Module-3
Smart Contracts
Ethereum Blockchain

- Module-3 Smart contracts and Ethereum
- Smart Contracts and Ethereum 101: Smart Contracts: Definition, Ricardian Contracts. Ethereum 101: Introduction, Ethereum blockchain.
- Further Ethereum: Blocks and Blockchain, The genesis block, block validation mechanisms, Block difficulty, Gas Fee schedule, Forks in blockchain, Nodes and miners, The consensus mechanism, CPU mining, GPU mining, Benchmarking, mining rigs, mining pools

Smart Contracts

- Smart contracts were first theorized by Nick Szabo in the late 1990s.
- "A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs."

Smart Contracts Definition

- Generalized definition of a smart contract: A smart contract is a secure and unstoppable computer program representing an agreement that is automatically executable and enforceable.
- Smart contract is in fact a computer program that is written in a language that a computer or target machine can understand.
- Also, it encompasses agreements between parties in the form of business logic.
- Another key idea is that smart contracts are automatically executed when certain conditions are met.
- They are enforceable, which means that all contractual terms are executed as defined and expected, even in the presence of adversaries.

- They must be designed to be secure, fault-tolerant, and able to maintain their state even in unfavorable external conditions.
- Some suggest they can be "automatable" with human input, but truly smart contracts should be fully automated with the help of Oracles.
- Smart contracts often use a state machine model to manage and update their internal state based on rules.
- Legal concerns remain about whether courts can accept code as contracts, and how disputes, regulation, and compliance can be handled.
- Research is exploring ways to make smart contract code both human- and machine-readable, such as using markup languages like LKIF(Legal Knowledge Interchange Format.).

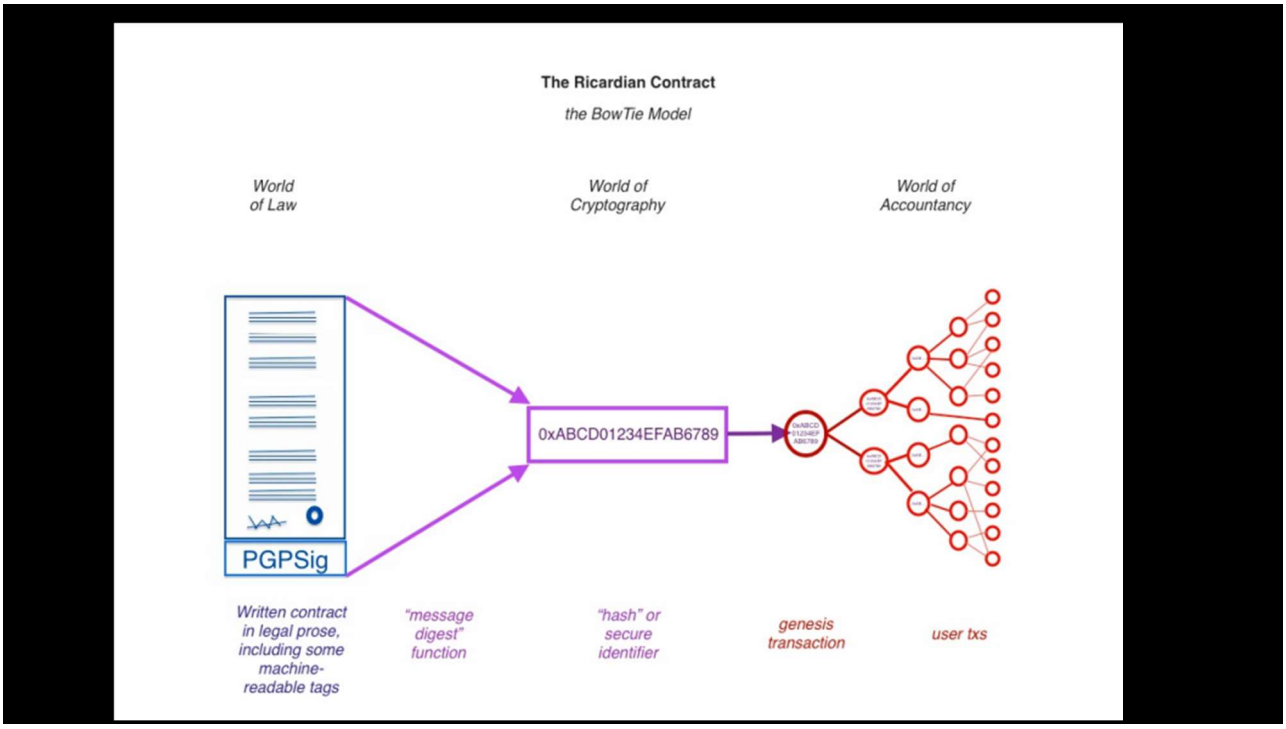
- Smart contracts must be deterministic, meaning they always produce the same result for the same input.If different nodes get different results, consensus fails and the blockchain system breaks down.
- The programming language used should avoid non-deterministic functions to maintain stability and integrity.
- Floating-point operations or some math functions in different environments can give different results, which is unacceptable.
- Determinism ensures all nodes executing the contract get identical results, keeping the network synchronized
- Deterministic contracts always produce accurate and predictable outputs, executing business rules as intended.

- A smart contract has the following four properties:
- Automatically executable
- Enforceable
  - They should work on the principle that code is law (there is no need for an arbitrator or a third party to control or influence the execution of the smart contract)
- Semantically sound
  - A smart contract, the language must be understood by both computers and people(Ricardian contracts)
- Secure and unstoppable.
  - They are required to be designed in such a fashion that they are fault tolerant and executable in reasonable amount of time.
  - These programmes should be able to execute and maintain a healthy internal state, even if external factors are unfavorable.

Ricardian contracts

- Ricardian contracts were originally proposed in the "Financial Cryptography in 7 Layers" paper by Ian Grigg in late 1990s.
- These contracts were used initially in a bond trading and payment system called Ricardo.
- The key idea is to write a document which is understandable and acceptable by both a court of law and computer software.
- Ricardian contracts address the challenge of issuance of value over the Internet.

- Ricardian contract is a document that has several of the following properties:
  - A contract offered by an issuer to holders
  - A valuable right held by holders, and managed by the issuer
  - Easily readable by people (like a contract on paper)
  - Readable by programs (parseable, like a database)
  - Digitally signed
  - Carries the keys and server information
  - Allied with a unique and secure identifier



- The diagram shows number of elements:
  - The World of Law on the left-hand side from where the document originates. This document is a written contract in legal prose with some machine-readable tags.
  - This document is then hashed.
  - The resultant message digest is used as an identifier throughout the World of Accountancy, shown on the right-hand side of the diagram.
- The World of Accountancy element represents any accounting, trading, and information systems that are being used in the business to perform various business operations.
- The idea behind this flow is that the message digest generated by hashing the document is first used in a so-called genesis transaction, or first transaction, and then used in every transaction as an identifier throughout the operational execution of the contract. This way, a secure link is created between the original written contract and every transaction in the World of Accounting

Smart Contracts Vs. Ricardian Contracts

Characteristics	Smart Contract	Ricardian Contracts
Purpose	Execute the terms of an agreement	Record the terms of an agreement as a legal document
Flow	Automate actions on the blockchain-based applications	It can also automate operations on the blockchain-based applications
Validity	It is not a legally binding document	It is a legally binding document or agreement
Versatility	They can't be Ricardian Contracts	Any Ricardian Contract can be a Smart Contract as well
Readability	Smart contracts are machine-readable but not necessarily human-readable	Ricardian Contracts are both machine-readable as well as human-readable

Oracles

- Oracles are an important component of the smart contract ecosystem.
- The limitation with smart contracts is that they cannot access external data which might be required to control the execution of the business logic
- Ex: Imagine a smart contract that pays dividends based on the stock price of a company. The contract has the rule: "If the stock price goes above $100, release dividend payments", But smart contracts cannot directly fetch the stock price from outside sources like a financial news website or stock exchange.
- Oracles can be used to provide external data to smart contracts.
- An Oracle is an interface that delivers data from an external source to smart contracts.

- Depending on the industry and requirements, Oracles can deliver different types of data ranging from weather reports, real world news, and corporate actions to data coming from Internet of Things (IoT) devices.
- Oracles are trusted entities that use a secure channel to transfer data to a smart contract.

- Oracles are also capable of digitally signing the data proving that the source of the data is authentic.
- Smart contracts can then subscribe to the Oracles, and the smart contracts can either pull the data, or Oracles can push the data to the smart contracts.
- It is also necessary that Oracles should not be able to manipulate the data they provide and must be able to provide authentic data.
- Even though Oracles are trusted, it may still be possible in some cases that the data is incorrect due to manipulation.
- Therefore, it is necessary that Oracles are unable to change the data. This validation can be provided by using various notary schemes.

Decentralized Oracles

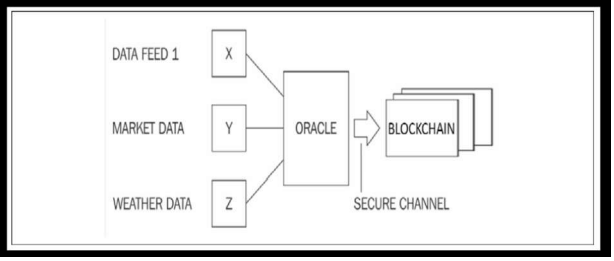
- Another type of Oracle, which essentially emerged due to the decentralization requirements, can be called decentralized Oracles.
- These types of Oracles can be built based on some distributed mechanism.
- Oracles can get data from another blockchain that is secured by many participants, which helps make sure the data is accurate and trustworthy.

Hardware Oracles

- Used when real-world data from physical devices is required.
- For example, this can be used in telemetry and IoT.
- However, this approach however requires a mechanism in which hardware devices cannot be tampered with.
- This can be achieved by using tamper-proof devices.

Generic model of an oracle and smart contract ecosystem

- Multiple external data feeds (like X = Data Feed 1, Y = Market Data, Z = Weather Data) provide information from the outside world.
- These feeds are sent into an Oracle, which acts as a trusted middleware.
- The Oracle validates and formats this external data.
- Through a secure channel, the Oracle sends verified data to the Blockchain.
- Once on the blockchain, the data can be used for smart contracts, decentralized applications, or automated execution of agreements.



Ethereum Blockchain

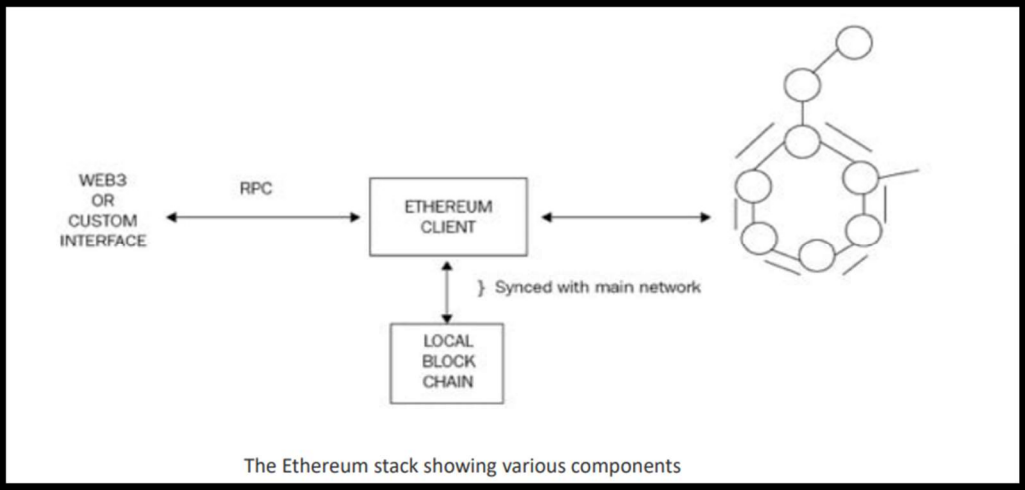
- Ethereum was conceptualized by Vitalik Buterin in November 2013.
- The key idea proposed was the development of a Turing-complete language that allows the development of arbitrary programs (smart contracts) for blockchain and decentralized applications.

Ethereum clients and releases

- An Ethereum client is a software application that implements the Ethereum specification and communicates over the peer-to-peer network with other Ethereum clients.
- Pantheon — Java client by PegaSys.
- Geth — Go client.
- Parity — Rust client.
- Aleth — C++ client.
- Pyethapp — Python client using pyethereum.
- Trinity — Python client using py-evm.
- Ethereumjs — JS client using ethereumjs-vm.
- Ethereumj — Java client by the Ethereum Foundation.

- The first release of Ethereum was known as Frontier, and the next release of Ethereum is called homestead release.
- The next version is named metropolis and it focuses on protocol simplification and performance improvement.
- The final release is named serenity, which is envisaged to have a Proof of Stake algorithm (Casper) implemented with it.
- Other areas of research targeted with serenity include scalability, privacy, and Ethereum Virtual Machine (EVM) upgrade.

The Ethereum stack

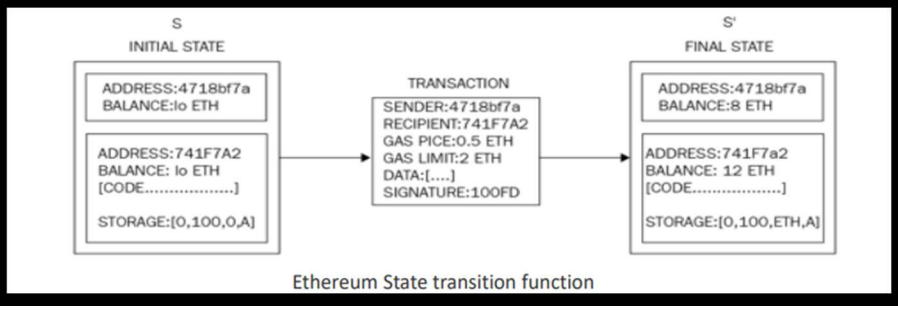


The Ethereum stack showing various components

- At the core, there is the Ethereum blockchain running on the P2P(peer to peer) Ethereum network.
- Secondly, there's an Ethereum client (usually geth) that runs on the nodes and connects to the peer-to-peer Ethereum network from where blockchain is downloaded and stored locally.
- Ethereum client provides various functions, such as mining and account management.
- The local copy of the blockchain is synchronized regularly with the network.
- Another component is the web3.js library that allows interaction with geth via the Remote Procedure Call (RPC) interface. Enables developer's to Send transactions, Read blockchain data, Interact with smart contracts

Ethereum blockchain

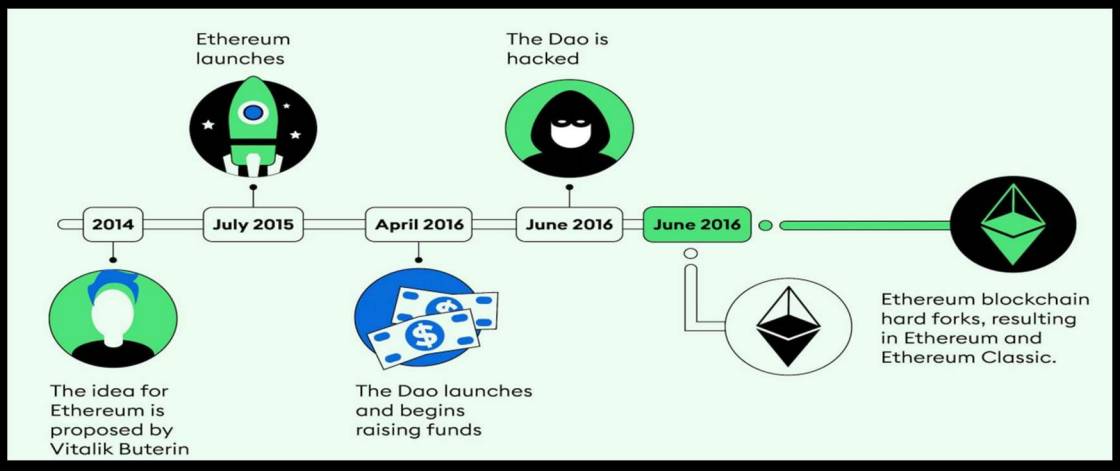
- Ethereum blockchain can be visualized as a transaction-based state machine.
- The idea is that a genesis state is transformed into a final state by executing transactions incrementally.
- The final transformation is then accepted as the absolute undisputed version of the state.



Currency (ETH and ETC)

- As an incentive to the miners, Ethereum also rewards its native currency called Ether, abbreviated as ETH.
- After the DAO hack, a hard fork was proposed in order to mitigate the issue;
- Therefore, there are now two Ethereum blockchains:
  - ETC- Ethereum Classic – Original blockchain
  - ETH- Hard-forked version
- ETH continues to grow and on which active development is being carried out.

The DAO Hack



Forks

- Blockchain forks are essentially a split in the blockchain network.
- Two types of forks:
- Hard forks
  - Involve a significant change to the rules of a blockchain.
  - It is a change to the blockchain protocol that is not backward-compatible and requires all users to upgrade their software in order to continue participating in the network.
- Soft forks
  - They are backward compatible
  - Nodes that do not upgrade will still see the chain as valid.
  - Soft forks are often implemented to add new features without changing the rules of the blockchain.

Ethereum Forks

- Initial fork due to major protocol upgrades, it resulted in a hard fork.
- The protocol was upgraded at block number 1,150,000.
- This was the migration from the first version of Ethereum known as Frontier to the second version of Ethereum called homestead.
- There was unintentional fork that occurred on November 24, 2016, at 14:12:07 UTC was due to a bug in the geth client's journaling mechanism.
- Network fork occurred at block number 2,686,351.
- This bug resulted in geth failing to revert empty account deletions in the case of the empty out-of-gas exception.
- gas as execution fee is paid upfront by the transaction originators

Gas

- Ethereum blockchain are required to cover the cost of computation they are performing.
- The cost is covered by something called gas or crypto fuel, which is a new concept introduced by Ethereum.
- This gas as execution fee is paid upfront by the transaction originators.
- The fuel is consumed with each operation.
- Each operation has a predefined amount of gas associated with it.
- Each transaction specifies the amount of gas it is willing to consume for its execution.
- If it runs out of gas before the execution is completed, any operation performed by the transaction up to that point is rolled back.
- If the transaction is successfully executed, then any remaining gas is refunded to the transaction originator.

Transactions

- A transaction in Ethereum is a digitally signed data packet using a private key.
- It contains the instructions that, when completed, either result in a message call or contract creation.
- Transactions can be divided into two types based on the output they produce:
  - Message call transactions: This transaction simply produces a message call that is used to pass messages from one account to another.
  - Contract creation transactions: As the name suggests, these transactions result in the creation of a new contract. This means that when this transaction is executed successfully, it creates an account with the associated code.

Transactions are composed of a number of common fields

- NONCE
  - Nonce is a number that is incremented by one every time a transaction is sent by the sender.
  - It must be equal to the number of transactions sent and is used as a unique identifier for the transaction.
- GASPRICE
  - The gasPrice field represents the amount of Wei(smallest denomination of ether) required in order to execute the transaction.
  - 1 Ether (ETH) = 1,000,000,000,000,000,000 Wei = 10^{18} Wei
- GASLIMIT
  - The gasLimit field contains the value that represents the maximum amount of gas that can be consumed in order to execute the transaction.
  - This is the amount of fee in Ether that a user (for example, the sender of the transaction) is willing to pay for computation.

- TO
  - As the name suggests, the to field is a value that represents the address of the recipient of the transaction.
- VALUE
  - Value represents the total number of Wei to be transferred to the recipient; in the case of a contract account, this represents the balance that the contract will hold
- SIGNATURE
  - Signature is composed of three fields, namely v, r, and s.
  - These values represent the digital signature (R, S) and some information that can be used to recover the public key (V).
  - The signature is based on ECDSA scheme and makes use of the SECP256k1 curve

Field	Type	Purpose
v	1 byte	Used to recover the public key from the signature (also called the *recovery ID*).
r	32 bytes	One part of the elliptic curve point derived during signing.
s	32 bytes	The second part of the signature, computed mathematically using the message hash, `r`, the private key, and a random number.

- S is calculated by multiplying R with the private key and adding it into the hash of the message to be signed and by finally dividing it with the random number chosen to calculate R.
- S=(hash+R×privateKey)/k
- S is also a 32 byte sequence.
- R and S together represent the signature.
- In order to sign a transaction, the ECDSASIGN function is used, which takes the message to be signed and the private key as an input and produces V, a single byte value; R, a 32 byte value, and S, another 32 byte value.
- ECDSASIGN (Message, Private Key) = (V, R, S)

•Message Hashing:
The transaction or message is first converted into a hash (a fixed-length unique value).
•Random Number (k):
A random number k is chosen — this ensures every signature is unique even for the same message.
•R Calculation:
Using elliptic curve math, a point is calculated from k.
The x-coordinate of this point becomes R, which is then encoded as a 32-byte sequence.
•S Calculation:
The value S is computed using:
S=(hash+R×privateKey)/k
S is also 32 bytes.
•V Calculation:
The single-byte V value tells which of the possible public keys (there are two mathematical possibilities) corresponds to this signature.
So, V helps "recover" the correct public key from the signature later.
•ECDSASIGN (Message, Private Key) = (V, R, S)

Blocks and blockchain

- Ethereum blocks consist of various elements, which are described as follows
- The block header
- The transactions list
- The list of headers of Ommers or uncles

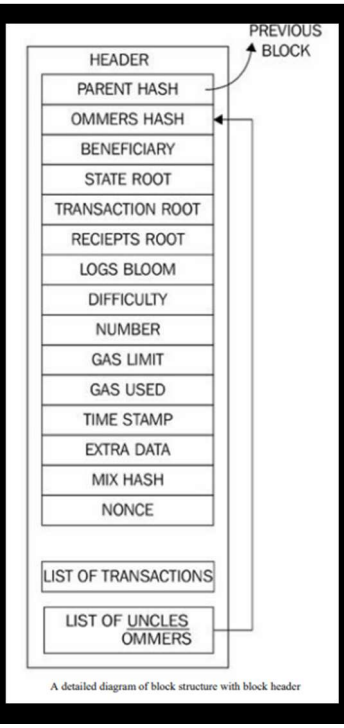
The block header

- The header contains valuable information
- Parent hash:
  - This is the Keccak 256-bit hash of the parent (previous) block's header. It ensures **blockchain continuity and linkage**.
- Ommers hash:
  - This is the Keccak 256-bit hash of the list of ommers (uncles) blocks included in the block. It helps to reward miners and improve security.
- The beneficiary:
  - Beneficiary field contains the 160-bit address of the recipient that will receive the mining reward once the block is successfully mined
- State root:
  - The state root field contains the Keccak 256-bit hash of the root node of the state trie. It is calculated after all transactions have been processed and finalized.
- Transactions root:
  - The transaction root is the Keccak 256-bit hash of the root node of the transaction trie. Transaction trie represents the list of transactions included in the block.

- Receipts root:
  - The receipts root is the Keccak 256-bit hash of the root node of the transaction receipt trie. This trie is composed of receipts of all transactions included in the block. Each receipt contains **post-transaction information** for all transactions included in the block.
- Logs bloom:
  - The logs bloom is a bloom filter that is composed of the logger address and log topics from the log entry of each transaction receipt of the included transaction list in the block. Enables **quick search of logs** without scanning every receipt.
- Difficulty:
  - The difficulty level of the current block. Determines how hard it is to mine a block.
- Number:
  - The total number of all previous blocks; the genesis block is block-0.
- Gas limit:
  - The field contains the value that represents the limit set on the gas consumption per block. Protects the network from overly large blocks.

- Gas used:
  - The field contains the total gas consumed by the transactions included in the block
- Timestamp:
  - Timestamp is the epoch Unix time of the time of block initialization/mined.
- Extra data:
  - Extra data field can be used to store arbitrary data related to the block.
- Mixhash:
  - Mixhash field contains a 256-bit hash that once combined with the nonce is used to prove that adequate computational effort (PoW) has been spent in order to create this block.
- Nonce:
  - Nonce is a 64-bit hash (a number) that is used to prove, in combination with the mixhash field, that adequate computational effort (PoW) has been spent in order to create this block.

The block header



A detailed diagram of block structure with block header

The genesis block

- The genesis block varies slightly from normal blocks due to the data it contains and the way it has been created.

Element	Description
Time stamp	Timestamp (Jul-30-2015 03:26:13 PM +UTC)
Transactions	8893 transactions and 0 contract internal transactions in this block
Hash	0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3
Parent hash	0x0000000000000000000000000000000000000000000000000000000000000000
SHA3 uncles	0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347

Element	Description
Mined by	0x0000000000000000000000000000000000000000 IN 15 secs
Difficulty	17,179,869,184
Total difficulty	17,179,869,184
Size	540 bytes
Gas used	0
Nonce	0x0000000000000042
Block reward	5 Ether
Uncles reward	0
Extra data	
Gas limit	5,000

The block validation mechanism

- An Ethereum block is considered valid if it passes the following checks:
- Consistent with uncles and transactions, this means that all ommers (uncles) satisfy the property that they are indeed uncles and also if the PoW for uncles is valid.
- If the previous block (parent) exists and is valid.
- If the timestamp of the block is valid. This means that the current block's timestamp must be higher than the parent block's timestamp.
- If any of these checks fails, the block will be rejected.

Block finalization

- Block finalization is a process that is run by miners to validate the contents of the block and apply rewards. It results in four steps being executed.
- Ommers validation:
  - Validate ommers (stale blocks also called uncles).
- Transaction validation:
  - Validate transactions. In the case of mining, determine transactions. The process involves checking whether the total gas used in the block is equal to the final gas consumption after the final transaction
- Reward application:
  - Apply rewards, which means updating the beneficiary's account with a reward balance. In Ethereum, a reward is also given to miners for stale blocks, which is 1/32 of the block reward.
- State and nonce validation:
  - Verify the state and block nonce. In the case of mining, compute a valid state and block nonce.

Block difficulty

- Block difficulty is increased if the time between two blocks decreases, whereas it increases if the block time between two blocks decreases. This is required to maintain a roughly consistent block generation time.
- The difficulty adjustment algorithm in Ethereum's Homestead release is shown as follows:

```
block_diff = parent_diff + parent_diff // 2048 *
max(1 - (block_timestamp - parent_timestamp) // 10, -99) +
int(2**((block.number // 100000) - 2))
```

- The algorithm means that, if the time difference between the generation of the parent block and the current block is less than 10 seconds, the difficulty goes up.
- If the time difference is between 10 to 19 seconds, the difficulty level remains the same.
- Finally, if the time difference is 20 seconds or more, the difficulty level decreases. This decrease is proportional to the time difference.

- The algorithm also includes another part that increases the difficulty exponentially after every 100,000 blocks. This is called Difficulty Time Bomb or Ice Age, introduced in the Ethereum network, which will make it very hard to mine on the Ethereum blockchain at some point in the future
- This will encourage users to switch to Proof of Stake (PoS) scheme proposed by Ethereum called Casper as mining on the POW chain will eventually become prohibitively difficult.
- In the Byzantium release, Ethereum changed the difficulty formula to include the number of uncle blocks in the calculation.

Gas fee schedule

- Gas is charged in three scenarios as a prerequisite to the execution of an operation:
- The computation of an operation
- For contract creation or message call
- Increase in the usage of memory

Forks in the blockchain

- Forks are the splitting of the blockchain into two. This can be intentional or non-intentional. Usually, as a result of major protocol upgrade, a hard fork is created, and unintentional fork can be created due to bugs in the software.
- With the release of Homestead, due to major protocol upgrades, it resulted in a hard fork. The protocol was upgraded at block number 1,150,000, resulting in the migration from the first version of Ethereum known as Frontier to the second version of Ethereum called Homestead.
- An unintentional fork, which occurred on November 24, 2016, at 14:12:07 UTC was due to a bug in the Geth client's journaling mechanism. As a result, a network fork occurred at block number 2,686,351. This bug resulted in Geth failing to prevent empty account deletions in the case of the empty out-of-gas exception. This was not an issue in Parity (another popular Ethereum client). This means that from block number 2,686,351, the Ethereum blockchain is split into two, one running with the Parity clients and the other with Geth. This issue was resolved with the release of Geth version 1.5.3

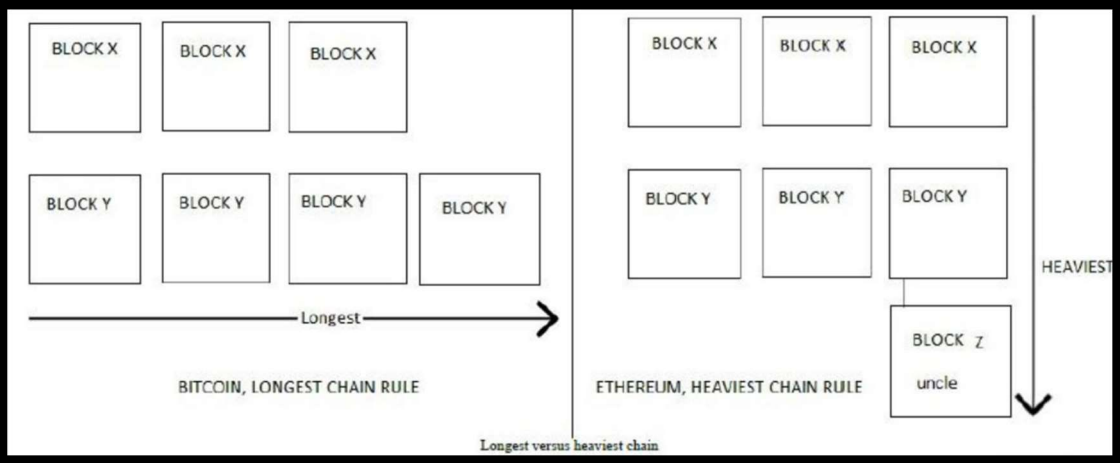
Nodes and miners

- Ethereum network contains different nodes. Some nodes act only as wallets, some are light clients, and few are full clients running the full blockchain. One of the most important type of nodes are mining nodes.
- Mining is the process by which new blocks are elected via a consensus mechanism and added to the blockchain. As a result, currency (Ether) is rewarded to the nodes which perform mining operations. These mining nodes are known as miners.
- Miners are paid in Ether as an incentive for them to validate and verify blocks made up of transactions. The mining process helps secure the network by verifying computations.
- At a theoretical level, a miner node performs the following functions:
  - Listens for the transactions broadcasted on the Ethereum network and determines which transactions to be processed.
  - Determines stale blocks called uncles or ommers and includes them in the block
  - Updates the account balance with the reward earned from successfully mining the block
  - A valid state is computed, and the block is finalized, which defines the result of all state transitions

- The current method of mining is based on PoW, which is similar to that of bitcoin. When a block is deemed valid, it has to satisfy not only the general consistency requirements, but it must also contain the PoW for a given difficulty.
- The PoW algorithm is replaced with the PoS algorithm with the release of Serenity in Ethereum.
- An algorithm named Casper has been developed, which replaced the existing PoW algorithm in Ethereum.
- This is a security deposit based on the economic protocol where nodes are required to place a security deposit before they can produce blocks. Nodes have been named bonded validators in Casper, whereas the act of placing the security deposit is named bonding.

The consensus mechanism

- The consensus mechanism in Ethereum is based on the Greedy Heaviest Observed Subtree (GHOST) protocol proposed initially by Zohar and Sompolinsky in December 2013.
- Ethereum uses a simpler version of this protocol, where the chain that has most computational effort spent on it to build it is identified as the definite version. Another way of looking at it is to find the longest chain, as the longest chain must have been built by consuming adequate mining effort.
- In GHOST, stale blocks are added in calculations to figure out the longest and heaviest chain of blocks. Stale blocks are called uncles or ommers in Ethereum.
- The following diagram shows two rules of figuring out which blockchain is the canonical version of truth. In case of Bitcoin, shown on the left-hand side in the diagram, the longest chain rule is applied which means that the active chain (true chain) is the one that has the most amount of PoW done.
- In case of Ethereum, the concept is similar from the longest chain point of view but it also includes ommers (also called uncles), the orphan blocks which means that it rewards those blocks too that were competing with other blocks during mining to be selected and performed significant PoW or were mined exactly at the same time as others but did not make it to the main chain. This makes the chain heaviest instead of longest because it also contains the orphaned blocks. This is shown on the right-hand side of the diagram.



CPU MINING and GPU mining

- Even though not profitable on the mainnet, CPU mining is still valuable on the test network or even a private network to experiment with mining and contract deployment.
- CPU mining may be done using Geth client, web3 Geth console.
- GPU mining may be performed through Ethminer can be run in order to start mining. Ethminer is a standalone miner that can also be used in the farm mode to contribute to mining pools.
- GPU mining requires an AMD or NVIDIA graphics card and an applicable OpenCL SDK.
- Blockchain benchmarking is the process of measuring and evaluating the performance of a blockchain platform under different scenarios and workloads.

Mining rigs AND MINING POOLS

- As difficulty increased over time for mining Ether, mining rigs with multiple GPUs were starting to be built by the miners.
- A mining rig usually contains around five GPU cards with all of them working in parallel for mining, thus improving the chances of finding valid nonces for mining.
- Typical mining rig configuration includes:
  - Motherboard: A specialized motherboard with multiple PCI-E x1 or x16 slots, for example, BIOSTAR Hi-Fi or ASRock H81, is required.
  - SSD hard drive: An SSD hard drive is required. The SSD drive is recommended because of its much faster performance over the analog equivalent.
  - GPU: The GPU is the most critical component of the rig because it does most of the work for mining.
  - Operating system: Linux Ubuntu's latest version is usually chosen as the operating system for the rig because it is more reliable and gives better performance as compared to Windows. EthOS is specially built for Ethereum mining.
  - Power Supply Units (PSUs): In a mining rig there are multiple GPUs running in parallel, therefore there is a need for a constant powerful supply of electricity.
  - Many online mining pools offer Ethereum mining. Ethminer can be used to connect to a mining pool using the special command. Each pool publishes its instructions, but generally, the process of connecting to a pool is similar.