

Blockchain and Distributed Ledger Technologies' Notes

Daniele Bertagnoli

2023/2024

Contents

1	Introduction	4
1.1	What is a Blockchain	4
1.2	Double Spending Problem	4
1.3	Decentralization	4
1.3.1	Proof of Work	4
1.4	Smart Contracts	4
1.5	Private Public and Permissioned Permissionless Transactions	5
2	Main Concepts	6
2.1	Blockchain as protocol	6
2.2	Transactions and Assets	6
2.3	Ledger and Blocks	6
2.3.1	Conflicts	7
2.3.2	Reaching Consensus	8
2.4	Hashing	8
2.5	Pseudonymity	9
3	Bitcoin	9
3.1	Proof of Work (PoW)	10
3.1.1	Empty Blocks	10
3.1.2	Block Hashing	11
3.1.3	Block Time	11
3.1.4	Considerations on PoW	11
3.2	Consensus	12
3.3	Signin	12

3.4	Privacy	13
3.5	Full Nodes and Light Nodes	13
3.6	Script	14
3.6.1	Script Limitations	16
4	Ethereum	16
4.1	Accounts	16
4.1.1	Addresses	16
4.1.2	Nonce	17
4.2	Contracts	17
4.3	Transactions	18
4.3.1	Gas	19
4.3.2	Transaction Execution	19
4.3.3	Transaction and States	20
4.4	Consensus Protocol in Ethereum	20
4.4.1	Old Ethereum Version	20
4.4.2	Proof of Stake	20
4.4.3	Casper the Friendly Finality Gadget (Casper FFG)	21
4.4.4	Eth2	21
4.4.5	Casper FFG + LMD-GHOST (Consensus)	21
4.4.6	Rewards	22
4.4.7	Slashing	23
4.4.8	Effective Balance	23
5	Hashing	24
5.1	Merkle Trees	24
5.2	Patricia Radix Trees	26
6	Solidity	26
6.1	Tokens	26
6.2	Solidity Types	27
6.3	Solidity Functions	28
6.3.1	Function Visibility	28
6.3.2	Data Locations	28
6.3.3	View and Pure	28
6.3.4	Exceptions	28
6.4	Decentralized Applications (DApps)	29
6.5	Truffle and Ganache	29
6.6	MetaMask	29

7	Seminars	29
7.1	Blockchain based Resource Governance for Decentralized Web Environments	29
7.2	Decentralized Oracles	30
7.3	Algorand	31
7.4	Frauds on the Blockchain	34
	7.4.1 Pump and Dump	34
	7.4.2 Rug Pull	35
7.5	Enabling Data Confidentiality with Public Blockchains	35
	7.5.1 IPFS	35
	7.5.2 CP-ABE	35
	7.5.3 CAKE (Control Access Key Encryption)	36
	7.5.4 MARTSIA	36

1 Introduction

1.1 What is a Blockchain

The blockchain is an **open and distributed ledger that can record transaction between two parties efficiently, in a verifiable and permanent way**. Transactions are immutable and every node can access to a copy of the blockchain. The consensus is achieved using several and different algorithms. The blockchain is based on **ordered (order matters!) transactions, transfer of cryptoassets from an user to another**. A block is set of ordered transactions, each of those have the reference to its predecessor. From a graph point-of-view, the **blockchain is a tree with only one node per layer** and with only one leaf. Moreover, this type of graph is said as **weakly connected graph** (there exist always a way to reach a node starting from a node).

1.2 Double Spending Problem

Using the blockchain we can solve the double spending problem. A user cannot send the same cryptoasset to two different users simultaneously, otherwise this would mean that he has duplicated that assets and we cannot determine who is the owner of the transferred asset. This is ensured by inferring the order to every transaction that is stored in the blockchain.

1.3 Decentralization

The **blockchain is a decentralized system, this is fundamental since it ensures that the informations are not stored in a single point of failure**. However, this opens to several problems like inconsistencies between data. To solve those problems, as said before, the **nodes must reach a consensus on each transaction to guarantee consistency**.

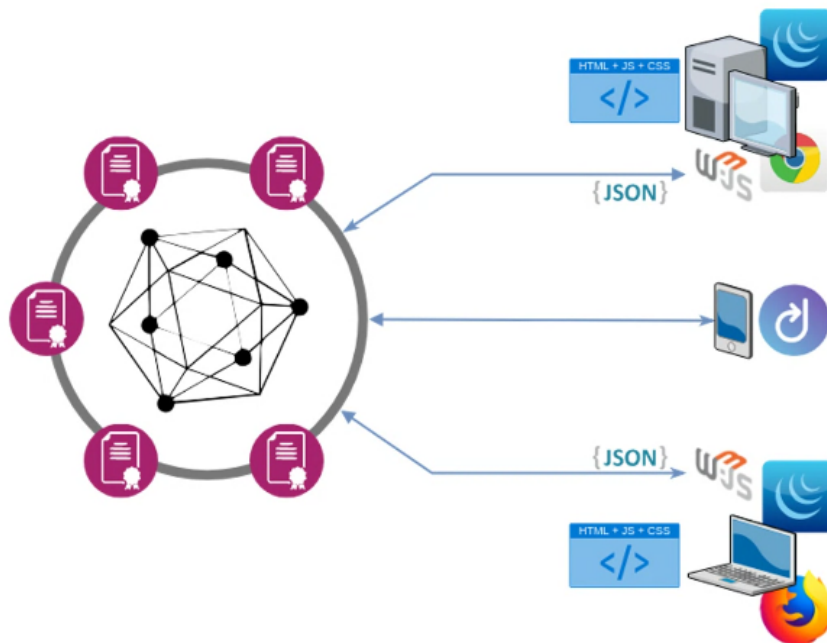
1.3.1 Proof of Work

Bitcoin, for instance, uses the **Proof of Work** to reach the consensus. The idea behind PoW, is to **solve a puzzle to publish the block on the blockchain**. Checking that the puzzle is correct is very fast, instead solving it is much more expensive! To encourage nodes to solve the puzzles, they achieve a fee every time that a block is solved by them.

1.4 Smart Contracts

Smart contracts are pieces of code that offer services. These services are functions that can be called by everyone. They live in the environment where they are deployed, so in a

fully-distributed system. Smart contracts have full control over their own balance and key/value storage. In Ethereum they are run on the Ethereum Virtual Machine (EVM), that is globally accessible VM. Each operation has a cost called **gas price**, so typically, who calls the function has to pay the gas price and additional fees for the service. **Each code invocation is a transaction, so is registered on the blockchain.** Even the deployment of a smart contract is a transaction. They are fundamental for the Web 3.0 concept, where the computation runs on the backend rather than the frontend.




1.5 Private|Public and Permissioned|Permissionless Transactions

Transactions can be either:

- **Private or Public.**
- **Permissioned or Permissionless.**

Based on these properties, the transaction's visibility pool and consensus pool can be reduced or enlarged:



		Transactability / visibility	
		Private	Public
Consensus	Permissionless	Selected nodes can transact and view, all nodes can participate in consensus	Every node can transact and view, participate in consensus
	Permissioned	Selected nodes can transact and view, or participate in consensus	Every node can transact and view, selected nodes participate in consensus

Bitcoin, Ethereum and Algorand (and so on...) are all public permissionless.

2 Main Concepts

2.1 Blockchain as protocol

In a telecommunications context, a protocol is a system of rules that describes how a computer can connect, participate and transmit in the network. All the **computing systems that participate to the network are called nodes**. The blockchain is not a software, computer or software system and this is why different blockchains exist.

It is **architecturally decentralized** (no center authorities), **logically centralized** (every node behaves like he has the main blockchain) and the **information is fully distributed**.

2.2 Transactions and Assets

A **transaction is a recording of a transfer of assets among parties**. An **asset can be either digital currencies, units of inventory, etc...** In the blockchain, transactions are identified using unique IDs called **transaction ID**. The assets are transferred from one (or more) input account to one (or more) output account.

2.3 Ledger and Blocks

A **ledger is a collation (means ordered) of transactions**. In this case the ledger that we are considering is a distributed ledger, so every node stores its own copy (on which the consensus

must be reached). The main advantages of a distributed ledger are:

- Cannot be lost or destroyed.
- Transactions cannot be altered.
- Transactions are always valid.
- Transaction list is always complete.

Users submit a **candidate transaction** by sending it to some nodes in the blockchain. Those nodes will propagate that transaction in the network, this mean that now the transaction becomes part of the **transaction pool** until it be added to some block by a mining node. **Blocks collates transactions**, so they are stored and sorted inside blocks.

Mining nodes (or publishers) are the subset of **nodes that maintain the blockchain by publishing new blocks**. Each **transaction is added to the blockchain when is published in a block** by the miners.

The **validity of a block is ensured by checking**:

1. The providers of funds (input account) have **signed the transaction and have enough funds** on their account.
2. The **block timestamp is greater then the median timestamp of previous 11 blocks and less than network-adjusted time** (median of the timestamps of adjacent nodes) + **2 hours**.

The **blocks that contain invalid transactions are refused by the other nodes**.

2.3.1 Conflicts

Each **node has its own local clock time**, the events (**transactions**) are stored according to that clock time that is not global. **Conflicts** are not due to errors, but are **matter of propagation of transactions in the network**. A transaction could be received and stored in a block by some nodes in a different order with respect to other nodes in the network. It is **not mandatory that every node receives a transaction**, the more are the nodes that hold that transaction, the faster will be its insert into a block. **Also blocks do not have the need of being received by every node, when the node will receive the next block it will request the missing blocks** (it knows that some blocks are missing due to the fact that every block has the pointer to the header of the previous block).

The **CAP Theorem** states that at most only two of the following three properties can be guaranteed:

- **Consistency:** every read receives the most recent write.
- **Availability:** every request receives response (not-necessarily the most recent write).
- **Partition tolerance:** The system continues to operate despite an arbitrary number of messages being dropped by the network between nodes.

The **blockchain only takes account of Availability and Partition tolerance**. The blockchain to consider as valid a block only after that at least some blocks are published after that block. In Bitcoin we need 6 blocks, instead in Ethereum we need 12 blocks.

2.3.2 Reaching Consensus

Due to the block propagation, **it could happen that a partition (set of nodes) receives a new block computed by another partition**. At this point, since this block holds the reference to its ancestor, if this ancestor is not held by the partition 1, the partition 2 (that has computed that block) sends also the block pointed by the new received block. **This step is repeated until they reach a common known block. When this happens, the common block will be pointed by two different blocks, so the two partition must reach a consensus to determine which of the two chains must be kept.** Typically, the longest valid chain is adopted, so miners **tend to accept new received blocks** to enlarge their chain and **increase the possibility of having the longest chain**.

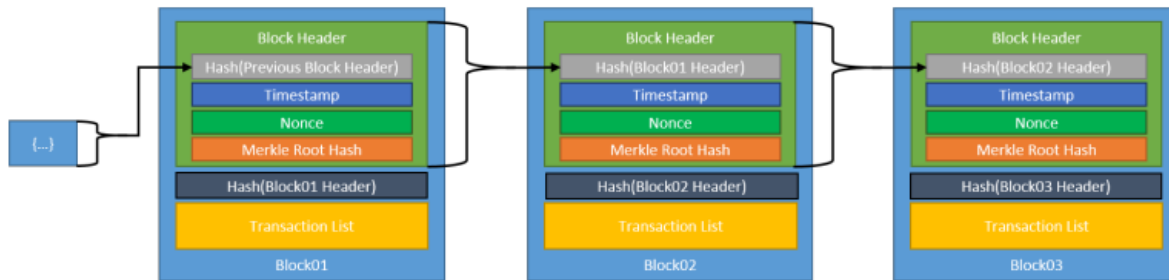
The **consensus algorithm depends on the blockchain**, Bitcoin uses its own that is different from the Ethereum's algorithm and from others.

Consensus .

2.4 Hashing

The hashing function used by the blockchain is a **deterministic algorithm**, so given the same input it will return always the same output. **All the operations are public and do not depend on private keys**. The **input** is a **string of any length** and the **output** it's always a **fixed-length hash value called "digest"**. Even small changes in the input will produce a totally different digest.

Each block computes its own hash that will be pointed by the next block. Since the **header identifies univocally the block**, it is also the part of the block that **is hashed**.



2.5 Pseudonymity

Users of the blockchain are **pseudonymous**, this means that their **real-world identity is secret but their account are not**. This promotes complete transparency in the blockchain. Using **private/public keys (a.k.a asymmetric key cryptography)** we can make other users able to verify our identity (and we can also sign the transactions claim the ownership on them).

Full anonymity is not appropriated for the blockchain due to many reasons, one of those is the fact we cannot identify who is involved in the transactions.

Note that **users can have how as many accounts as they want**, each of those is a separated wallet.

3 Bitcoin

Bitcoin is pure **peer-to-peer electronic cash system**. It is **fully decentralized**, hence payments do not transact through financial institutions. The hash is used for timestamping and the consensus protocol is the Proof of Work (PoW).

The **Bitcoin can be also seen as a State Transistion System**, in which every state is the collection of unspent coins. Each state is composed by **Unspent Transaction Unit Output (UTXO)**, so the unspent transactions owned by each account in the blockchain. The transition from a state to another occurs when the UTXO changes, so when one (or more) sender create a transaction to move the UTXO to a reciever (or more).

In every transaction, that are public (!), we must specify which Unspent Transaction Output (UTXO) we intend to spend (one or more) and also the receiver(s). If is the case, **indicate the amount of coins we wish to retain from these UTXOs if the sum of the coins received with them exceeds our spending needs**. If we specify a retention amount lower than the remaining coins, **the excess coins will be allocated as a fee for the miner**.

The amounts in the transactions are expressed in Satoshi (sat), whit $1 \text{ sat} = 10^{-8} \text{ BTC}$. We use it in order to avoid floating point values with transfers due to approximation errors.

3.1 Proof of Work (PoW)

The **PoW is NOT a consensus protocol**, it is a way to build the consensus but isn't the consensus protocol itself. In the Proof of Work (PoW) model, the righth to publish the next block is **granted by solving a computationally intensive puzzle**. Since the puzzle solving costs in term of time and electricity, **miners are rewarded with coins once that the block becomes a valid block on the chain**. The validity check can be performed very easily and quickly by every node in the network. In particular, when a new node submits a new transaction, it pays the fee destined to the miner. **Miners could show preference for transactions that offer higher fees**. Moreover, the **highest part of the reward is given by the coins achieved for mining a whole block, those coins are generated "from the air"**, so are effectively created (this is why is called mining). The miners sign the blocks through the **coinbase transaction**, in this way they can be rewarded by the blockchain with the newly created coins. **The amount of reward achieved by mining a blockchain is righthshifted (integer division by 2) every 210.000 blocks.**

The transaction fees are not mandatory, however since for miners including a transaction requires time to validate the transaction itself, the higher is the fee the more is the probability of being included into a block. Moreover, this is not only parameter that miners take account of, to validate a whole transaction requires time depending on its size. Typically, big transactions requires bigger fees to be included into a block.

A transaction is considered as invalid if we are trying to send more coins that we have or we are trying to spend UTXO that are not existant/have been sent to us or even if in the transaction we specify an amount of coin to keep that is higher than the remaining. **The invalid transactions will be refused by the miners.**

In Bitcoin we also have a limit to the number of coins that can be mined, when this limit will be reached, then the miners will no longer receive coins rewards from mining a new block.

3.1.1 Empty Blocks

In **Bitcoin is perfectly legit to have empty block** (no transactions except the coinbase transaction), this can be used to consolidate the previous blocks. However, **the time and effort needed to compute the hash of an empty block is still the same.**

3.1.2 Block Hashing

The **puzzle** that the miners have to solve in order to compute a block, consists into **finding the nonce** (word of 32 bits) **to inject inside the block's header, such that the binary digest of the block that contains that nonce, is lower than a given target** (threshold), in particular the N most significant bits should be equal to 0. **The higher is N , the more difficult is to find the right nonce.** Since the **possible numbers that the nonce can assume are finite, we can slice those possible values and assign them to different machines** to speedup the whole process. Sometimes, **ASICs** (Application-Specific Integrated Circuit, these circuits can be programmed only one time by operating directly on the board) **can be very effective as nodes to find the nodes**, since they **don't have main memory** so they are **cheap and consume less** than a normal computer. The hashing function used to hash the blocks is **SHA256**.

3.1.3 Block Time

The **block time**, so time needed to compute a new block by solving the puzzle, can be adjusted by the blockchain by changing the threshold used. A **shorter block time** implies that **transactions are validated more rapidly, but this will generate more forks so more effort is required to reach the consensus.** On the other hand, a **longer block time** means that the **consensus can be reached more easily** since nodes have more time to do it and probably less forks. Moreover, if the **block time is high** means that a more computational power is required to solve it, hence **attackers cannot perform the 51% attack easily.** In **Bitcoin the block time is 10 minutes** (derived from hard-encoded variables). **The difficulty of the puzzle is changed every 2016 blocks (≈ 2 weeks) to keep the average block time fixed, this is called retarget.**

3.1.4 Considerations on PoW

Given that mining a new block in a blockchain involves significant computational effort and comes with associated costs such as electricity and hardware expenses, and also considering that verifying the validity of a block is a relatively simple task, **if a node attempts to cheat by creating fake blocks, it would incur financial losses.** This is because the dishonest node would not receive rewards and would have to expend resources to solve the computational puzzle.

This **consensus algorithm guarantees fault tolerance, attack resistance and also collusion resistance** (some miners and validators collaborate to manipulate the blockchain).

3.2 Consensus

If two chains are both valid, in order to have a longer chain we can choose the highest cumulative work. Typically, this involves picking the longest chain, but not always. The cumulative work is the sum of all work put in the entire chain, the work for a block is the expected number of hashing attempts needed to find a valid PoW block:

$$\frac{2^{256}}{\text{target} + 1}, \text{ with target} = \text{number of valid digests}$$

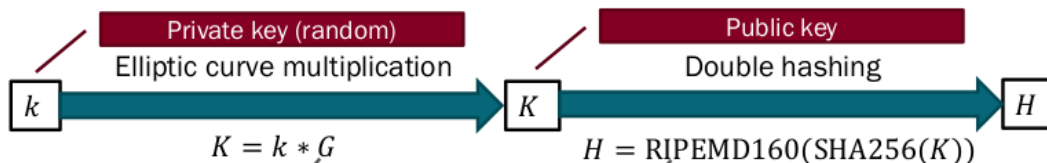
We call **stale** (or **orphan**) those blocks for which the PoW has been solved, but they didn't become part of the chain.

We can also define **difficulty that is inverse proportional with respect to the target**, in fact, the higher is the target the lower will be the difficulty (higher target means that the threshold is higher, hence also the number of possible valid digests is higher):

$$\frac{\text{target}_{\text{MAX}}}{\text{target}}, \text{ where target}_{\text{MAX}} \text{ is the highest target possible (so the one that leads to the lowest difficulty admitted).}$$

3.3 Signin

Every user has its own private key used to sign the transactions. The private key k is a 256 bit (32 byte) random number generated by the protocol. The public key K is used to verify the identity of the users and is obtained starting from the private key through the Elliptic Curve Digital Signature Algorithm (ECDSA). The public key is a 33 byte number, and the ECDSA is one way operation. Then using the public key, we can also obtain the fingerprint (we will use it later for the address calculation) by applying to different hashing functions consecutively. We first apply SHA256 (which produces a digest of 256 bits) and then the resultant digest is provided as input to RIPEMD160 (which produces a digest of 160 bits).



Base64: is composed by 26 capital letters + 26 lowercase + 10 numerals + '+' and '/'.

Base58: is the Base64 without the 0 (number zero), O (capital o), l (lower L), I (capital i), and the symbols '+' and '/'. This is done in order to avoid confusion for human.

The address is composed by following these steps, called $\text{Base58Check}(\text{version}, H)$:

1. **Version:** it is a single byte that specifies the address type or network. For example, in the main Bitcoin network this byte is usually 0x00.
2. **Compute the checksum:** $\text{SHA256}(\text{SHA256}(\text{version} \cdot H))$, where the dot stands for appending the H to the version.
3. **The final address** is obtained: $\text{Base58}(\text{version} \cdot H \cdot \text{checksum}[0,4])$, $\text{checksum}[0,4]$ are the most significant bytes.

We will have one unique address for every public key. Moreover, every public key is associated to only one private key (since is obtain through it). I cannot produce a signature (generated using the private key) that is authenticated from a public key.

The Base58Check is a two-way function, therefore we can obtain the public key starting from the address (it is sufficient to remove the last 4 bytes that are those relative to the checksum and take the next 160 bits).

The **signature is message-specific**, therefore a new signature is computed every time that the user has to use it. The **signature is computed starting from the private key and can be verified by the public key**.

3.4 Privacy

As said before, **users do not use their real names but they use pseudonymous** instead. However, even if we don't know the real identity of the user, **we can always mark an outlaw user since he will be identified through his signature** (that we know is unique). When someone becomes well-known on the blockchain for being an outlaw, all the transactions in which that user is involved will also be marked to notify all users in the chain about it. Since a transaction can be also signed by more than one users, the chain could consider a single owner for those two accounts (or at least the same organization, they trust eachother to the point to share their private keys).

3.5 Full Nodes and Light Nodes

Does every node need to download the full blockchain? No, there are **light nodes** that hold only the block headers, download the branches of the chain associated to relevant to them. These nodes are also called **SPV nodes** (Simplified Payment Verification). **A possible threat could be if too many nodes become SPV nodes, since they are demanding to other nodes the blocks verification and that nodes could be malicious ones.**

3.6 Script

Script is the **scripting language used in Bitcoin**. In particular, it is a **Non-Turing complete language used in UTXOs to create conditions that must be met in order to spend that UTXO**. We can use Script in both input and output:

- **locking-script (ScriptPubKey)**: It's part of the output script of a transaction. It specifies the conditions that must be met to spend the funds locked in a particular output (UTXO). The locking script is typically created by the sender.
- **unlocking-script (ScriptSig)**: It's part of the input script of a transaction. It provides the data and instructions needed to satisfy the conditions set by the locking script. The unlocking script is created by the person who wants to spend the funds (the transaction's sender) and is included in the input script, which corresponds to the UTXO being spent.

The elements provided by the unlocking-script are stored using a stack. We can use the **combination of locking and unlocking scripts to verify that the spender is the one that spend those UTXOs**:

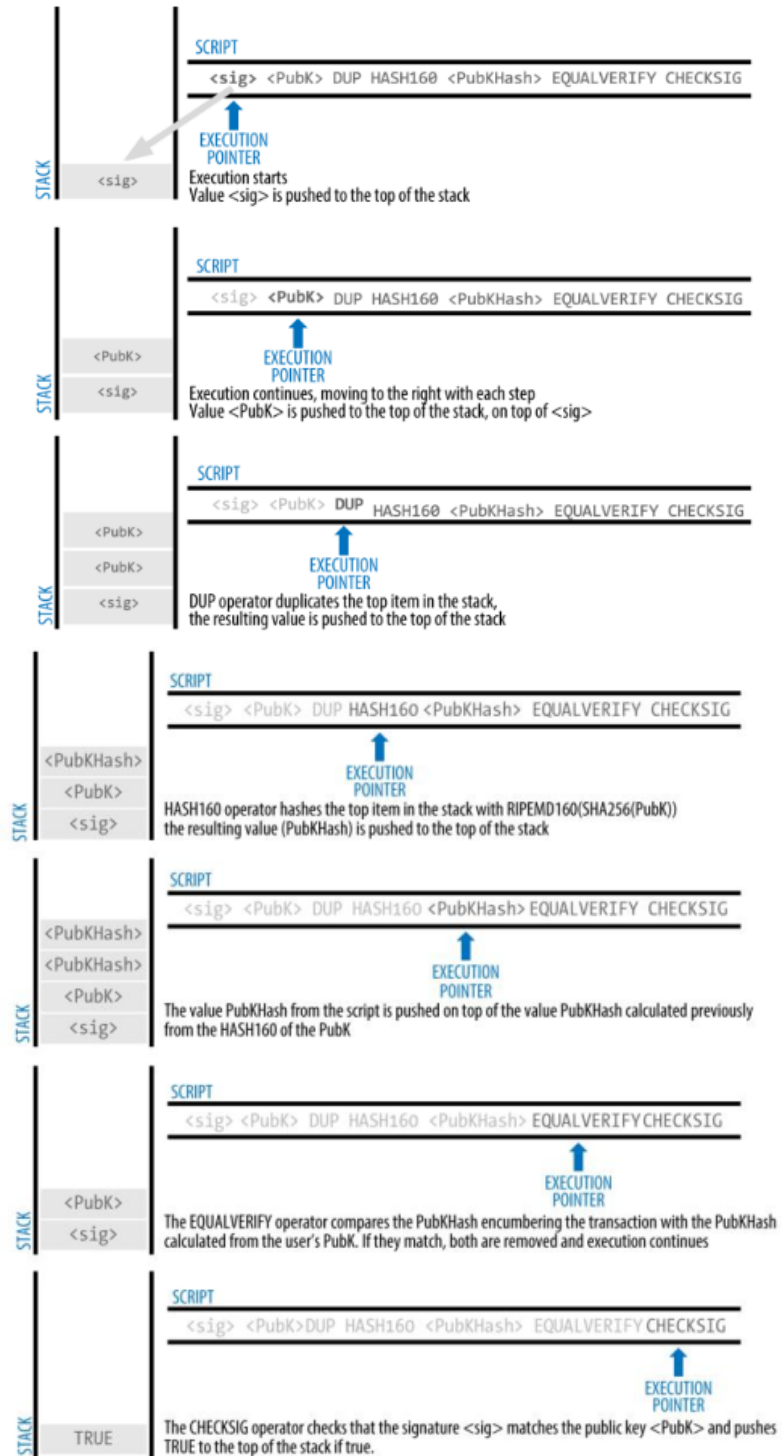
1. Locking-script:

```
OP_DUP OP_HASH160 <PubKHash> OP_EQUALVERIFY OP_CHECKSIG
```

2. Unlocking-script:

```
<sig> <pubk> OP_DUP OP_HASH160 <PubKHash> OP_EQUALVERIFY OP_CHECKSIG
```

The <sig> is the receiver signature and <pubK> is the public key.



We can also create a **multi-signature check, if the transaction is signed by N accounts** (so it belongs to every of those, they must trust each other), **we can allow the spending of that transaction if at least M out of N signatures are provided**. For instance, if the transaction is signed by 3 accounts, we can ensure that at least 2 of these two accounts wants to spend that transaction.

3.6.1 Script Limitations

As we said before, script is Non-Turing complete, hence **we don't have loops**. Moreover, since the UTXO must be entirely spent (we cannot spend a part of them keeping the remaining amount), we cannot use Script to control the withdrawn amount. We neither impose limitations on the timestamp since we're in a distributed system, therefore there is no global clock.

4 Ethereum

4.1 Accounts

In Ethereum **accounts keep their own local state**, they can be owned by:

- **Externally Owned Accounts (EOA)**: also known as simple accounts. They are controlled by private keys (like Bitcoin) and they're capable to send transactions. An account contains the balance in ETH (unlike Bitcoin in which the balance is derived by the blockchain).
- **Contract Artifacts (CA)**: It's an immutable code, due to the fact that is deployed using a transaction (that is immutable since the blockchain is immutable too), which can be triggered by humans sending a transaction or other contracts by sending a message. They have no owner and store their own state and cannot send transactions.

Therefore, **Ethereum adopts a balance model**, unlike **Bitcoin that adopts a transaction model**. Since Ethereum uses a balance model, the **balance is a number expressed in Wei's**. The **granularity achieved using Wei's is much higher** rather than the one achieved by the Satoshi's in Bitcoin.

4.1.1 Addresses

The **accounts are identified through a 20 bytes address**, it's derived by the **public key or contracts using the Keccak-256** one-way hash function. **Addresses are expressed using a mixed-capitals encoding called EIP-55** (Ethereum Improvement Proposal 55):

1. **Hash the lowercase address**, without the prefix 0x.

2. **Capitalize each alphabetic character in the original address if the corresponding hex-digit in the obtained hash is greater than or equal to 8.**

Note that since the **Keccak produces a 256 bits digest and the address are 160 bits long, we consider only the first 20 byte of the Keccak digest** during the capitalizing. E.g. We have the address 0x001d3f1ef..., the hash of this address is: 23a69c16..., our final EIP-55 address will be 001d3F1e... As we can see the 4th character 'd' has not been capitalized since its corresponding hash hex-digit is '6', instead the 6th character 'f' has been capitalized since it was associated to the digit 'c' that is higher than 8.

We can **double check the validity of an EIP-55 address** by taking the lowercase version of the EIP-55 address, compute the hash using Keccak-256 and see if the provided EIP-55 address coincides with the one calculated. This **allow us to avoid that the address specified by a user is not correct**, so the user is **sending the money to a non existing address**.

4.1.2 Nonce

The **nonce in Ethereum is different from the Bitcoin's nonce concept**, in fact:

- For **EOA** the **nonce** is the **number of sent transactions**.
- For **CA** the **nonce** is the **number of contract-creations made**.

Transactions have the nonce specified inside, every transaction that has a nonce beyond the current count is discarded, only those with a nonce ahead of the current count are kept.

The **nonce is used to infer an order on the transactions sent by an account**, in this way even if two transactions are recieved in a different order (we are in a distributed system!), we can understand which of those two has been sent as first. Since anyone can see the transaction in the network, an attacker could just copy and paste and existing transaction that would be accepted every time. With the nonce this is not possible since all the copies are automatically refused. Obviously, an attack cannot change the nonce in the forwarded copies since the transactions hold the sign that is document-specific.

In **Bitcoin we don't need the nonce since transactions are transferring the UTXOs**, if the sender cannot prove using his sign that the specified UTXO belongs to him, then he cannot spend it at all.

4.2 Contracts

A contract account is **deployed through a transaction sent to the address 0x0**. The **byte-code of the contract is sent in the data payload**. The **address** of the newly deployed account is computed by taking the **first 20 bytes of the digest produced by applying Keccak256**

to the RLP encoding of deployer's account address concatenated with the deployer's nonce (with the EIP-55 capitalization). In a contract account, every time the contract account receives a message, its code gets activated allowing to:

- Read and write to internal storage.
- Send other messages.
- Create contracts.

The code is executed in the Ethereum Virtual Machine EVM. This VM does not depend on any programming language, in fact, it can be used with ANY programming language since it will be compiled in the EVM bytecode.

There is a cost to be paid to execute the programs on the EVM, it is called gas. The gas is not measured in ETH, due to the fact that ETH is exchanged on platforms and therefore the value of a single ETH changes over the time. The gas indicates the cost of every operation, that is always the same, so it is a currency used to indicate how much an execution of that program or transaction costs. The operations that have a higher gas cost are those which require to store data.

When the contract's method 'self-destruct' or 'deactivate' is called by the owner of the contract, all the ETH in the contract balance are sent to the owner itself.

4.3 Transactions

Transactions must be signed by the sender. Contracts cannot send transactions since they should sign them, therefore the private key would be insert as static field inside the code that will be visible to everyone. However, the contracts can invoke other contracts' functions that are not visible in the blockchain, it is said that these messages are never serialised. Transactions represent the transition from an old state to a new one.

Obviously, the sender also have to specify the amount of Wei to be transferred.

Serialisation: Serialisation means storage in a persistent state. Deserialisation is the opposite operation, thus, loading from the memory the data and decode them. Ethereum uses the RLP (Recursive Length Prefix) to serialise the transactions.

Marshalling: Marshalling means turning in- memory objects into readable, structured formats (say, XML, JSON, ...). The opposite operation is called unmarshalling.

4.3.1 Gas

Before London Upgrade: Every transaction sent before the London Upgrade (**August 2021**) was specifying also the **Gas Limit and the Gas Price**, so how much gas can be at most be paid by that transaction and how many ETH corresponds to a gas unit. This **allows to solve the halting problem**, if a contract at run-time won't stop it will run out of gas at a certain point. The limit is specified since **we cannot predict the gas cost of an algorithm, it can be seen at run-time**. All the **unused gas are refunded**. Obviously, **the higher is the gasPrice, the higher is the chance that our transaction is mined relatively soon**.

After London Upgrade: The Gas Price is no longer existing. Now for every gas unit the price in ETH is given by the **base fee + priority fee**, where the **base fee is chosen by the protocol and the priority fee is chosen by the user**. The **base fee will be burnt** (none will receive them), the **priority fee will be given to the miner/validator**. The base fee is calculated by the protocol such that, if the network **demand is increasing too much, the base fee is increased**. If on the other hand, the **demand is decreasing, the base fee is decreased to encourage the usage of the platform**. The **maximum increase/decrease percentage is 12.5%**. The **base fees*gas unit amount is burnt to reduce the ETH circulating, thus adjusting the inflation/deflation mechanism**. The **network usage is given by the usage of blocks, given by the sum of the gas units** derived by the operations of the transactions stored in the block itself. The **maximum amount of gas units per block is 30M** (≈ 6 smart contract deployments) and the **target is 15M**.

The **gas is very useful to make Denial-of-Service attacks very expensive**, the attacker that want to submit a large amount of operations has to pay them.

4.3.2 Transaction Execution

In order to execute the transaction we have to:

1. **Check if it is well-formed:** Check the signature, if the nonce is correct and all the numbers are correct.
2. **Compute and apply transaction costs:** Calculate the transaction fees and subtract them from the sender's account balance. After that, the sender's nonce is incremented. If there is not enough balance to spend, an error is returned.
3. **Initialize the gas:** Add the initial gas units for the payment of the transaction itself.
4. **Transfer the transaction value to the receiver's account:** If the receiving account does not yet exist, then it's created (contract deployment). If the account is a contract, its code is ran.

5. **Finalize the transfer:** If the code fails since the sender has not enough funds or the code runs out of gas, then all the changes are reverted except the payment of the fees. If the code is executed and terminates regularly, refund the remaining gas to the sender and send the fees to the miner.

4.3.3 Transaction and States

Since transactions represent the state transition functions, the results of these functions are the **states**. There exist **full/archival nodes that store all transactions and the resulting state transition locally**, even the old state that are no longer valid. Conceptually, the blockchain data can be splitted in:

- **Chain Data:** list of the blocks forming the chain. These data are fundamental since every user can check the validity of the whole blockchain. By storing the whole chain it is impossible to tamper it without that the altered block is spotted. Any block modification will alter its hashing and since every block points the hash of the previous block, it will break the chain.
- **State Data:** result of each transaction's state transition. They can be discarded, they're said to be ephemeral data, allowing the blockchain networks to reduce storage requirements by discarding data that is no longer needed for the validation and verification of new transaction. This mechanism is called **Pruning**.

4.4 Consensus Protocol in Ethereum

4.4.1 Old Ethereum Version

In the previous version of Ethereum, the protocol was using a modified version of the PoW. The **Ethash algorithm** was an improvement of classical PoW against mining hardware optimizations. **It's a memory-intensive algorithm that cannot be bruteforced using ASICs**. The key of this algorithm was a **1GB Directed Acyclic Graph (DAG) file that was created every 30k block or 125 hours**, this period was called **epoch**. The **block time** for this protocol was approx. **15 seconds**. The main problem of the **PoW was related to the intensive energy usage needed by the mining rig to compute the new blocks**, therefore the solution adopted in Ethereum 2 is to the **switch from PoW to Proof of Stake protocol**. Eth2 is fully compatible with the old Ethereum.

4.4.2 Proof of Stake

The **stake is an amount of crypto-commodities invested into the system**, in particular, they **cannot be spent** and are **locked using a special transaction or by sending them**

to a specific address. The idea is that, **the more the users left in stake, the less is the probability that they would subvert the blockchain.** At a high-level, the proof of stake works as follows, the validators place a "bet" on the chain(s) and if the block is appended then they are rewarded with a reward that is proportional with respect to the bet. Unlike PoW, there is no puzzle to be solved. In Bitcoin there are no punishment mechanisms, since if the miners act against the chain they will not be rewarded and therefore they are implicitly punished by wasting time and electricity. **To avoid the "Tragedy of the commons" problem, validators bet on every chain, in Ethereum 2 the slashing mechanism is used.**

4.4.3 Casper the Friendly Finality Gadget (Casper FFG)

Casper FFG was introduced with the EIP-1011 for an hybrid transition from PoW to PoS. In detail, it has introduced the **slashing mechanism** in which, **if a validator votes for two conflicting forks (Tragedy of the commons), part of its deposit is burnt and it is immediately kicked out from the validator set as punishment.**

4.4.4 Eth2

The **PoS chain is called beacon and act as synchronizing backbone for 64 shards** (side chains) **that roll-up with it.** The beacon was merged with the PoW chain during the merge step in the updating cycle from Eth to Eth2. In the PoW the miners put their capital at risk by expending energy, instead, in PoS the validators explicitly stake capital in ETH. **Every validator has to put 32ETH (\approx 45k euros) at stake** (that **can be slashed** because of misbehavior), **if the user has not enough funds to become a validator, he can still join the validator set through the stacking pools.** In the stacking pools, several users put their ETH in a common stake to become validators. **A validator is responsible for checking that the propagated blocks are valid and (occasionally) creating and propagating new blocks.** Each validator send attestations (votes) for a block they deem valid across the network.

4.4.5 Casper FFG + LMD-GHOST (Consensus)

The **LMD-GHOST is a choice algorithm** (Latest Message-Driven Greedy Heaviest Observed Sub-Tree) that in most cases is not required to be run. It's **necessary when validators have different views of the head of the chain due to either network latency or block proposer has proposed two blocks.** The algorithm identifies the **fork with the greatest accumulated weight of attestations, if there are more attestations of the same validator, only the latest one is considered.** The weights are given by the stake involved with the attestations.

In the Ethereum's PoS the **tempo is fixed**, we have **slots of 12 seconds.** In every slot at least **128 validators are randomly chosen as committee, among those a single validator**

is picked as proposer that has to propose a block within 6 seconds. The committee has to vote for that block if it will be the next block or not. An epoch is composed by 32 slots (≈ 6 min 24 sec), the first block of the epoch is a checkpoint. Every committee must be composed by all different nodes within the same epoch, hence a total of $32 \cdot 128 + 1 = 4097$ different validators for every epoch. Why 128? Since is the power-of-2 round-up of 111, that is the minimum number ensuring that we will not pick $2/3$ malicious node in the validator set (assuming that we have at most $1/3$ malicious node in the network). Attesters have to vote for pairs of checkpoint, the recent one is called target and the older one is called source. If the $2/3$ of the total stake vote for the target, then it becomes justified, therefore the source (that was already justified) becomes finalised. Finalised blocks cannot be changed, in fact, when a block become finalised, all the previous blocks will be considered as main chain. A supermajority link must exists between the finalised block and its descendant block that becomes justified. The consensus is reached by looking the cumulative attestations weights. Very likely happens that every EBB is justified and then finalised, however it is totally legit to have EBBs that are "skipped". This is called **k-finalisation**, where k is the number of steps between a supermajority link (if $k=1$, then the source and the target are consecutive EBBs, if $k=2$ then an EBB has not been justified).

A validator can voluntarily exit after 2048 epochs (≈ 9 days), their stake can be withdrawn after 4 epochs. If the validator is slashed the delay is 8192 epochs (≈ 36 days) (correlation penalty + immediate penalty, see next sections). The stake withdraw was introduced in April 2023.

Every 256 epochs, a sync committee of 512 validators is randomly assigned. Every member signs the block headers every time that a new block is produced (hence, each slot), therefore they "glue" together the blocks between the finalised blocks. The proposer will collect the signatures and aggregates them into a final sync committee signature. All the committee members' public keys are store inside the header block. In this way the light nodes can simply take these headers to represent validated blocks rather than check the whole chain to ensure which one is validated or not.

Chain Instabilities: All these mechanisms are used to avoid as much as possible chain instabilities (switches from a chain to another one). Malicious nodes should own $2/3$ of the total ETH to produce changes in the original chain.

4.4.6 Rewards

We have 3 types of rewards given at each slot:

- **Attestation rewards:** correct attestation for source/target checkpoint and block attestation.

- **Sync committee rewards.**
- **Proposer reward for an attested block:** head inclusion, source inclusion, target inclusion and sync-committee inclusion.
- **Whistleblower.**

4.4.7 Slashing

Dishonest behaviours are subject to slashing:

- **Double proposal (equivocation):** A proposer signs two beacon blocks for the same slot. If the proposer doesn't propose a block no slashing mechanisms are applied.
- **FFG double vote:** An attestator votes for two checkpoints in the same slot.
- **Surround vote:** An attestator votes for a checkpoint that surrounds another one. Hence, the source epoch of the first block precedes the source block of the second, but the target epoch of the first block follows the target epoch of the second. In this case, the attester will contradict what a validator said.

These three violations are detected through the **whistleblowing**, so when a validator propagates the offense to the proposer that will insert it inside the block. Both **whistleblower and proposer are rewarded**. The **slashed node is also kicked out from the validator set**. We can also have penalties, that simply remove some currencies from the stake without being kicked out:

- **Inactivity Penalty:** If the chain fails to finalize more than 4 epochs.
- **Sync Committee Penalty:** For not participating, or signing the wrong head block.
- **Correlation Penalty:** Attititional penalty inferred by the protocol if many validators have been slashed near the same time: **validator_balance*3*fraction_of_validators_slashed**. Therefore, if **1/3 validators are slashed within a similar period, they will lose the entire balance**.

4.4.8 Effective Balance

The "weight" of the attestations are proportional with respect to the stake of the attestator. In detail, what is considered is not the actual stake (or balance) of the node but the effective balance. It is always an integer value such that, to increase by one the worth of the balance, the balance must be $\times 0.25$ (example: 30.1ETH is rounded to 29, 30.25ETH is rounded to 30). To decrease the balance it is worth to be under $\times 0.75$ (example: suppose that our effective balance is 30ETH with

a balance of 30.25ETH, then we are penalized by 0.5ETH. Our balance becomes 29.75ETH that is still 30ETH, but if we were penalized by 0.51ETH then the balance becomes 29.74ETH and the effective balance is rounded to 29ETH). This is the **hysteresis zone**, in which the effective balance does not change at all.

5 Hashing

A cryptographic hash function h must be:

- **Preimage Resistant:** Given a hash k should be difficult to find any message m that produces that hash.
- **Second-preimage Resistant:** Given a message m_1 should be difficult to find another message $m_2 \neq m_1$ such that they have the same hash.
- **Collision Resistant:** Should be hard to find two messages $m_1 \neq m_2$ ((m_1, m_2) is called **cryptographic hash collision**) such that they produce the same hash.

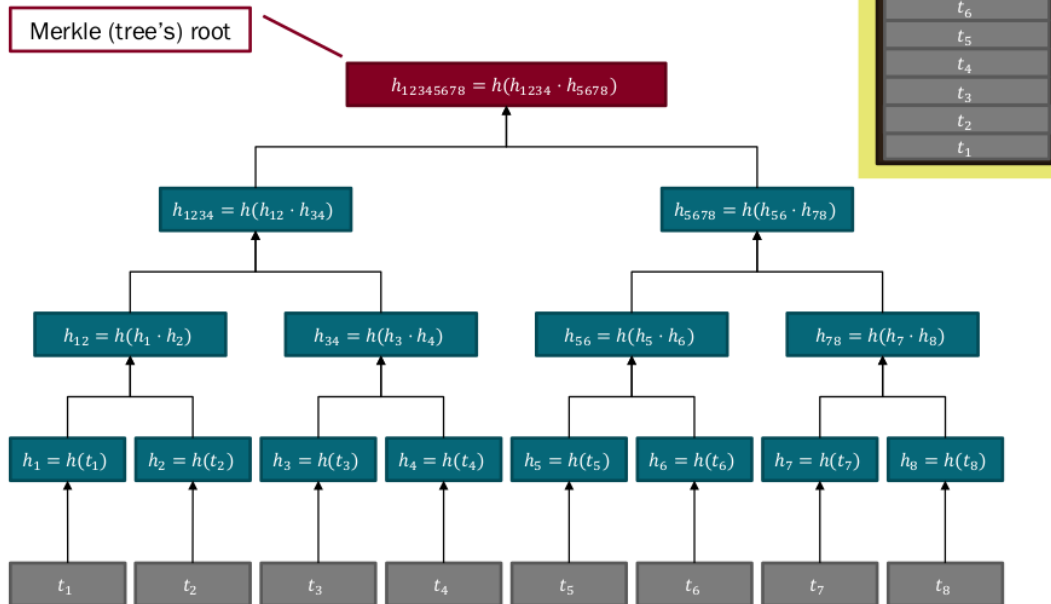
In general we can say that, **the larger is the digest in term of bits, the much resistant will be the hash function.**

SHA-family uses binary operations to obtain the digest.

5.1 Merkle Trees

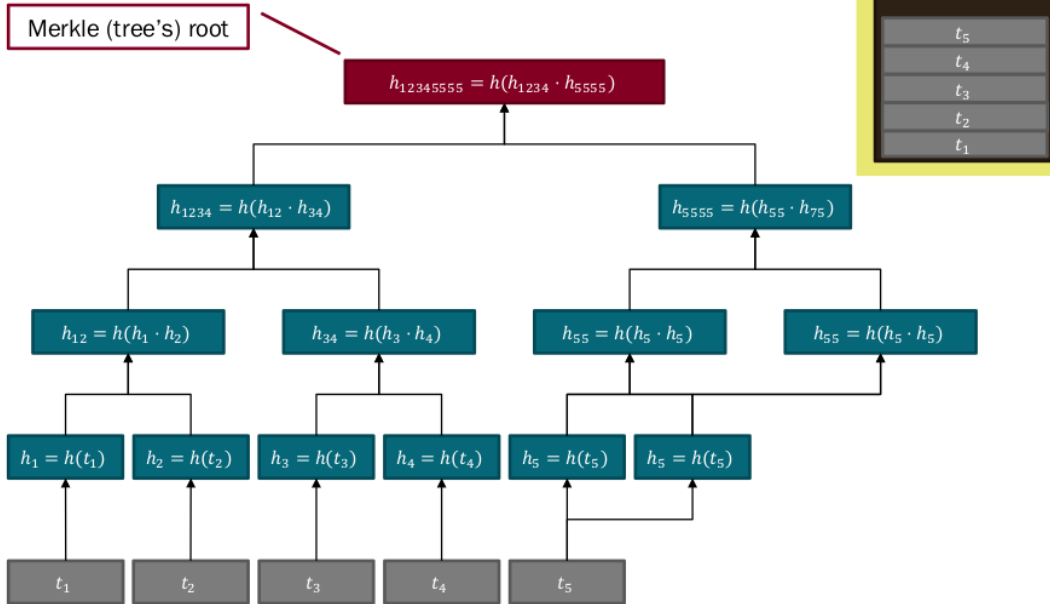
In Bitcoin the hash of the whole transaction set is not simply obtained by hashing the set itself, in fact, we have to keep the order of the transactions. **The hash of the transactions is obtained as a binary tree over the single hashed transactions:**

(Binary) Merkle trees and Merkle roots in Bitcoin



If a light node wants to verify the transaction t_2 , the full archival node must send $h_1, h_{3,4}, h_{5,6,7,8}$.
 If we do not have 2^n transactions, the last transaction is repeated in the tree until reaching 2^n .

What if we do not have 2^n transactions?



5.2 Patricia Radix Trees

In Ethereum we need 3 roots:

- **State Root:** Saving all the states (composed by the single account states) would be too expensive. Therefore, the state root are used to represent them.
- **Receipts Root:** These are the receipts of the transactions, therefore are used to determine the effect of the single transactions.
- **Transaction Root.**

All of them are included inside the execution payload header.

6 Solidity

6.1 Tokens

Tokens are crypto-assets that can have multiple functions:

- **Currency:** They can be exchanged for goods and services.
- **Commodity:** Basic good tradeable of exchangeable with other goods of the same type.
- **Utility:** Satisfaction quantifier for an economic good or service.
- **Security:** Financial instrument that guarantee ownership, credit or decision power.

Tokens can be distinguished in three classes:

- **Fungible Tokens:** Individual units can be mutually substituted.
- **Non-Fungible Tokens:** Units are unique and not interchangeable.
- **Semi-Fungible Tokens:** Fungible until redeemed or expired, then becomes unique hence Non-Fungible.

The NFTs are an example of Non-Fungible tokens.

6.2 Solidity Types

- **address:** 20 byte value that contains the address of an account. Through
 - balance: we can access the account balance.
 - transfer(uint): we can send cryptos to the account.
- **bool**
- **int & uint:** we can specify with steps of 8 their size (uint8, uint16, ...).
- **Fixed-size byte arrays** (bytes1,..., bytes32).
- **string:** can be written with both "" or ", we should avoid them since are very expensive to be stored in the chain.
- **Arrays:** we can instantiate an x fixed-length array of type T[x] or of variable length using T[].
- **mapping(K => V):** a map from elements of type K to elements of type V.
- **Struct:** set of variables of mixed type.

We have some variables that are already defined in a contract like msg that is the transaction or block that is the block in which the transaction is included.

6.3 Solidity Functions

6.3.1 Function Visibility

A function must be declared as:

- **private**: can be seen only from the contract itself.
- **public**: can be invoked by everyone.
- **internal**: can be seen by the contract and the ones that derive from it.
- **external**: can be called only from the outside of the contract.

6.3.2 Data Locations

When we specify an input or output for a function, we also have to declare where it will be stored. From the less to the most expensive:

- **calldata**: non-modifiable and is function-scoped.
- **memory**: modifiable and is function-scoped.
- **storage**: modifiable and persist for the smart contract lifetime.

6.3.3 View and Pure

When a function does not modify the contract' state but it accesses the memory, to read a value for example, it must be declared as **view**. If a function, does not even read on the memory, it must be declared as **pure**. View and pure functions are **less expensive**.

6.3.4 Exceptions

We have 3 main ways to manage exceptions:

- **assert**: should be used only for debug purposes.
- **revert**: is like throwing an exception.
- **require**: we can specify a condition that must be met otherwise throws an exception.

6.4 Decentralized Applications (DApps)

A **DApp** is a software composed by a **frontend running on the local computer** and a **backend running on the decentralized peer-to-peer network** (e.g. CryptoKitties and Dock.io). The **HTML, CSS and JS** are retrieved from the web server. In order to **access the blockchain**, we can use the **web3 javascript library**. Thanks to that library, we can access the Ethereum blockchain's accounts and even smart contracts. Each **smart contract has an ABI (Application Binary Interface) attached to it that exposes the elements** of the contract itself (such as name, functions, parameters, ...). We can even **subscribe ourself to events by using the event listener** provided by web3. Those event that are broadcasted by the chain are being captured by the event listener.

6.5 Truffle and Ganache

Truffle is a platform used to manage the backend of the **DApps**, it allows to initialise new projects and it can **compile, link and deploy smart contract on the chain**. **Ganache** is a **personal blockchain for rapid Ethereum distributed application development**.

6.6 MetaMask

Allows Ethereum DApps to be run in the browser without the need to run a full Ethereum node (it implements a SPV node).

7 Seminars

7.1 Blockchain based Resource Governance for Decentralized Web Environments

The user's data can be considered as gold for companies that can use them without asking the consensus to users. **Solid has been proposed by Tim Berners Lee and its goal is to decentralize the web by giving users the ownership of their data**. **Solid Personal Online Datastore (POD)** is a storage where the user can put all his data by specifying the **Access Control** of those data (who can access the data). The **access control is not enough** for modern platforms, so we can use **Usage Control that imposes constraints that must be met by the consumer of the data in order to allow the access**. Examples of usage control constraints are:

- Number of accesses

- Geographical limitations
- Time limit
- Purposes of the access

ReGov (Resource Governance Framework) is a platform that uses Solid with Usage Control for exchanging data. Each node runs on a user device and is composed by three main components:

- **Data Provision Component:** This component manages the communication between the user and the Storage Manager in order to save data in the Resource Storage. It also manages the received access requests sent by other users by monitoring the usage of those provided information.
- **Data Consumption:** This component is used to create access requests for retrieving the data from other users. The data are not directly accessed by the user by through the Data Manager that enforces the constraints.
- **Governance Interface:** This component manages the outgoing and incoming messages.

The **Governance Ecosystem is the part of the platform that provides the metadata of the data saved in the system and manages the policies attached to them.** It is **deployed in the EVM based blockchain** since the indexing step is done on the chain in order to certify all the data that are published on the platform.

7.2 Decentralized Oracles

Oracles are component that act as bridge between the blockchain and the real world. They are composed by an **on-chain tier (smart contract)** used to manage the interaction between the chain and the oracle and an **off-chain tier (web3 components)** used to manage the interaction between the real word and the oracle. **Typically the on-chain component notifies the off-chain components using events that are caught by those.** The key features of oracles are:

- **Reliability:** They must be secure in both sides of the communication.
- **Shared usage protocol:** They must implement a protocol that can be used in both the communication directions.

We can **classify oracles** in:

- **Data Oracles:** They can be either:
 - Software Oracles: Take the information from the internet and forward them to the blockchain.
 - Hardware Oracles: Take the information from the physical world and forward them to the blockchain.
 - Human Oracles: They forward their skills and knowledge.
- **Flow Oracles:** They are classified based on two actions: Pull and Push:
 - Pull in: Allows on-chain components to request information to the off-chain components.
 - Pull out: Allows off-chain components to request information to the on-chain components.
 - Push in: Allows off-chain components to send information to the on-chain components.
 - Push out: Allows on-chain components to send information to the off-chain components.
- **Trust Oracles:** They can be either:
 - Centralized Oracles: They are controlled by a central authority and provide information for a smart contract.
 - Decentralized Oracles: They can consider multiple provider information by keeping the decentralization of the blockchain even when we are out of the chain. However, they have an higher communicaiton overhead with worse performances.

7.3 Algorand

Algorand uses the **Pure Proof of Stake (PPoS)** which is based on the **VRF (Verifiable Random Function)**. The idea behind that protocol is about rolling tamper-proof dices in order to participate to the protocol's decisions:

1. **Block Proposal** (Phase 1): A set of proposers (≈ 100) is elected by running the VRF, each proposer will propagates both the block and the VRF output (used to prove that he is the winner of the "lottery").
2. **Soft Vote** (Phase 2): In this phase the nodes can verify which are the valid proposers thanks to the VRF output. The only accepted block is the one propagated by a valid proposer and with the lowest VRF hash. A committee is elected to vote the best proposal among the received ones, so there are no forks in Algorand since on every block the quorum must be reached.

3. **Certify Vote** (Phase 3): A new committee is elected to check the validity of the block avoiding double spending problems, overspending, etc...

Every node has a public key and a secret key. The private key is used to generate a pseudo-random number called **hash** and the verifiable associated proof π . When the hash is in the specified range for "winning the lottery" the account becomes part of the committee or a proposer (depending on the selected role for which the VRF has been run for). **The others node can use the proof and the public key of the node to verify if the hash was generated by that account, thanks to the verifyVRF. The more Algo are held by the account, the higher will be the probability of winning the lottery**, since every algo represents a produced hash so a user can even produce more than one valid hash (his vote will count for two or more). Algorand can **tolerate any number of malicious node as long as the honest users held at least 2/3 of the total stake in the system (Byzantine Agreement*)**. The malicious nodes don't know in anticipate the users to be corrupted or to attack.

The nodes can be classified in:

- **Non-Relay Nodes:** They can participate to the PPOS. They can communicate only with Relay Nodes. They can be have either a Light Configuration in which only the latest 1000 blocks are stored or a Archival Configuration where all the blocks are stored.
- **Relay Nodes:** They can communicate with both Non-Relay and Relay nodes and are used to reduce the communicaiton hops by routing the blocks.

To change the blockchain status in Algorand the transactions play a fundamental role, we have 6 types of transaction:

1. Payment
2. Key Registration
3. Asset Configuration
4. Asset Freeze
5. Asset Transfer
6. Application call

Each **transaction must be signed by one (or more) user or a smart contract. There are fees associated to them in order to avoid DDoS attacks, indeed, fees are not meant to be used as reward and are paid only if the transaction is included in a block (0.001 Algo)**. Every transaction have a validity of at most 1000 blocks. From a **random seed, both public**

and private key (both of 32 byte length) are generated. A 4 byte checksum is appended to the public key obtaining the address. The private key is used to generate 25 11-bit integers that are then mapped into English words obtaining the mnemominc phrase. Each account is associated to a local on-chain state, so they held their own balance.

The **Algorand Standard Assets (ASA)** are tokens the are defined in the protocol itself, **without the need of writing a smart contract**. Indeed, they're not identified through an address but using a 64-bit unsigned int. There are some default parameters that must be specified in order to create a new ASA:

- Total: Total number of base units.
- Decimal: The number of digits that can beused to represent a fraction of the unit.
- defaultFrozen: Used to freeze holdings for this asset.
- UnitName and AssetName.
- URL: URL where more information about the asset can be retrieved.
- MetaDataHash.
- ManagerAddress: Address of the account that can manage the configuration of the asset and destroy it.
- ...

The **Algorand Virtual Machine (AVM)** is a **Turing-complete execution enviromentment that runs on Algorand used to accept or reject the effect of transaction**. It works using a stack, on top of that only one non-zero value must be present in order to accept the transaction effect. **Code can be written in TEAL that is the original programming language or by using libraries of other languages that will be then translated in TEAL and then compiled in bytecode. A smart contracts are identified using an Application ID (not an address) and are composed by:**

- **Approval Program:** Handles all the request to the application (except of Clear Call)
- **Clear Program:** Handles the Clear Call to remove the smart contract from the account's balance record (free the local storage used in the account by the smart contract).

The **smart contracts can be updated** without the need to re-deploy a brand new contract. **Users have to explicitly make a OPTIN transaction to specify that they agree with the interaction with the application**, this prevent the spam of coins since users have to pay for their

local storage. In Algorand the **re-entrancy attack** (a contract call itself before the information is stored on the chain in order to change it) is **prevented by denying the possibility to an application to call itself**, even indirectly. Moreover, the application call stack depth can be at most 8.

In Algorand the smart contracts can use three different types of storage:

- **Local Storage:** Attached to the account and it is allocated when the account OPTIN to an application. It can include between 0 and 16 key/value pairs and the size (in term of k/v pairs) must be specified at contract creation. It be read by any application that specifies the app ID in the foreign accounts, but can be edited only byt the app and cleared by the user.
- **Global Storage:** Attached to the application and can contain up to 64 k/v pairs. It can be read by any application that has the app ID in the foreign accounts and can be edited only by the app itself. It is deleted only when the application is deleted.
- **Box Storage:** It is a dynamically allocated storage, it can be read and written only by the application that creates it. They must be explicitly deleted by the application since are not removed at the application termination.

This space must be paid with a rent that is represented by the minimum balance that every account must held (the lowest value is 0.1 Algo).

7.4 Frauds on the Blockchain

In order to trade cyptocurrencies we can use:

- **Centralized Exchange (CEX):** Centralized organization that requires identification to participate and allows to exchange also Fiat currencies (dollars, euros, ...). The order book matches the buyers and the sellers.
- **Decentralized Exchange (DEX):** Users are anonymous and only Altcoins and Stablecoins are available through a peer-to-peer trading.

7.4.1 Pump and Dump

Users join some channels (e.g. Telegram channels) in which other users decide when and which asset (in this context, the coin) should be rapidly bought. This indirectly increases the value of a cryptocurrency making the people outside the channel buying that coin too. **When the values is high enough, the scammers sells the coin by achieving a possible profit.** The admins of the groups buy in anticipate the coins at a very low price and sells them when the value is very high. The members of the groups depending

on how fast they are, buy the coins when the value is rapidly increasing trying to make a profit. The **outside investors always lose money** by buying the coin when the value is increasing without being aware of the fact that the value will drop after the dump operation. The **members of the groups can be notified some seconds before** the others by **being part of the VIP group** (e.g. by paying an access-cost or by inviting more users in the group).

7.4.2 Rug Pull

On the **DEX every user can create a token by deploying a smart contract**. In a **Liquidity Pool**, the creator of a token **put a certain amount of token and a certain amount of typically stable coin**. The value of the new token is derived by the amount of stable coins in the **Liquidity Pool**. The idea behind the **rug pull fraud is to create a token with a certain amount of liquidity, wait that some users start buying the token adding more liquidity in stable coins and then remove all the liquidity**. This phenomenon is **very frequent on both Ethereum and Binance Smart chain** and can be observed by looking at the 1-day tokens. These tokens are created by a very small percentage of the accounts in the system and are typically **bought by sniper bots** that aim to buy newly-created coins in order to make a profit by selling them.

7.5 Enabling Data Confidentiality with Public Blockchains

Blockchain data can be publicly accessed by everyone so data privacy among parties is a big issue.

7.5.1 IPFS

InterPlanetary File System, it a **distributed system for storage and access files**. It based **on content-addressing**, so the address of a file changes if the file changes. It can be used for storing large files that should not change over the time (otherwise the hash will change as well)

7.5.2 CP-ABE

Attribute-Based Encryption (ABE) is a type of public-key encryption. The **Ciphertext-Policy (CP)** is a ABE version in which **users (actors) are associated with the attributes and messages (the one that we want to cypher) are associated with policies**.

1. **Pre-Phase**: Public Key and Master Public Key generation
2. **Ciphering**: Plaintext + Policies and PK = Ciphertext
3. **Access Key Generation**: Attributes + PK and MPK = Access Key

4. **Deciphering:** Ciphertext + Access Key and PK = Plaintext.

7.5.3 CAKE (Control Access Key Encryption)

This system relies on two trusted parties: **Secure Data Manager (SDM)** and **Secure Key Manager (SKM)**. **SDM will store the data in IPFS and send the hash to the smart contract. SKM will decipher the data for the readers.** In the pre-phase the actor will send the attributes to the smart contract in order to save them. In the ciphering phase the data owner sends the data to the SDM that will store them on IPFS and put the hash in the smart contract. These data can be accessed accordingly with the provided policies. In the key generation phase, a reader will send a request to the SKM that generates the keys and retrieves the attributes from the smart contract. Obviously the PK is sent back to the reader via SSL to avoid sniffing. In the deciphering phase, thanks to the key, if the reader has the needed attributes, he will be able to decipher the data. The main drawback is the **single point of failure represented by SDM and SKM.**

7.5.4 MARTSIA

The proposed solution, MARTSIA, is designed to address the single point of failure in CAKE by eliminating two central authorities. MARTSIA is intended for use with both Algorand and Ethereum, utilizing CP-MABE instead of CP-ABE. Unlike CP-ABE, each attribute in CP-MABE must be certified by an authority.

During the pre-phase, the designated authorities are specified, and for each authority, a private-public key pair is generated. The access key is created using the authority's private key. In this approach, the data owner encrypts the message without a centralized authority (no SDM), and the reader decrypts the message independently (no SKM).

In the key generation phase, since there are no trusted parties, every authority is aware of the metadata of other authorities. This information is stored by each authority on IPFS and then placed inside a smart contract. This ensures that if any authority attempts to alter the metadata, the resulting hash will be different.

During key generation, each authority employs Elliptic Curve cryptography to generate a pair of numbers, hashing them afterward. The first hash is kept privately, and the second is made public. The non-hashed numbers are stored in the blockchain. This allows the detection of any malicious authority through the public hash. Authorities then calculate public parameters by XORing their public numbers with those generated by other authorities. All authorities generate the same public parameters, which are stored on IPFS, and the resulting hash is recorded on the blockchain. Using these parameters, authorities generate their private and public keys, with the public key stored on IPFS and its hash saved in the blockchain.

In the Ethereum version, all blockchain storage phases are executed through smart contracts. Conversely, in the Algorand version, these phases are implemented using boxes, which are public storages writable only by the respective authority that owns the box.

The key used to decipher the data is sent to the user either using a SSL connection, or using the blockchain. The reader asks for the deciphering keys to the authorities using a transaction, then the authorities will send back the partial deciphering keys encrypted with the public key of the reader (so he is the one that can decipher them thanks to the private key). The keys are partial keys since, every authority will certify a sub-set of attributes. The whole key is used with the public parameters to decipher the encrypted text.

The data owner uses the public parameter (retrieved by the blockchain), the authorities public keys, the policy to encrypt a symmetric key. The obtained cipher key is used to cipher the plain text. The ciphered text is put on IPFS and then on the blockchain.