|  |
| --- |
| lec 1: bitcoin protocol and consensus |
| Blockchain: a method of storing data amongst multiple parties that ensures data integrity  immutability and transparency  “distributed ledger,” or a shared database where everyone holds a copy |
|  |
| Each entity is represented with a unique public key |
| Public key for receiving, private key for redeeming |
| What makes a transaction valid?  ○ Proof of ownership (a signature)  ○ Available funds  ○ No other transactions using the same funds |
| Gloria sends 4 BTC to Brian by:  Redeeming her UTXO (unspent transaction outputs) containing 5 BTC  Sending 4 BTC to Brian  And sending 1 BTC back to herself |
|  |
| Proof of Work:  ● In order to propose a block, must include Proof-of-Work or the solution to a hash puzzle  ● Hash puzzle can only be solved using brute-force computation, which spends resources  ● People who create and propose blocks are called miners |
|  |
| lec 2: BITCOIN TO BLOCKCHAIN |
| Solution to distributed consensus: Proof-of-Work, “one-CPU-one-vote” |
| ● Rise of interest in "private blockchains" or "permissioned ledgers."  ○ Not open  ○ Not trustless  ○ No economic incentives like in Bitcoin  ○ Separate "blockchain" from "Bitcoin"  ● Con:  ○ Glorified public key cryptography  ● Benefit:  ○ More compliant |
|  |
| lec 3: bitcoin mechanics and optimization |
| Cryptographic hash function:  A hash function with three special properties:  ● Preimage resistance  ● Second preimage resistance  ● Collision resistance |
| Avalanche effect |
|  |
| merkle root: complete binary tree    take O(logn) to check a transaction |
| Hash puzzles need to be:  1. Computationally difficult.  2. Adjustable  3. Easily verifiable. |
| PARTIAL PREIMAGE HASH PUZZLE  find nonce satisfies |
|  |
| checksum: 4-byte error-checking code appended to the end of an address |
| The "hash" of a block is actually only the hash of the block header, a roughly 200-byte piece of data that contains the timestamp, nonce, previous block hash and the root hash of a data structure called the Merkle tree storing all transactions in the block. |
|  |
| chapter 4: WALLETS, MINING, & MORE |
|  |
| chapter 5: ETHEREUM AND SMART CONTRACTS ENABLING A DECENTRALIZED FUTURE |
| smart contract: code that facilitates, verifies, or enforces the negotiation or execution of a digital contract  Trusted entity must run this code |
| Ethereum is a decentralized platform designed to run smart contracts |
| Ethereum has a native asset called ether |
|  |
| Etherum  ○ Block creation time: ~15 sec vs ~10 min  ○ Proof-of-Work: Ethash (currently ASIC resistant) vs SHA-256  ○ Exchange Rate: $230.56 (2018-09-30) |
|  |
| Ethereum Contracts generally serve four purposes:  ○ Store and maintain data  ● Data represents something useful to users or other contracts  ● Ex: a token currency or organization’s membership  ○ Manage contract or relationship between untrusting users  ● Ex: financial contracts, escrow, insurance  ○ Provide functions to other contracts  ● Serving as a software library  ○ Complex Authentication  ● Ex: M-of-N multisignature access |
| A full-fledged Ethereum miner must:  0. Download the entire Ethereum  blockchain  1. Verify incoming transactions and  Run Smart Contract code  invoked by transactions  2. Create a block  3. Find a valid nonce  4. Broadcast your block  5. Profit! |
| Every Ethereum node runs EVM |
| Every contract requires “gas”, which “fuels” contract execution |
| Every transaction specifies:  ● the startgas , or the maximum quantity of gas it is willing to consume  ● the gasprice , or the fee in ether it is willing to pay per unit gas |
| ● At the start of the transaction  ○ startgas \* gasprice (units = ether) are subtracted from the sender’s account (the one “poking” the contract)  ● If the contract successfully executes …  ○ the remaining gas is refunded to the sender  ● If the contract execution runs out of gas before it finishes …  ○ execution reverts  ○ startgas \* gasprice are not refunded  ● Purchasing gas == purchasing distributed, trustless computational power  ● An attacker looking to launch a DoS attack will need to supply enough ether to  fund the attack |
|  |
|  |
| lec 6: attacks |
|  |
| lec 7: Distributed Systems and Alternative Consensus |
| Safety：This will not happen |
| Liveness: This must happen |
| To ensure correctness, one uses a consensus algorithm achieving the following: Validity, Agreement, and Termination  Validity: any value decided upon must be proposed by one of the processes  Agreement: all non-faulty processes must agree on the same value  Termination: all non-faulty nodes eventually decide |
| Safety Properties: Validity and Agreement: Honest nodes will never decide on trivial, random, or different values  Liveness Properties: Termination: all nodes eventually decide on a value |
| CAP theorem:    Consistency: Every node provides the most recent state  Availability: Every node has consistent read and write access  Partition Tolerance: The system works despite partitions in the network |
| Byzantine Generals’ Problem: generals stationed around a city are trying to coordinate an attack  ● success = they all do the same thing (retreat or attack)  ● send messages to each other  ● some messages may be sent from saboteurs  Byzantine Generals’ Problem:  ● How do we ensure that all honest generals make the same decision?  ○ No solution for ⅓ or greater corruption  ● Solution to all remaining cases:  ○ Practical Byzantine Fault Tolerance |
| Assumptions of fault tolerant systems vs Byzantine fault tolerant systems:  ● Fail-stop fault: Nodes can crash, not return values, crash detectable by other nodes  ● Byzantine fault: Nodes can do all of the above and send incorrect/corrupted values, corruption or manipulation harder to detect |
|  |
| Nakamoto Consensus:  ● Elect leader through some “lottery” (Proof of work, …)  ● Leader creates next block  ● Others vote implicitly by including block in their chain |
| Proof-of-stake  Resource(s) consumed: native currency  ● Nodes must stake tokens to participate in the consensus process  ○ Staked tokens cannot be spent  ○ Voting power in proportion to how much they stake  ○ Slashing: destroying staked tokens to punish bad behavior  ● Main Idea: someone who is invested in the network will behave in its best interest  Cosmos & Tendermint  ● Nodes stake tokens to become validators  ● Proposer (selected validator) proposes a block, likelihood of being selected is proportional to stake  ● Block is committed if ⅔ majority of validators vote on it  ○ Node waits for news from ⅔ before proceeding  ● Validators’ stake is slashed (punishment) for bad behavior  ○ Voting duplicitously: voting on 2 blocks at same height  ○ Going offline  (sacrifice availability, Byzantine fault tolerant) |
| Proof-of-burn  Proof-of-space  Proof-of-elapsed-time (Based on assumptions of randomness and trust in manufacturer (Intel)) |
| Proof-of-stake pros:  ● Environmentally friendly: does not rely on meaningless computation  ● Scalability: possible to reach consensus faster  ● Security: more expensive to reach 51% stake  ● Decentralization: arguably more decentralized (no mining pools)  cons:  “the rich get richer” |
| Paxos: (Not byzantine fault tolerant)  Within the Paxon Parliament...  Proposer: legislator, advocates a citizen's request, moves protocol forward  Acceptor: legislator, voter  Learner: remembers and carries out result for citizen  Quorum:  ● any majority of Acceptors  ● any two Quorums must overlap |
| Raft: (Not byzantine fault tolerant)  A Raft cluster has one and only one elected leader  ● Communicates with client directly  ● Responsible for managing log replication on the other servers of the cluster  ● Leads until it fails or disconnects, in which case a new leader is elected  Leader Election  1. Leader sends “heartbeats” to other nodes saying that it is online  2. If other nodes no longer receive “heartbeat,” they start an election cycle (and internal timer)  3. First candidate to timeout becomes new leader  Log Replication  1. Leader accepts client request  2. Leader ensures all other nodes have followed that request |
| PBFT: (Practical Byzantine Fault Tolerance)  Handle f Byzantine faults in a system with 3f + 1 nodes |
| Federated byzantine agreement: (Ripple, Stellar)  quorum is a set of nodes sufficient to reach agreement  Problem: How do we choose quorums in a decentralized way?  Solution: introduce quorum slices  ● Subset of a quorum that can convince one particular node of agreement  ● Individual nodes decide on other participants they trust for information  Pros:  Decentralized control: no central authority that authorizes consensus  Low latency: consensus achieved in a few seconds  Flexible trust: nodes choose who they trust, don’t have to trust the entire network |
|  |
| lec 8: cryptoeconomics and proof-of-stake |
| Cryptoeconomics: Application of economics to design a distributed system that aligns incentives such that users acting in self-interest benefit the system as well |
| Bitcoin’s economic design is what makes it so special  It has the ability to unite disparate actors,  1. none of whom trust each other,  2. all of whom have an incentive to steal from each other,  3. some of whom hope to destroy the system. |
| Mining rewards incentivize block creation  Transaction fees incentivize inclusion of transactions  Transaction fees also disincentivize spamming the network with microtransactions |
| Transactions must be valid:  ● Proof of ownership (signature)  ● Available funds  ● No other transactions using these funds  Private/Public Key Cryptography enables exclusive possession of funds. |
| Proof-of-work cons:  there’s no defender’s advantage  ● The cost of attack and the cost of defense are 1:1 ratio  ● Constraints are inflexible (i.e. computation, electricity) |
| Proof-of-stake pros:  maintain a defender’s advantage  Punish malicious behavior much more greatly than in PoW  cons:  Rich get Richer  Liquidity problem as funds are locked up  Can rewrite history of blockchain if someone with a huge share of stake sells private keys |
| Attacks on proof-of-stake  In chain-based Proof-of-Stake, validators know that if they create blocks on multiple competing chains they’ll maximize their chances of getting a reward. The blockchain never converges  defense:  Slashing  ● Make a rule that says: You are only allowed to “vote” on one fork  ● Two possibilities:  ○ If you are caught voting on the wrong fork, you get punished  ○ If you are caught voting on multiple forks, you get punished |
| Long range attack:  ● Attacker creates a new branch starting from genesis block  ○ Easy to make new blocks  ● Attempt to take over main chain  Flawed Preconception:  The longest chain is the most trustworthy and therefore the correct one |
| stake grinding attack:  In a Proof-of-Stake ecosystem, need a way to randomly pick the next validator.  Next chosen validator depends on previous block’s signature.  ○ The current validator can produce new signatures to improve his chance of being picked again.  defense:  ● Don’t use any mutable parameters of the previous block as entropy to generate randomness  ● Have all the validators deposit their stake well in advance.  ● Have some sort of secret sharing/threshold signature scheme, through which multiple validators collaboratively generate the random value. |
| [Algorand](https://blockchainatberkeley.blog/the-need-for-an-incentive-scheme-in-algorand-6fe9db45f2a7) |
|  |
| lec 9: enterprise block chain |
| Rise of interest in "private blockchains" or "permissioned ledgers."  ○ Not open  ○ Not trustless  ○ No economic incentives like in Bitcoin  ○ Separate "blockchain" from "Bitcoin"  (只是用花哨的词汇来☞公钥加密) |
| use cases:  ○ Solving coordination failures  ○ Horizontal integration  ○ Creating self-sovereign decentralized networks |
| Blockchains are a fundamentally inefficient data structure  full replication, redundant data & computation |
|  |
| distributed ledgers: A group of nodes that do not fully trust each other |
| Examples:  Hyperledger Fabric, CONSENSYS, R3CEV(Corda, Designed for banks to record, manage, synchronize, support financial transactions and agreements. Hopes to power frictionless transactions between banks, supplant current non-interoperable legacy systems), ENTERPRISE ETHEREUM ALLIANCE, |
| Quorum: open-source enterprise-focused blockchain  ● Soft fork of Ethereum and improves upon privacy, scalability, and network  ● Privacy: Uses advanced cryptography to obscure transaction details  ○ Zero-knowledge proof cryptography from Zcash partnership  ● Consensus: uses Voting-based BFT consensus algorithm (Raft-based)  ● Permissioned Nodes: higher level of trust, more private |
|  |
|  |
| lec 10: scalability |
| more volume of transactions  less block time |
| scalability trilemma  pick 2 of the three: security, decentralization, scalability |
| Techniques:  ● Vertical Scaling - add more RAM/ CPU power to each existing machine  ● Horizontal Scaling - add more machines of the same computational power  ● Diagonal Scaling - add more powerful machines |
| Layer 1 scaling refers to changing the blockchain itself  Layer 2 scaling refers to pushing computation off the blockchain |
| Why not increase the speed of blocks by decreasing difficulty of the PoW problem?   1. Time to broadcast block fixed 2. Size 3. Natural forks   idea: (ghost) Ethereum  ● Increase the speed of blocks by decreasing difficulty of the POW +  weighted POW blockchain (instead of longest)  ● + Decrease Incentive for Pooled Mining! |
| Increase blocksize  cons:   1. Hard fork 2. Size of block chain increases very fast 3. One time linear capacity increase 4. Longer propagation times |
| decrease size of transactions solutions:   1. Segwit (move signature outside blocks), avoid hard fork   merkle tree will include both transactions and witnesses  pros:  fix transaction malleability  soft fork  efficiency gains  smaller size of blockchain  cons:  one time linear capacity increase  very complicated and ugly (400 lines of code)  wallets have to incorporate it   1. Recursive snarks   idea:  Alice generates proof that she can send a valid transaction to Bob.  ● Include this proof and changes to the balance sheet instead of the  transaction itself.  ● Any machine in the network can verify the proof in milliseconds  BLOCK = root hash of the content of the ledger + proofs of valid transactions that changed ledger to current state + proof that previous block’s proof is valid  Drawbacks  ● Proofs are too time consuming to generate, could take hours.  ● Requires Trusted Setup (z-cash) |
| Vertical Scaling off-chain  issue with bitcoin payments:   1. Long delays (1 hour wait) 2. High fees   idea: (private channel)  Use Bitcoin script to create blockchain-enforceable contracts between Alice and Bob so that neither party can cheat the other, while maintaining the private balance sheet functionality (payment channels)  conclusions:  Only two transactions on the blockchain  ○ Supports arbitrary number of local transactions between Alice and Bob.  Cons:   1. Capital lock 2. Only send money between them   Lightning network  pros:  a. don't need to wait for confirmation times  b. transactions as fast as communication delay across network.  c. far fewer (expensive) transactions on the Blockchain  Instead of 3 tps, the Bitcoin network can support 10,000’s+ of tps  d.cheap  Raiden: Ethereum payment channel network |
| Horizontal scaling  sharding:  Sharding is the idea of not requiring every miner to be working on every single block  Node categories:  super-full, top-level, single-shard, light  challenges:   1. cross-shard communication 2. single-shard takeover   sidechain: (rootstock?)  Idea: If you can’t speed up the bitcoin blockchain, why not create multiple blockchains?    Pros:  Less things on bitcoin blockchain, but can still be pegged to it.  Cons:  Loses security as hashing power is spread over multiple chains |
|  |
| Advanced scaling & generalization  Plasma, the Fourth State, (Cosmos-SDK)  ● Application: processing transactions  ● Consensus: agreement on transactions & updates  ● Networking: propagation of transactions  Tendermint provides networking and consensus layers |
|  |
|  |
| lec 11: anonymity, mixing and altcoins |
| not possible in Ethereum (account-based) |
| coinjoin(2011)，coinshuffle(2014)，joinmarket(2015) |
| dash, Monero, z-cash, mimblewimble, |
| 参考笔记\_bitcoin |
|  |
| lec 12: a blockchain powered future |
|  |
|  |
|  |
|  |
|  |
|  |
|  |