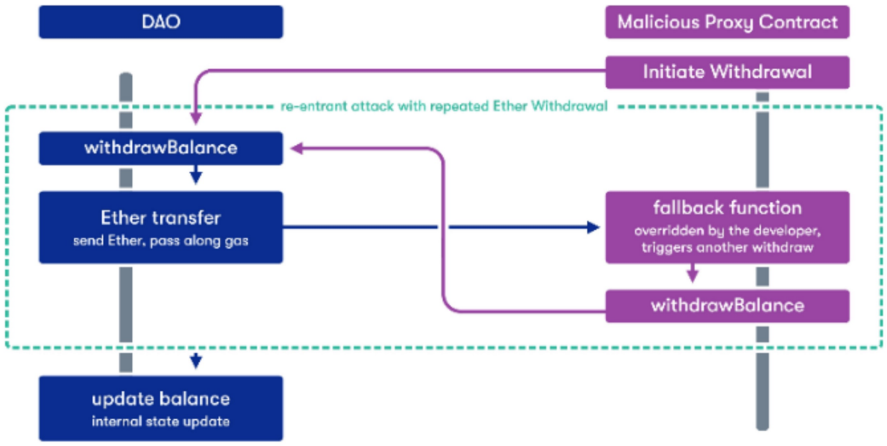
  - When both the adversary and honest network has mined one block, a racing condition happened.
  - Both blocks mined have the same difficulty, so whoever reaches more nodes in the network will be acknowledged as the legitimate block to be included in the main chain
  - **Propagation Parameter matters in here**
    - **(1-alpha)gamma:** Means the honest network mines on the adversary's block because the adversary has greater propagation power

        - **(1-alpha)(1-gamma):** Means the honest network mines on the honest network's block
        - **(alpha):** Means the adversary has mined another block and will invalidate previous honest network's block
    - **State 0, 1 and 2**
        - If the adversary is in state 3 (or any number strictly larger than 2), and the main chain finds a block, the attacker is still ahead, even if they publish only one block. Think of it this way: The attacker still secretly has the other blocks for the adversarial chain (more than the main chain will find), and can publish these at any time to make the adversarial chain the main chain.
        - This is different for the case 2: In the case 2, the attacker needs to publish both blocks if the main chain finds a block, to make the attacker's chain the main chain. This is because If the attacker only published one block from state 2, she would go to state 1. But then if the main chain finds a block, the attacker is out of blocks, and the blocks would become competing blocks.
        - The last special case is case 1: In this case, if the main chain finds a block, there will be two blocks competing (state 0'). We need to see who "wins" before the attacker can start building an advantage again.
        - So in the case >2 it doesn't matter if there's competing blocks, because the attacker can always just publish all blocks to make a longer chain and claim all the rewards. But in the cases 0,1,2, this doesn't hold, and it becomes a special case.
    - **Implication**
        - 

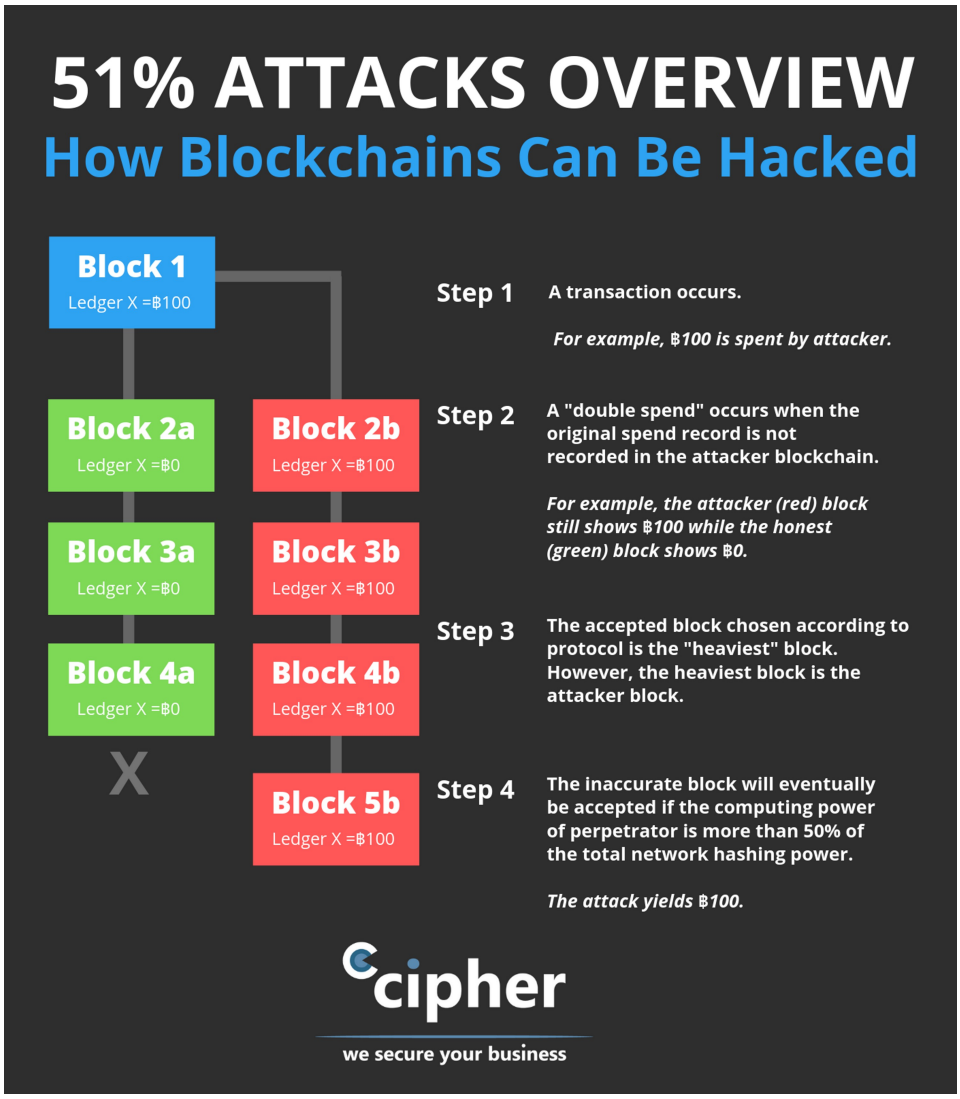
        - With honest network, a pool requires 50% mining power to receive 50% relative pool revenue
        - With selfish mining and 1.00 propagation parameter, it only requires around 33% mining power to receive 50% relative pool revenue
        - With selfish mining and consec. Denied blocks in Eclipse attack, even less amount of mining power is required to achieve 50% relative pool revenue

Blockchain Security

- **Smart Contract Security Issue**
    - **Private Variable Attack**
        - Variable does not means 'Private'. Any variable in the smart contract is **readable on the public Ethereum blockchain.**
    - **Reentrancy Attack**
        - https://quantstamp.com/blog/what-is-a-re-entrancy-attack

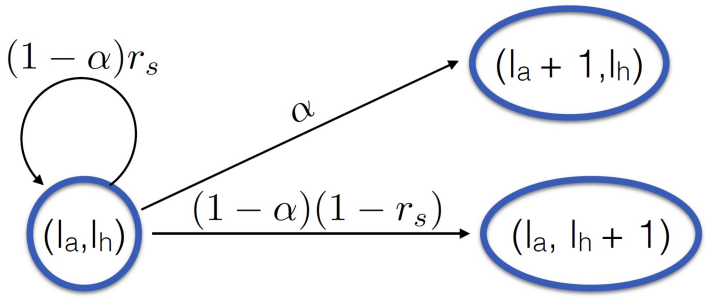


    - **Mitigation Techniques**
        - Using the **functions send() or transfer() instead of call.value()** would not allow for recursive withdrawal calls due to the low gas stipend. Manually limiting the amount of gas passed to call.value() would achieve the same result.
        - Note that the DAO contract updates the user balance after the ether transfer. **If this was done prior to the transfer, any recursive withdraw call would attempt to transfer a balance of 0 ether.** This principle applies generally—if no internal state updates happen after an ether transfer or an external function call inside a method, the method is safe from the re-entrancy vulnerability.
    - **Unreliable Timestamp**
        - No, because Ethereum computers are usually slower than real life computers. Therefore, a miner can actively choose to invent a future (i.e., mine a block) whose "random" properties will yield a favorable outcome.
- **DAO**
    - **https://www.coindesk.com/understanding-dao-hack-journalists**
    - A DAO is a **Decentralized Autonomous Organization**. Its goal is to codify the rules and decisionmaking apparatus of an organization, eliminating the need for documents and people in governing, creating a structure with decentralized control.
    - **Main Mechanism**
        - A group of people writes the **smart contracts (programs)** that will run the organization
        - There is an initial funding period, in which people **add funds to the DAO** by purchasing tokens that represent ownership – this is called a crowdsale, or an initial coin offering (ICO) – to give it the resources it needs.
        - When the funding period is over, the DAO begins to operate.
        - People then can make proposals to the DAO on how to spend the money, and the **members who have bought in can vote to approve these proposals.**
    - **DAO Attack Recursive Mechanism (Reentrancy Attack)**
        - The first thing the hacker does is **send ether (75 wei)** to the **vulnerable contract** through the **deposit function of the malicious contract.** This function calls the addToBalance function of the vulnerable contract.
        - Then, the hacker withdraws, through the **withdraw function of the malicious contract**, the same amount of wei (75), triggering the withdrawBalance function of the vulnerable contract.
        - The withdrawBalance function first sends ether (75 wei) to the malicious contract, triggering **its fallback function**, and **last updates the**

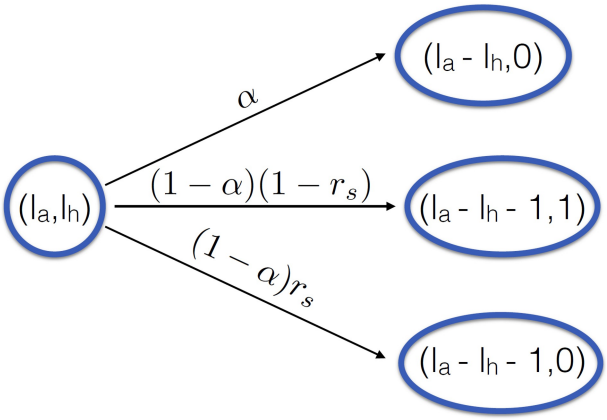
userBalances variable (that this piece is done last is very important for the attack).
    - The malicious fallback function calls the **withdrawBalance function again (recursive call),** doubling the withdraw, before the execution of the first withdrawBalance function finishes, and thus, without updating the userBalances variable.
- **Security vs. Scalability in Blockchain**
    - https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44
    - Faster Block Generation => More Forks / Stall Blocks => **Less Security**
        - 10 mins / block in Bitcoin V.S. 15 secs / block in Ethereum
        - A new block would be generated before **an old block would be received** by most of the blocks in the network.
    - Smaller Block Size => Faster Propagation => Less chances for conflict (Because more peers can realize a newly generated block faster)
- **51% Attack Rule**
    - 

    - The 51% Rule refers to a situation where an entity controls more than 51% of the computing (hashing) power within a blockchain network. The entity then creates fraudulent, yet personally validated transactions records. These records might not include previous payments leading to a double payment. **The 'Owner' of the blockchain always overwrites the forks by minor parties.**
    - The protocol of a blockchain system **validates the record with the longest transactional history**. If the attacker has more than 50% of the processing power, they will have the longest transactional history. This means that their

incorrect blocks will be the valid ones. Smaller networks are especially vulnerable to a 51% Attack. If trust is lost in a network, then the currency might crash.

- **Markov Decision Process (MDP)**
  - Markov decision process allows one to know the **optimal adversarial strategies** by comparing the performance and security of blockchain systems with different parameters. This allows one to see how likely is someone going to do honest ming compared to un-honest mining in a particular scenario
  - 

$r_a=0,\ r_h=0$

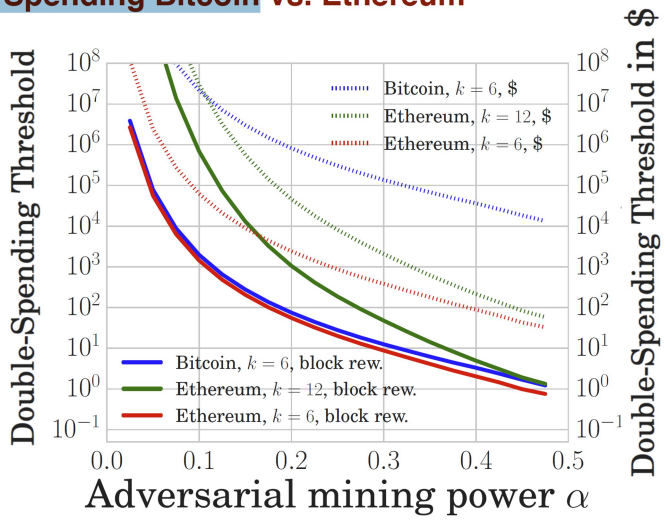
  - **(1-alpha)(1-r_s):** Honest network found a non-stale block
  - **(alpha):** Adversary found a private block - Selfish Mining
  - **(1-alpha)(r_s):** Honest network found a stale block
  - 

Reward for adversary: l_h+1

Action iif $l_a>l_h$

  - Essentially, when a honest network found a non-stale block, it will receive a profit of 1 block. Otherwise, the honest network is solely wasting its resource
- **Stale Block Rate**
  - Stale block rate is affected by two thing, **block generation rate** and **blocksize**. A higher block generation rate means a higher stale block rate since block are generated quicker, a larger blocksize means blocks are propagated slower, people receive correct block info slower more stale blocks.
- **Double Spending Bitcoin vs. Ethereum**

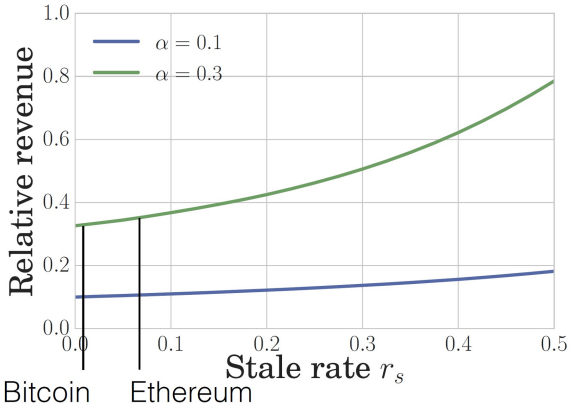
Double Spending Bitcoin vs. Ethereum



Double-spending resistance of
Ethereum (k in {6,12}) vs. Bitcoin (k=6)

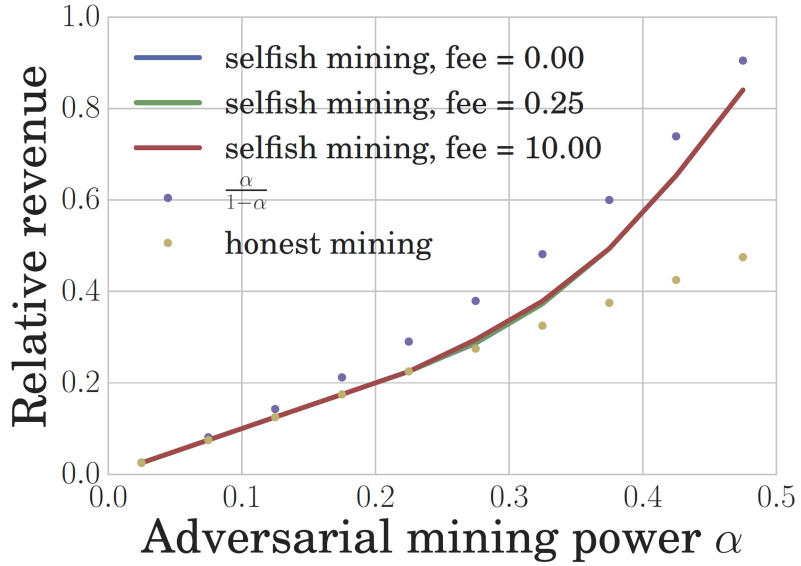
- Y-Axis represents the threshold in terms of **block rewards.** Any value below the threshold (i.e., represented by the line) will not attract adversary to double-spend.
- **Threshold decreases with growing adversarial mining power**
  - Because with a greater mining power, the double-spending is much easier to be achieved by the adversary

- **Stale Block Rate vs. Relative Revenue**



- With 0% stale rate, the relative revenue should equals to the mining power of the miner.
- **For an adversary with 0.3 mining power, under 0.3 stale rate, it has 0.5 relative revenue. This is approximately equals to owning the chain!!!**
- the higher the stale block rate is, the higher is the asynchrony among peers, and the stronger a selfish mining adversary gets, hence the higher the relative revenue grows.
  - ***Think as the honest network are mining 10 different blocks under a larger stale block rate, so the adversary is more likely to out-run the honest network when 30% of the computing power is used for computing one single block***

- **Selfish Mining Rewarding?**
  - **https://scalingbitcoin.org/transcript/milan2016/on-the-security-and-performance-of-proof-of-work-blockchains**
  - For 1000 blocks and 30% adversary mining power, selfish miner only mines 209 blocks instead of 300. That means sometimes the adversary fails to override the honest network thus receiving less rewards than the honest mining.
  - **This should discourage selfish miners from doing so.**

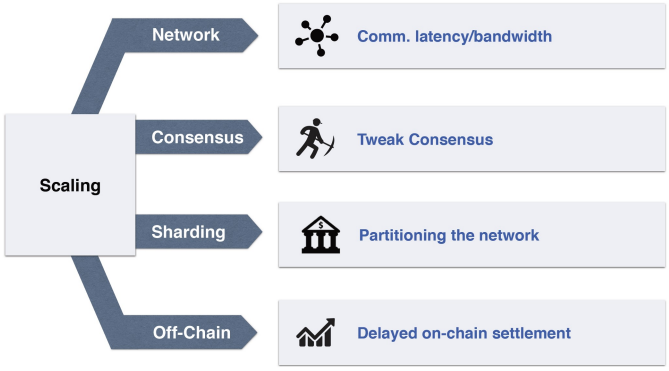
- **Bitcoin-NG and Comparison with Bitcoin**
  - **Key Block and Micro Blcoks**
    - Keyblock contains no transactions, but more like a private key that can be used for later micro-blocks.
    - Keyblock is less than 300 bytes, so with a smaller block size, it will also have a small stale block rate
  - **DANGEROUS: Most people will be attracted by Key Block, but not micro-mining.**
  - **Bitcoin-NG may seems more secure** because the adversary needs more mining power / hash rate in order to receive the 50% relative revenue
  - 

  - **Alpha / (1-Alpha):** Theoretically maximum revenue an adversary can get from selfish mining
  - Selfish mining with fee = 0.00 => Equivalent to Bitcoin; Selfish mining with fee = 10.00 => Equivalent to Bitcoin NG.
    - https://hackingdistributed.com/2015/11/09/bitcoin-ng-followup/
    - In this graph we can now that security-wise, bitcoin-NG is equivalent to bitcoin
    - There are some distinct benefits to Bitcoin-NG compared to Bitcoin, **higher throughput and lower latency.** So Performance-wise, Bitcoin-NG is better than Bitcoin, but security-wise it is equivalent to Bitcoin, then **Bitcoin-NG is strictly better than Bitcoin.**

Scalability

- **Bitcoin Scalability Issue**
  - Bitcoin's blocks contain the transactions on the bitcoin network. The on-chain transaction processing capacity of the bitcoin network is limited by the **average block creation time of 10 minutes** and the **block size limit of 1 megabyte**. These jointly constrain the network's throughput. The transaction processing capacity maximum estimated using an average or median transaction size is between **3.3 and 7 transactions per second**. There are various proposed and activated solutions to address this issue.
- **Bitcoin Limited # Txns per Second**
  - $$\#\ of\ Transactions\ per\ Block = \frac{Block\ Size\ in\ Bytes}{Average\ Transaction\ Size\ in\ Bytes} = \frac{1{,}048{,}576}{380.04} \cong 2{,}759.12$$
  - The current Bitcoin block generation time is 10 minutes; i.e., every ten minutes, a new block is mined. In ten minutes (600 seconds), Bitcoin can average around

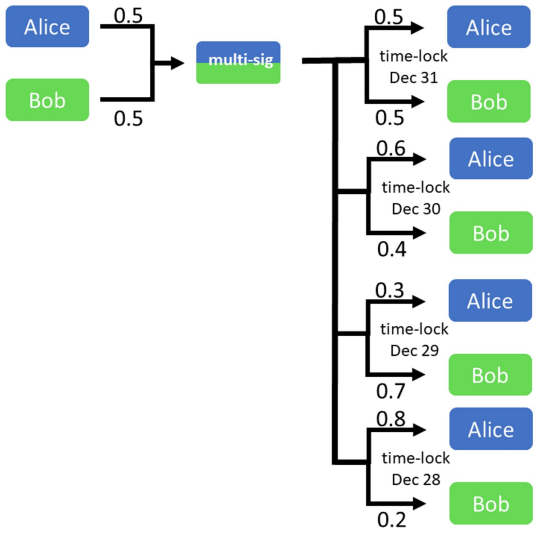
2,759.12 transactions based on previous assumptions. In other words, the Bitcoin blockchain can currently guarantee only 4.6 transactions per second.

- **Bitcoin Increase scalability based on Simulator**
  - How can we increase the throughput? Based on our simulator, we find that if we set the block size to 1 megabyte and 1 minute block interval, we do not penalize block propagation. The security is not substantially impacted. You can go to roughly 7 tps to 60 tps without sacrificing security.
- **Blockchain Layers**
  - **Hardware Layer - 1:** Signature Validation causes computing time
  - **Network Layer - 0:** Latency, Bandwidth in the P2P Network
  - **Blockchain Layer 1:** Consensus Mechanism (Proof of Work Mechanism)
  - **Blockchain Layer 2:** Overhead caused by EVM / High-Level Applications
- **Scaling Options**
  - 

- **Difference between On-Chain and Off-Chain Scalability**
  - https://bitcoin.stackexchange.com/questions/63375/what-is-the-difference-between-on-chain-scaling-and-off-chain-scaling
  - **On-Chain Scale**
    - Blocksize/blockweight increase
    - Faster blocks
    - The witness discount of segregated witness
    - More efficient new address formats
    - Smaller size of Schnorr signatures
    - Cross-input signature aggregation
    - Key aggregation
  - **Off-Chain Scale**
    - Batching multiple payments into one transaction
    - Virtual payments within the system of a custodian (Tipbots, Coinbase,...)
    - Payment Channels/Tumblebit/Lightning Network
    - Sidechains
    - Colored Coins
- **Why Off-Chain**
  - https://en.bitcoin.it/wiki/Off-Chain_Transactions
  - **Speed**
    - On-chain transactions take some time to accumulate enough confirmations to ensure that they can-not be reversed; accepting a transaction without any confirmations is potentially risky. Confirmations take time and the time they take to accumulate is random. Off-chain transaction systems can record that a transaction has happened immediately, and, subject to the guarantees of the system itself, immediately guarantee it won't be reversed.
  - **Privacy/Anonymity**
    - All on-chain transactions are recorded publicly on the block chain; Bitcoin transactions are not inherently anonymous. It may be possible for a third-

party to use the block chain transaction data to determine the source and/or destination of a transaction if they can gather enough information linking addresses to identities. Because off-chain transactions do not happen on the block chain they need not be public. Using cryptographic techniques such as chaum tokens it can be made impossible for even the operators of the system itself to determine who participated in a transaction.

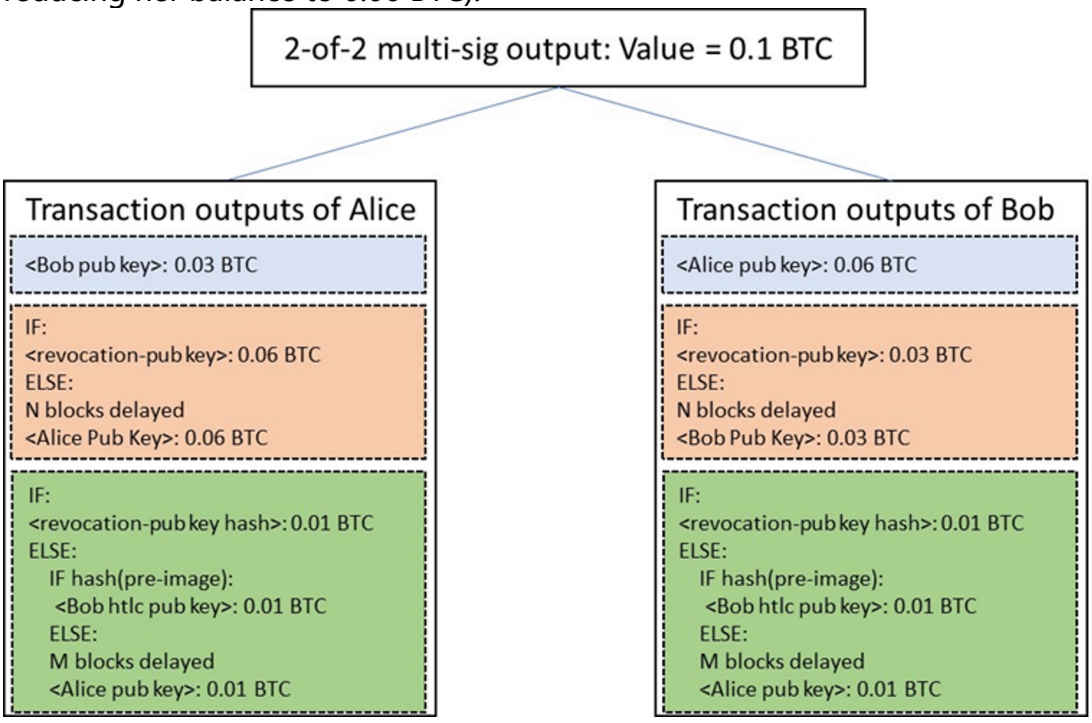
    - **Cost/Scalability**
        - Miners usually charge fees to confirm a transaction. While currently the demand for transactions is sufficiently low that fees are relatively small, and transactions can often be confirmed for free, for many applications even paying a few cents per transaction is unaffordable.[1] In addition Bitcoin currently has a limit of 7 transactions per second, the blocksize limit. This limit is related to the scalability of the system as a whole, and one option to achieve higher transaction volumes is to keep the blocksize limit as is and use off-chain transactions for lower-value transactions; with higher volumes fees for transactions done on-chain will rise due to supply and demand.
- **Payment Channels**
    - **Properties**
        - **Balance Security:** Any party can always withdraw the agreed balance on-chain with a dispute
        - **State Progression:** Any party can enforce a state transition on-chain to reach a terminal state
        - **Unanimous Establishment and Unanimous Transition**
    - Only when the channel is closed, the latest balance sheet will be submit to blockchain
    - **IMPORTANT:** Off-Chain Transactions will only be written to the on-chain when a dispute has encountered
- **State Replacement**
    - https://blog.chainside.net/understanding-payment-channels-4ab018be79d4
    - **Time Lock State Replacement**



        - A time-based channel achieves security by time-locking transactions in so that **the last state of the channel has always the lowest time-lock**, meaning it is the first one that can be broadcast on the blockchain. Every **new transaction having a lower time-lock** invalidates all the previous ones, updating the state of the channel.
        - To make such system really trust-less, it is necessary to create the **first transaction that sends back the funds to both parties** before any bitcoin is moved to the multisig, just in case one of the two parties becomes

unresponsive.
    - Such design however has an evident problem, the security of the channel is guaranteed **only until the first time-lock expires**, after **which it has to be closed** (i.e. **sending the funds to the two parties**), giving the channel a **predetermined limited life time**.
- **Time Lock Initial Time-lock Commitment and Replacement Technique**
    - The **commitment is signed before the funding transaction** to ensure that no funds can be taken hostage by one party, as **the other party already holds the means to recover its stake.** Both parties can close the channel at any time by broadcasting the prepared commitment. As the **opposing party cannot spend from the shared account without both signatures,** the funds are safe and the broadcast of the commitment can be delayed to a later point in time. Given a scheme to replace transactions, the channel can now be used to transfer funds by **replacing the commitment transaction with new commitment transactions**, which change the amount of currency sent to each party, as shown in figure 2. It is important to **ensure that the old version of the commitment transaction cannot be used any more**. We will look at methods to accomplish this in the next subsections.
- **Time Lock State Replacement Example**
    - Alice sends 1 BTC to a 2-of-2 multisig contract between Alice and Bob.
    - To send 0.1 BTC to Bob, Alice creates and signs a transaction that gives 0.9 BTC to herself and 0.1 BTC to Bob and sends it Bob.
    - To send other 0.2 BTC in a second moment Alice updates the state of the channels creating another transaction that send 0.7 BTC to herself and 0.3 to Bob.
    - Alice cannot broadcast to the network and have included on the blockchain any of those transactions, since she does not have the required signature of Bob (remember that the funds are locked in a multisig contract).
    - Bob is **always incentivised to broadcast only the last state of the channel,** since it represents the outcome where he receives more bitcoins.
    - To protect Alice from the **risk that Bob is unresponsive** and does not cooperate to broadcast any state of the channel (blocking also Alice's funds), a **time-locked transaction that refunds Alice of one BTC is created at the opening of the multisig contract.** The time-lock is needed to make sure that **Alice can use this transaction only in the case Bob is unresponsive.**
    - Before the end of the time-lock period, Bob will **broadcast the last state of the channel** in order to avoid the risk of Alice broadcasting the time-locked transaction.
- **Revocation State Replacement**
    - A different approach to create channels with **no expiry date** is to base the security on the **punishment of a malevolent counterparty** rather than on time. The idea is still to lock funds in a multisig contract between the two parties involved in the channel, but to guarantee that the right behaviour is respected both parties sign transactions that spend to a more complex smart contract design to make transaction replacement secure. Such payment channel design is what is used for Lighting Network implementations.
- **Revocation State Replacement Example**
    - **https://bitcoin.stackexchange.com/questions/91211/how-can-someone-lose-funds-in-lightning-network**
    - When two parties open a channel in Lightning Network what they

essentially do is send some bitcoins to a 2-of-2 multi-sig that they **both control** (in current specs only one party can send it). This 2-of-2 multi-sig UTXO is then consumed as an input to **create commitment transactions where the outputs pay out the bitcoins to the two parties based on who is owed what (just like a balance sheet).**

- However, the commitment transaction outputs are asymmetric. Below is an example of commitment transactions that Alice and Bob will hold for their channel with total capacity of 0.1 BTC. Alice currently has 0.07 BTC and Bob has 0.03 BTC in terms of their balance in the channel. The below commitment transaction state also considers the case where **Alice is asking Bob to add a HTLC of 0.01 BTC** (thus reducing her balance to 0.06 BTC).
- 

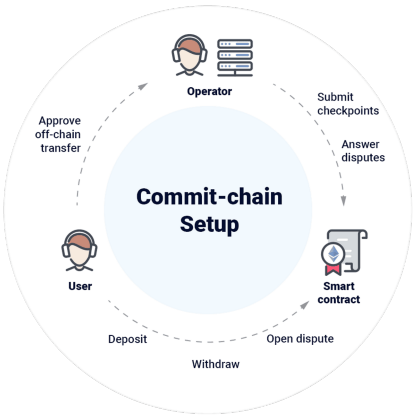
- As you can see from the infographic above, the **outputs that pay to self are guarded by the revocation keys**. It pays the revocation key immediately but pays the self's public key after a delay of say N blocks. Whenever, Alice and Bob will sign a new commitment transaction, **they will revoke the previous commitment state by sharing their side of the revocation secret with the other.** Thus, if one of the party tries to broadcast the previous state of the channel, the other party can take the entire balance of the channel by using the revocation secret (until N blocks after which both parties can spend). This deters them from broadcasting a commitment transaction that reflects the previous state of the channel **due to the risk of losing their entire funds from the channel.**
- Revocation is a double edged sword. Both the parties **must keep the revocation keys** that were shared for the previous states and all of the commitment transactions. If one of them is lost, the other party can possibly cheat the other. This is especially scary when your machine crashes making you to lose your backups.

- **Synchronization**
  - **Hash Time Locked Contracts**
    - A Hash Time Locked Contract or HTLC is a class of payments that use hashlocks and timelocks to require that the receiver of a payment either **acknowledge receiving the payment prior to a deadline** by **generating cryptographic proof of payment** or forfeit the ability to claim the

payment, returning it to the payer.

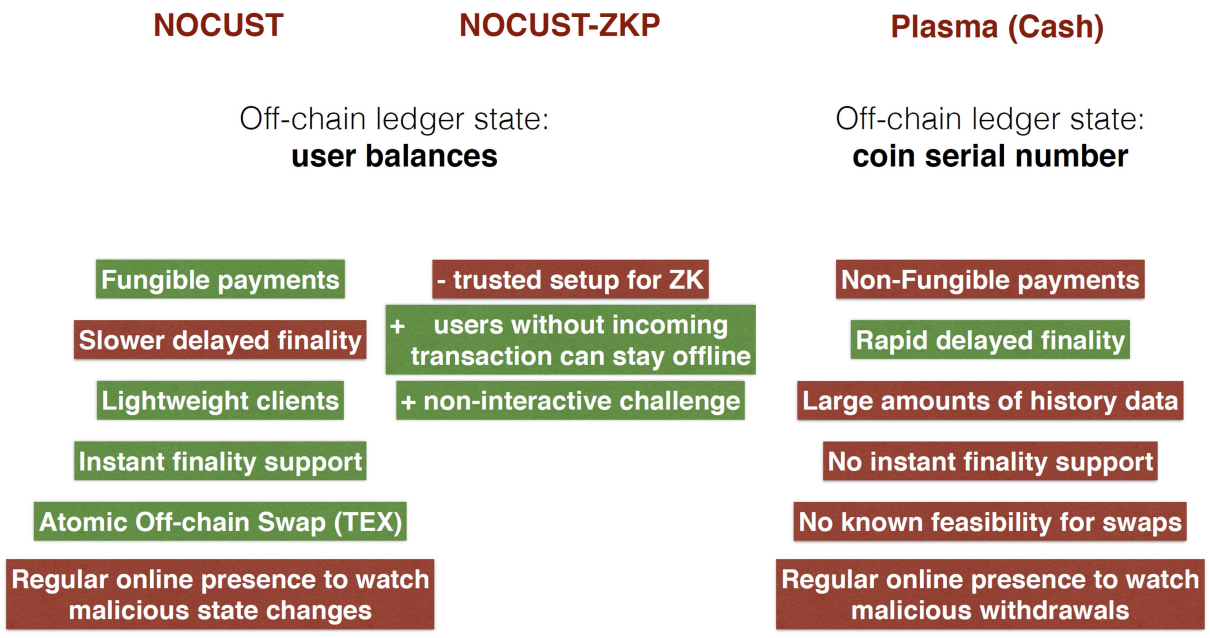
- **Usage Example**
  - Alice opens a payment channel to Bob, and Bob opens a payment channel to Charlie.
  - Alice wants to buy something from Charlie for 1000 satoshis.
  - Charlie generates a **random number** and **generates its SHA256 hash**. Charlie gives that hash to Alice.
  - Alice uses her payment channel to Bob to pay him 1,000 satoshis, but she **adds the hash Charlie gave** her to the payment along with an extra condition: in order for Bob to claim the payment, he has to **provide the data which was used to produce that hash**.
  - Bob uses his payment channel to Charlie to pay Charlie 1,000 satoshis, and Bob adds a copy of the same condition that Alice put on the payment she gave Bob.
  - Charlie has the original data that was used to produce the hash (called a **pre-image**), so Charlie can use it to **finalize his payment and fully receive the payment from Bob**. By doing so, Charlie **necessarily makes the pre-image available to Bob.**
  - Bob uses the pre-image to **finalize his payment from Alice**

- **Routing**
  - **Scalable**:
    The routing algorithm should remain effective and efficient for large-scale PCN and high transaction rates.
  - **Effectiveness**:
    Given the network topology, the routing algorithm should find the path that will likely be successful.
  - **Efficiency**:
    The overhead of path discovery should be low in terms of communication, latency, and computation.
  - **Source-Routing**
    - Topology is known, and the sender computes the best path
  - **Per-hop Routing**
    - Each hop is aware of the final destination and they choose the next hop
    - **Not Transfer-Privacy (i.e., the final destination is known for every hop in the path)**
- **Payment Channel Security Assumption**
  - The blockchain is functioning well (confirming transactions quickly)
  - Channel nodes can keep secret data safe
  - There aren't any significant bugs in the software
- **Payment Channel Evaluation**
  - Decentralized, limited censorship
  - No direct connection needed (Routing)
  - Optimistically fast and cheap (Keep the channel indefinitely open with revocation protocol)
  - On-chain channel establishment
  - Collateral for each hop: Need to lockup assets
  - Wormhole Attacks Vulnerability
- **Payment Channel Drawbacks**
  - Require all parties along the payment route to **be online simultaneously**
  - Payment amount is limited by **capacity of channels** along the payment route
  - May tend toward **"hub" model with less decentralization** (easier to set up channel with a well-connected node, like an exchange etc)
  - May take a while to **settle in the case of dispute**

- **Payment Channel Maximum Capacity**
  - The **amount of locked funds determines the maximum imbalance between sent and received funds**, until all funds are with a single partner only. This is the **capacity of the channel.** When a channel's capacity is depleted, currency must move in the other direction or the channel needs to be closed and reopened on the blockchain with additional funds.
- **Trusted Execution Environment (For Commit-Chain)**
  - Allows efficient and fast payments among peers by reserving an invisible hardware unit in the CPU.
  - Peers can remain **offline** while receiving transactions. When the node wake up, it will know node 1 has sent a transaction
    - **IMPORTANT:** For Bitcoin / On-Chain Transactions, we only need the recipient address (public key) to send the transaction. For Off-Chain Transactions, we need the IP address to establish a off-chain transactions with the peer
- **Payment Channel Hub**
  - Great choice for instant finality and no trust, but very expensive to run
  - Each user needs to use collateral to open channels and transactions
  - **Benefit**
    - Most efficient solution comparing to a fully decentralized network. The Payment Channel Hub needs the least amount of payment channel established
- **Commit-Chains**
  - https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/commit-chains/
  - Commit-chains are the generic term for what is also called 'Plasma' - a layer 2 scaling solution for Ethereum and other blockchains.
  - Commit-chains, also sometimes described as **non-custodial side chains**, don't introduce a new consensus mechanism like side chains - they rely on the **parent-blockchains consensus** which makes them **as safe as the parent-blockchain itself.**
  - In commit-chains, **untrusted and non-custodial operators** facilitate the communication between transacting parties. The operator is expected to **commit the state of user account balances** by sending periodic updates to the parent-blockchain.
  - 

  - **Unlike payment channels,** commit-chains are on an always ongoing state once launched and don't rely on a three-state - opening, live, dispute/closure - model.
  - After an operator launches a commit-chain, users can join and conduct transactions that are recorded on the commit-chain while keeping the freedom to withdraw or exit their assets to the parent-blockchain at any time.
  - **Periodic Checkpoint Commitments**

        - Commit-chain users may need to periodically observe on-chain checkpoint commitments, which can be instantiated as a Merkle tree root or Zero-Knowledge Proof (ZKP).
        - While ZKPs enforce consistent state transitions on-chain, Merkle root commitments do not, requiring users to participate in challenge-response protocols to challenge operator misbehavior.
    - **Data Availability Requirement**
        - Users must retrieve and maintain data required to exit a commit-chain since data isn't broadcasted.
        - Depending on the implementation, if the data is unavailable, you are either forced to exit (like in Plasma) or the operator can be challenged to provide the necessary data (like in NOCUST).
        - On misbehavior, users are allowed to exit with their last confirmed balance.
    - **Centralized but Untrusted Intermediary**
        - The **centralized operator never holds custody of funds** so if the operator is not available the worst-case scenario is that users are just **unable to make any further off-chain transactions**, but they can exit and move to another commit-chain at any time.
    - **Transaction Finality (Eventual Finality ONLY! NOT INSTANT FINALITY)**
        - **Unlike payment channels, the commit-chain operator does not require on-chain collateral to securely route payments between users.**
        - Commit-chain transactions **do not offer instant transaction finality** like payment channels but offer **eventual finality after an on-chain checkpoint**.
        - However, if an operator chooses to allocate collateral to each user, essentially implementing a payment channel on top of a commit-chain, instant transaction finality becomes possible.
    - **Reduced Routing Requirements**
        - A single commit-chain can potentially host millions of users, so a few statically connected commit-chains are envisioned to spawn stable networks with low routing complexity. Atomic cross commit-chain transactions have not been proposed yet.
- **NOCUST Commit Chains**
    - NOCUST is an **account-based commit-chain** where an **on-chain address is associated to a commit-chain account.**
    - The NOCUST on-chain contract expects to periodically receive a constant-sized commitment to the state of the commit-chain ledger from the operator containing each user's account in the collateral pool.
    - **Free establishment**
        - Users can deposit any amount of coins within the contract and perform commit-chain payment of any denomination towards other users and with free establishment, users can join the commit-chain without on-chain transaction by requesting an operator signature and immediately receive commit-chain transactions.
    - **Agreed transition**
        - A transaction within NOCUST is enacted with the signature of the sender and the operator to deter potential double-spend scenarios.
    - **Instant transaction finality**
        - State progression is only possible if the operator stakes collateral towards the recipient. NOCUST specifies a mechanism to allocate collateral towards all commit-chain users within a constant-size on-chain commitment, which enables instant transaction finality for specified amounts.

        - Allocated collateral is reusable after each on-chain checkpoint and at this point, the transaction throughput is only limited by the operator's bandwidth and computational throughput - independent of checkpoint commitment interval.
    - **Commitment integrity**
        - Each user is only required to verify their own balance proof by requesting data from the operator and comparing it with their locally stored state at regular time intervals to observe integrity.
        - In the case of misbehavior, a user can issue a challenge using the NOCUST smart contract. If the operator comes back with invalid information or does not respond, users have an accountable proof of misbehavior.
        - NOCUST supports a provably consistent mode of operation through zkSNARKS. Layer two-state transitions will be validated by the underlying smart contract and the operator is unable to commit invalid state transition without being halted.
- **NOCUST Security Issues**
    - **Server Disappear**
        - On-chain committed transactions will be secured, whereas non-committed transactions will be covered by insurance pool
        - Server might colluded with clients in order to attempt double-spending. The adversary will be challenged if 1) Attempted to create coins 2) Attempted to steal coins.
            - Smart Contract will challenge because the balance doesn't match, and will terminate
- **Comparison between Plasma Cash and NOCUST**
    - Plasma is a **UTXO-based commit-chain** while NOCUST is **account-based.**
    - In Plasma Cash, a coin is minted with an on-chain deposit and cannot be merged or split with another coin on the commit-chain , hence it is useful for non-fungible tokens but not practical as a payment system.
    - NOCUST uses ZKPs to enforce **consistent state transitions on-chain**, Plasma Cash uses **Merkle root commitments**, which do not and require users to participate in challenge-response protocols to challenge operator misbehavior.
    - 

- **Advantages and Disadvantages of Commit Chains**
    - **Recipient can be offline to receive a transaction:** Based on hardware
    - **Instant and Free onboarding:** Join the network for free and fast

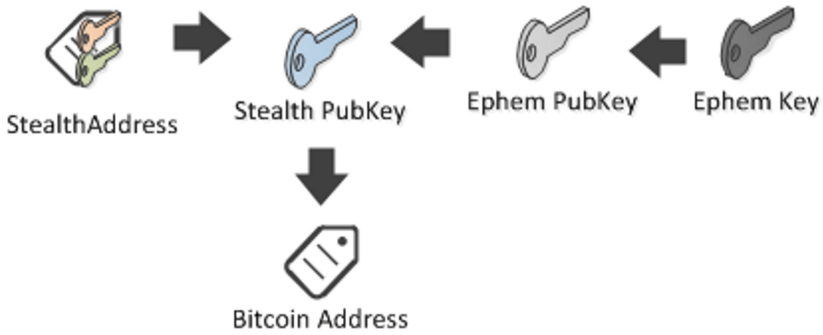
    - **No collateral for delayed finality:** Wait till the checkpoints
    - **No Decentralization:** Centralized operator is required
        - Centralized operator needs to **sign off** transaction from users, which can be censused
    - **Censorship Resistance:** The centralized operator can censor the transactions in details
    - **Data Availability Challenges:** Need to wait until the operator responding to the client

Blockchain Privacy

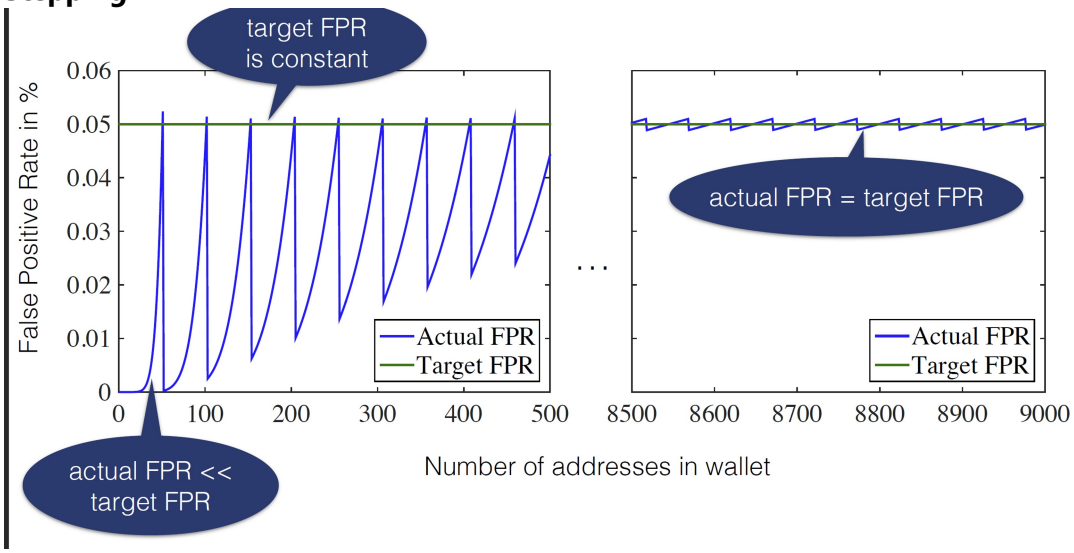
- **Ethical Concerns**
    - Anti-Money Laundering Support
        - Zcash enables 'selective disclosure' for tax and AML purposes
        - Tornado Cash allows 'Compliance Tool'
- **Linkability and Traceability**
    - Linkability refers to the ability to link transactions that go into a recipient address
    - Traceability refers to the ability to trace origin of funds
- **TxProbe**
    - **https://arxiv.org/abs/1812.00942**
    - Edge Inferring Technique relying on mempool content to infer if two nodes are connected to each other.
- **UTXO Model in Bitcoin**
    - Each transaction is traceable and linkable, which we can both find where
- **Zcash**
    - Fork from Bitcoin
    - **Address Categories**
        - Transparent Transactions (t-addr) => Equivalent to Bitcoin transactions
        - Shielded Transactions (z-addr) => Information is hided
    -

	Shielding	De-shielding	Shielded
Source	t-addr	z-addr	z-addr
Destination	z-addr	t-addr	z-addr
Amount	Public	Public	Private

    - **Z-address will not reveal #coins entering nor # coins exiting.** Use Zero-Knowledge Proof to ensure the transaction is valid
- **Monero**
    - Unlikability: Stealth Address => Ensure the destination will not be known
    - Untraceability: Ring Signatures
- **Stealth Address**
    - **Hierachical Deterministic (HD) Wallets**
        - Multiple Addresses derived form the same key (i.e., the Key Phrase initiated at the beginning)
        - From one address cannot deduce / derive another address that are generated by the same key phrase
    - **Monero Stealth Address**
        - **https://web.getmonero.org/resources/moneropedia/stealthaddress.html**
        - **https://programmingblockchain.gitbook.io/programmingblockchain/key_generation/stealth_address**
        - **Requirements**
            - The Payer knows the StealthAddress of the Receiver.

        - The Receiver knows the Spend Key, a secret that will allow him to spend the coins he receives from such a transaction.
        - Scanner knows the Scan Key, a secret that allows him to detect the transactions that belong to the Receiver.
    - **Steps**
        - StealthAddress is composed of one or several Spend PubKeys (for multi sig), and one Scan PubKey.
        - 

        - The payer, will take your **StealthAddress**, generate a temporary key called **Ephem Key** and will generate a **Stealth Pub Key**, from which the Bitcoin address to which the payment will be made is generated.
        - Then the payer adds and signs the inputs, then sends the transaction on the network to this new bitcoin address **(stealth)**
        - The **Scanner** knowing the **StealthAddress** and the **Scan Key** can recover the **Stealth Pub Key** and the **expected BitcoinAddress payment.**
        - Then the scanner checks if one of the outputs of the transaction corresponds to that address. If it does, then **Scanner notifies the Receiver about the transaction.**
          The Receiver can then get the **private key of the address with their Spend Key.**
- **Ring Signatures**
    - **Anonymity**
      An adversary cannot identify which ring signature corresponds to which of the public keys in the ring.
    - **Unforgeability**
      An adversary cannot produce a valid signature, if it does not know a secret key corresponding to a public key included in the ring.
    - **Exculpability**
      An adversary cannot produce a valid signature that links to the signature of another member of the ring, whose key the adversary does not control.
    - **Linkability**
      Any two signatures produced by the same signer within the same ring are publicly linkable (i.e., anyone can detect that they were produced by the same signer).
        - Verify against a set of public keys. Computationally infeasible to determine which of the group members signed.
        - - Groups can be formed on an ad-hoc basis (vs. group signatures)
        - - O(n) for the resulting signature size
            - n == number of public keys
        - - Does not hide transaction amounts!
- **Cash Join**
    - **Advantages**
        - Hides / Harder to infer the originator of a transaction

    - **Disadvantages**
        - Require trusted third party because no spontaneous mixing of transactions
        - Cannot hide amounts (i.e, 5 BTS from Frank most likely go to Eve with 3BTC output)
        - **Samll Anonymity Set =>** Determined by the size of the Joined parties in the cash join
- **Tornado Cash**
    - Each user has to only deposit a fixed amount into the pool.
    - Each user that wants to withdrawal has to claim its right to retrieve fund from the pool with a fixed amount
    - **Advantages**
        - Hide the originator of the transaction
        - Increasing anonymity set (i.e., a Hub)
        - No coordination among users
    - **Disadvantage:**
        - Transaction cost has to be paid from a neutral/new address to avoid linkability
        - No user-friendly amounts
- **Bloom Filter**
    - **Stair Stepping**
        - 

        - It will cause security concerns because for users with less than 500 address, the actual FPR is significantly smaller than the target FPR, meaning that the users' address cannot be hidden among 33000 address anonymity
    - **Resizing / Reseeding**
        - A new bloom filter will generate a new false positive rate
    - **Observation**
        - Using two bloom filters with resize / restart or > 2 filters yield almost 98% possibility in correctly guessing 1 address, leaking privacy
            - 1 Bloom filter critical if <20 bitcoin address
            - 3+ bloom filter intersection attack is very strong
        - **We need constant false positive rate, SPV client needs to store the state (i.e., avoid re-seed)**
    - **Proposed Solution**
        - Use multiple bloom filters that contains different sets of address, that will not cause resize

Exam Questions and Answers

- **Difference between Cryptographical Hash & Hash Function**

    - Every cryptographic hash function is a hash function. But not every hash function is a cryptographic hash.
    - A cryptographic hash function aims to guarantee a number of security properties. Most importantly that it's **hard to find collisions or pre-images and that the output appears random**. (There are a few more properties, and "hard" has well defined bounds in this context, but that's not important here.)
    - Non cryptographic hash functions just try to **avoid collisions for non malicious input**. Some aim to detect accidental changes in data (CRCs), others try to put objects into different buckets in a hash table with as few collisions as possible.
- **Interplay between Block Size and Decentralization**
    - Larger blocks make full nodes more expensive to operate.
    - Therefore, larger blocks lead to less hashers running full nodes, which leads to centralized entities having more power, which makes Bitcoin require more trust, which weakens Bitcoins value proposition.
- **What happens if two transactions use the same previous transaction output as input**
    - **https://en.bitcoin.it/wiki/BIP_0030**
    - **Motivation**
        - So far, the Bitcoin reference implementation always assumed duplicate transactions (transactions with the same identifier) didn't exist. This is not true; in particular **coinbases are easy to duplicate, and by building on duplicate coinbases, duplicate normal transactions are possible as well.** Recently, an attack that exploits the reference implementation's dealing with duplicate transactions was described and demonstrated. It allows reverting fully-confirmed transactions to a single confirmation, making them vulnerable to become unspendable entirely. Another attack is possible that allows forking the block chain for a subset of the network.
    - Blocks are not allowed to contain a transaction whose identifier matches that of an earlier, not-fully-spent transaction in the same chain. This rule is to be applied to all blocks whose timestamp is after a point in March 2012
- **Why always change bitcoin address**
    - This is done to protect your privacy, so that a third-party cannot view all other transactions associated with your account simply by using a blockchain explorer. All addresses that have been generated for your account will remain associated with your account forever.
- **Transaction Throughput Limitation Factors**
    - the communication complexity it takes for nodes to reach a consensus
    - the number of nodes that need to reach consensus
    - the communication latency/bandwidth among the nodes reaching a consensus
- **What is encrypted in Bitcoin/Ethereum**
    - **Nothing. Even the Signature is not encrypted.**
    - Ethereum (P2P network is encrypted)
- **What is cryptographically signed in Bitcoin/Ethereum? Why is signature necessary**
    - Digital signature is used in Bitcoin to provide **a proof that you own the private key without having to reveal it** (so proves that you are authorized to spend the associated funds). The digital signature, additionally, makes sure that a transaction cannot be modified by anyone after signed.
- **How much data is required to demonstrate that a leaf node is part of a given hash tree**
    - The height of the hash tree (Including the root node)
    - But when a SPV client wants to prove a transaction is included by asking another full node, it only needs height - 1 hashes because the root node information is

already included in the header.

- **Other use cases of the merkle tree?**
  - Bittorrent
- **CoinJoin Question**
  - **Technical Details of How CoinJoin Would Operate (Find Peers)**
    - A centralized officer receive two inputs from two persons, and join them together
  - **Why CoinJoin would improve user privacy**
    - Mix inputs with outputs so that the transaction cannot be easily traced
  - **Problems**
    - Still traceable because the **amount is still visible.**
      - **We can reverse engineer to calculate who paid whom**
    - Centralized service has all the information. (Like a bank)
  - **Blockchain Solution**
    - Fix amount!! Everyone has to only send 1 bitcoin in each transaction. This will reduce usability
- **Fork**
  - **Increase Block size**
    - It is a hard fork because the validity set has increased. Because for nodes that do not update with the protocol, they will treat the increased block size as invalid
  - **chainID field**
    - If replace an existing field / identifier, it will be a soft fork
    - If add a new field in the header, it will have to be a hard fork
- **Uncle Block**
  - **https://www.investopedia.com/terms/u/uncle-block-cryptocurrency.asp**
  - Uncle blocks are created on Ethereum-based blockchains, and they are similar to Bitcoin's orphan block.
  - In a process similar to the way Bitcoin creates orphan blocks, uncle blocks are created when more than one child block is created from a parent block. This situation is possible because all the nodes that maintain the ledger are not updated instantaneously when a new block is mined. Instead, you may have two blocks mined close together, but only one gets validated across nodes on the ledger. The one that is not validated is an uncle block.
  - **Incentivization Scheme**
    - To **increase the number of transactions** on the blockchain, Ethereum allows for the creation of more uncle blocks as a byproduct of shorter block times.
    - Valid uncle blocks are rewarded to neutralize the effect of network lag on the distribution of mining rewards.
    - Incentivizing uncles helps to **decrease the centralization of incentives** where large mining pools with high computing power end up claiming the majority of the rewards leaving nothing for individual miners.
    - It also increases the security of the network by **supplementing the work on the main blockchain** by the work done in mining uncle blocks.
- **GHOST Protocol in Ethereum**
  - The uncle blocks are purposefully incorporated into Ethereum's consensus method by a process called "GHOST: Greedy Heaviest Object Sub Tree. **nodes will get the number of uncles mined for the last seven blocks in each subtree.** That number is, in addition to the number of blocks in that subtree, used to calculate the tree's weight; the heaviest tree is then said to be the 'correct' one. **This particular use increases the security**

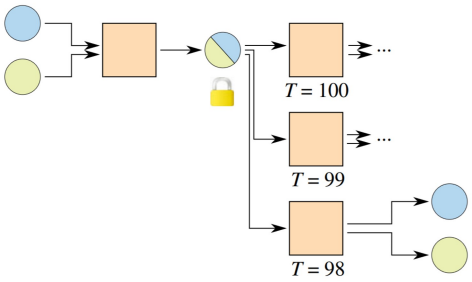
- **Invalidation Tree**
  - https://www.researchgate.net/publication/327299704_Scalable_funding_of_Bitcoin_micropayment_channel_networks
  - 

Figure 3. Micropayment channel with timelocks. The commitment with the lowest timelock can be included in the blockchain before the others.

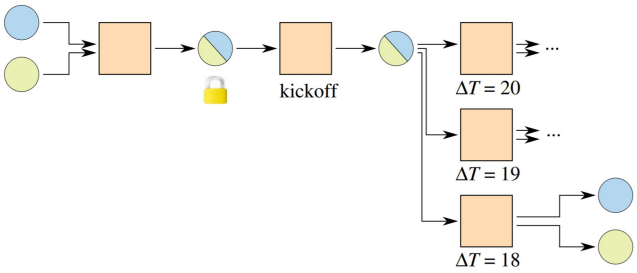
  - 

Figure 4. Micropayment channel with relative timelocks. Timelocks count relative to the inclusion of the previous transaction into the blockchain. No counters start until the kickoff transaction has been broadcast.

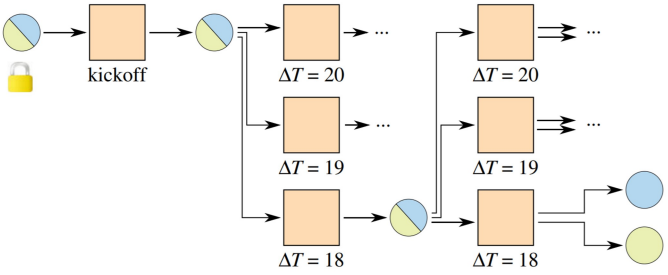
  - 

Figure 5. Invalidation tree with relative timelocks. The lowest path is the currently active one. The rest of the tree can be pruned, as it will never be valid.

  - A channel constructed this way has to **be closed by broadcasting the newest commitment transaction as soon as the first timelock has elapsed**, limiting the maximum lifetime of a channel.
  - With relative timelocks [7,8], this problem can be solved elegantly. Figure 4 introduces a **kickoff transaction (**This is just a regular transaction without any special script. Its inclusion in the blockchain just 'kicks off' the timers on the subsequent transactions, as they are relative to the previous transaction).
  - Timelocks only start ticking as soon as the kickoff transaction is broadcast, resulting in a potentially unlimited lifetime of a channel. Still, one quickly runs out of time by doing transactions in the channel, **each requiring a smaller timelock on the commitment transaction.** This was solved with a tree of transactions [2] as shown in figure 5.2 At any point in time **only the path where all transactions have the lowest timelock of their siblings can be broadcast.** In this way, many commitment transactions can be created before the timelocks get too low and the channel cannot be updated any more.
- **Web Wallet and Private Keys (12-word Recovery Phrase)**
  - https://support.blockchain.com/hc/en-us/articles/360000951966-Public-and-private-keys
  - The security of this system comes from the one-way street that is getting from the private key to the public address. It is **not possible to derive the public key from the address**; likewise, it is **impossible to derive the private key from the**

public key. In the Blockchain.com Wallet, your **12-word recovery phrase is a seed of all the private keys of all the addresses generated within the wallet.** This is what allows you to restore access to your funds even if you lose access to your original wallet. Using the backup phrase will copy over your private keys to a new wallet, essentially creating an exact replica of your original wallet, complete with used addresses and transaction history.

Solidity Coding Knowledge

- **Function Scope**
  - A public function can be accessed from anywhere, from outside the contract or internally. public is the most permissive visibility parameter for functions.
  - A private function is not accessible from outside of the contract, corresponding to the most restrictive permission you can set.
  - An external function is exposed only to the outside of the contract. You can't call the function from within your contract, while other contracts can enter the external function for communication purposes.
  - If internal, a function is not exposed outside of the contract. This appears similar to the private visibility, but they are not equal due to inheritance specificities as we will see later.
- **State Mutability**
  - If a function does not modify the blockchain state, it can be declared as a view function. The view keyword corresponds to how a function interacts with a contract and its variables. The visibility parameters are different to the degree that they define who can interact with a function. As such, the view keyword and visibility parameters are complementary.
  - If it doesn't use any internal variables of the contract, either for a read or a write, then it's pure.
    By default, a function can access and modify all contract's components. We declared the like function, for example, with a default state mutability, because it accesses and modifies the variable fans.
- **Payable**
  - In Solidity you can pay ETH to a contract function, but not directly the contract. In your message msg, there will be a special attribute called value which represents the amount of ETH you have sent to the contract. **(msg.value)**
- **Balance**

  Sending `ETH` is great, but how can we find out the balance that other addresses are holding? An address variable can access `.balance` to know how much the address holds. You can concert a contract to an address using the `address()` function.

  ```
  // Address balance
  address a = 0x0;
  uint balanceA = a.balance;

  // Contract address from within the contract
  address c = address(this);
  uint balanceC = c.balance;

  ```

- **Sender**
  - User can sell their PichuMuffin in exchange for cryptocurrency. The keyword **msg represents the transaction calling the contract**, and one of its attributes is sender. **msg.sender is therefore an address associated to the user that sent the message/transaction.** To transfer funds to msg.sender (or to any other address), you should use <publickey>.transfer(<amount>). This will check if the sending contract has enough funds and send those funds to msg.sender (or an other address).

```
pragma solidity ^0.4.24;

contract PichuMuffin {
  uint public lastTip = 0;

  // we shall talk about your poor security
  modifier securityCheck(bytes32 _password) {
    require(_password == "Super Super Muffin");
    _;
  }

  function tip() public payable {
    require(msg.value > 0);
    lastTip = msg.value;
  }

  function retrieve(bytes32 _password) public securityCheck(_password) {
    // ok, it's a one liner
    // but you're a one clicker
    msg.sender.transfer(address(this).balance);
  }
}
```

        - ***This will transfer the amount in the contract to msg.sender***
    - **Avoid at all costs using call() or send() to limit the gas consumption and hence avoid critical reentrancy vulnerabilities in your code.**
- **Contract Address**
    - When you deploy your contract on the Ethereum blockchain, it is published at a **given address (determined by the hash of the deployer address and its account nonce)**. All your contracts, the PichuMuffinFactory, PichuCookie and PichuMuffin have each a unique address, i.e., an identity for the contract.
    - An Ethereum address looks like this: 0xe1a15b265e58b7be2e8a626aee8e27b5d288a1b7. It is a hexadecimal string of length 42 (20 bytes), starting with 0x followed by letters and numbers. Because it is complex to type, and could be mis-typed, Solidity enforces each address to have a checksum. To associate a muffin variable to an address, you simply have to do the following: ***PichuMuffin muffin = PichuMuffin(0xe1A15B265e58B7bE2E8A626AeE8e27b5d288a1B7);***
    - Because the PichuMuffin is unique, your PichuCookie wants to be unique too. To access a contract address, you can request ***address(this)*** from within the respective contract. this is a keyword accessible from within the contract (similar to classes in object-oriented programming languages).