# Distributed Systems Notes

Daniele Bertagnoli

2022/2023

# 1 Bitcoins

## 1.1 Description

Bitcoin is a cryptocurrency that can be used to achieve a full anonimity. It is decentralized, so there are no central authorities that manage these virtual coins.

## 1.2 Hash Functions

Hash functions are functions that have these properties:

- **Hard to invert**, hard to find $x \neq y$ such that $H(y) = H(x)$.
- **Hard to break**, hard to find $z$ such that $z = H(x)$.

Typically these functions are built using a very large domain and a small codomain: $H : A \longrightarrow B$. We don't know if hash functions exists, but we believe that some functions have the presented properties. Hash function are used inside the bitcoin's consensous protocol.

## 1.3 Public-Private Keys

Public-private key pairs can be used to:

- **Encrypt and decrypt messages**
- **Sign messages**

Each user holds a public-private key pair, in particular the public key can be shared with everyone, the private key must be kept secret. A private key is able to decrypt a message that was encrypted by using the public key, and viceversa. If we want to encrypt a message for a user A, we use his public key to encrypt the plain text. After that, A can use his private key to decrypt the received message. If we want to sign a message, we can encrypt a message by using our private key, now everyone can check that the received messages has been sent from us, since we are the onlyone that hold the private key which match with the public key used to decrypt the message.
Bitcoin uses these keys to sign the coins by using the coin owner's public key.

## 1.4 Blockchain

To send a bitcoin to another person, we have to create a "new file" that represent the bitcoin and change our signature with receiver's signature (using his public key). Each bitcoin contract contains an hash of the entire file, in this way we can verify the integrity of the whole file.

**Double spending problem** A person sends his bitcoins to two different persons, in this way we are not able to understand who is the real owner of that contract.

**Blockchain** To avoid the double spending problem, each transaction is inserted inside the blockchain. The blockchain is a chain of transaction blocks, each block is composed by:

- **Hash of the current block**
- **Hash of the next block**
- **List of transactions**
- **Random integer (Nonce)**

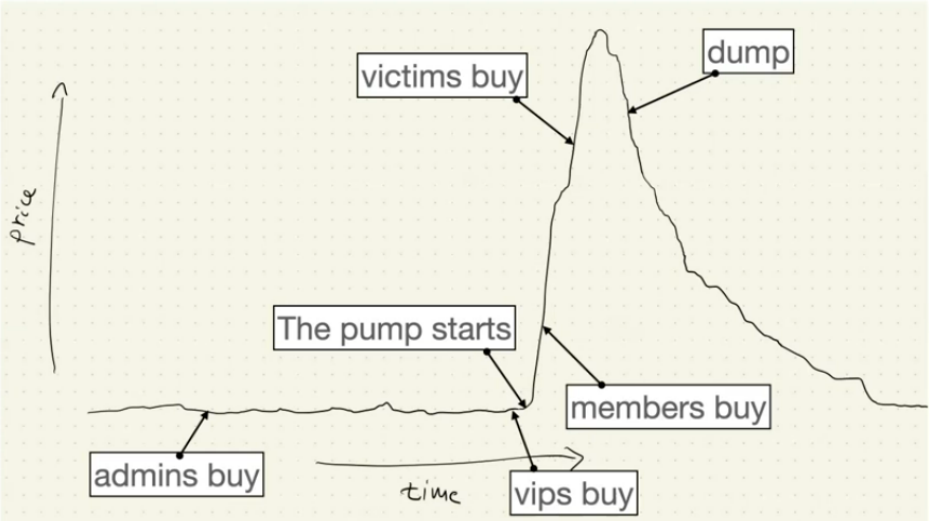
We can check if the received bitcoins are already in a transaction by looking the blockchain. Since the blockchain is decentralized, bitcoin uses a consensous protocol to manage the transactions (Paxos cannot be used since it's not live and it's not safe against byzantine failures). Bitcoins are based on the **proof of work** concept, basically each block to be considered as a valid block, needs that the last significant K bits of the hash are setted to zero. To achieve this property, each **miner** must use the brute forcing technique on the nonce value untile it achieve that property. Miners are computers that compute this work to gain small fees from each computed transaction. Each transaction block takes around 10 minutes to be computed (to reach this goal, the blockchain adjusts the K bits mentioned before for the proof of work property). Since the blockchain is a distrbuted system, can happen that the chain forks into two different branches. In this case each miner will continue on the longest branch, so it's very unprobable that these two branches are

updated simultaneously. After few blocks, one of those brances will become a dead branch, all the transactions that were part of that block will be added inside the transaction pool and will appear later in the blockchain. We can consider a transaction as "safe" after that at least 6 blocks are appended to the current block (so in approximately 1 hour).

# 2 Frauds in the cryptocurrency ecosystem

## 2.1 Pump and Dump

Some organized groups that choose a certain coin, the members start to buy these coins simultaneously. So they are inflating the coin's value. The people that are not part of the group does not know about that, so they will see a quick incresement on the market about a certain coin and they decide to buy that coin. After a few minutes, the group members will sell all the coins (in this moment the coin's value is typically very high), this means that the value will drop drastically. In this way, these scammers has bought coins with a low value and sold them by gaining a big profit. The scammed peaple has bought these coins when their value was already grown and after that all the scammers has sold their coins, these people have remained with coins that have no value. Admins and vips typically make a lot of moneys, the basic members in some times becoms also victims.




These groups typically use Telegram or Discord to organize themself, most of them has also an affilation program so you get more services if you invite more users inside the group. In some cases, these groups use a VIP hierarchy, so you can pay to get more services. More services means that these users knows earlier (some seconds) the coin that they must buy for the pump and dump.

Obviously, the first users buy the coins with a lower value since it grows very quickly. To detect these fraud we can monitorate the **Rush Holders** number, these users buy coins and sell them in few minutes. In a pump and dump fraud we can see around 0 rush holders for a certain coin but in a few minutes it grows up to hundred of users. So we can spot a pump and dump on that coin.

### 2.2 Scam Coins

Each user can easily create a new coin as a smart contract that implement liquidity pools. In particular, the user put inside a pool a certain amount of trusted coins (such as BTC or ETH) and define how many of the new coins correspond to the liquidity pool (for example 10k sapienza coin = 10 ETH). Other users will buy these new coins with the trusted coins, the amount of money inside the liquidity pool must remain the same, so when a user buys the new coin by putting inside the pool an amount of trusted coin, the new coins' value increses. Can happen that the new coin were a scam coin, so who has created the coin after a certain time will close the pool and keep all the trusted coins that he received. The users that have bought the scam coins now have lost all the invested money.

## 3 Privacy in the Internet Era

Some years ago when we want to connect to a new WiFi the connection was started by the smartphone that periodically was sending request to a certain WiFi network in order to check if it was available or not. To save energy, the smartphones sent every 60 seconds the whole list of the WiFi that it knows to check if there were any of them available. This list is not encrypted, because to encrypt a message we need to share a key with the other node, so this is a privacy problem since this list can be read form any one and they can understand usefull information. By looking this list we can understand where a person lives, how rich is that person, age, etc... We have another option that is more privacy oriented where the smartphones do not send the whole list of the known WiFi, but it simply sais to all the reachable WiFi that it is available. Now days the list is no more plain and even the MAC addres is no longer available.

## 4 TOR (anonimity in the internet)

### 4.1 Description

The achieve the anonimity in the Internet there are several techniques that can be used. For example, we can use a VPN which allows us to connect to an intermediate proxy to hide the connection. However, this can protect us against the ISP but not against the VPN provider. If we

use the incognite mode in the browser we need to trust the browser. To achieve a full anonimity we can use **TOR**. TOR's idea is similar to the VPN's technique, the client can choose a bunch of servers among a very large servers list. After that the client will follow a multi-path connections to reach the final destination.
CLIENT $\longrightarrow$ TOR SERVER: A $\longrightarrow$ TOR SERVER: B $\longrightarrow$ TOR SERVER: C $\longrightarrow$ DESTINATION
Each message is encrypted by using the **Diffie-Hellman Protocol**. The fist node is called **Guard**, so it knows the who is using the TOR service. The last node is called **Exit Node** and it knows the final destination. Obviously if a malicious user owns both guard and exit node can understand who is using the service and the final destination (this can be done through the *text analysis technique*). To avoid this problem, TOR chooses all the nodes from different part of the world. This is why TOR is called **Onion Router**. The major tradeoff is the connection speed that can be very low since our messages are forwarded through more than one connection. Moreover, can happen that one of the TOR's nodes can have a lower bandwidth than us, so we have a bottleneck in that point.

### 4.2 Diffie-Hellman Protocol

This protocol is used to obtain a simmertic encryption-decryption key. Two partecipants (A and B) choose a big prime value $g$, after that A chooses a value $a$, B chooses a value $b$. A sends to B the value $g^a$, B sends To A the value $g^b$. Now A can compute the value $[g^b]^a$, B can compute the value $[g^a]^b$. These two values are equivalent, and can b used to encrypt and decrypt messages. The value g cannot be obtained starting from $g^a$ or $g^b$ since is too hard to compute.

This value $K$ is used in TOR to encrypt the messages at each connection step. In TOR we use this technique insted of a public-private key pair beacause is more efficent. During the first connection, the clint will send the value $g^a$ encrypted by using the public key of the TOR' server. This guarantee that the value $g^a$ can be decrypted only by the server (this is used to achieve the anonimity in a secure way).

## 5 Algorand

### 5.1 Description

Algorand is a cryptocurrency which allows to use the the **smart contracts**, so piece of TEAL (programming language) code that can be executed directly on the blockchain. Each node use the P2P technique to communicate, in particular the used P2P protcol used is called **Gossip**. Each transaction is executed with small fee. Algorand uses PBFT in the blockchain to be safe against byzantine failures.

## 5.2 Protocol

We can split the protocol in 3 macro-categories:

- **Round**, a complete run of the protocol that ends with the block election.
- **Phase**, each round is composed by 3 phases.
- **Step**, happens when we broadcast a message.

Each node can be:

- **Observer Node (aka Standard)**, just listen the messages.
- **Proposer Node**, proposes new block.
- **Committee Node**, agrees on the proposed blocks.

The protocol is composed by 5 phases:

1. **Sortition**, all the nodes choose a role for this run.
2. **Block proposal**, each proposer broadcast a vote for a new block.
3. **Committee receives all the votes**, the committee group receive all the broadcasted votes form the proposers and start a **Bizantine Agreement** to choose the new block.
4. **Committee group agree on two values**, the committee group can agree on two different values:
    - **Block**
    - **Empty Block**
5. **Consensous is reached**, to reach a final decision the block must obtain at least $\frac{2}{3}$ of the votes. The consensous can be reached on two different alternatives:
    - **Final decision**
    - **Tentative**, we must run another round to choose a valid block.

### 5.3 Verifiable Random Function

During the Sortition phase of the protocol, each block must choose a role for the current round. Each node uses the **VRF** (Verifiable Random Function) to self elect and choose if the node will become a proposer or part of the committee group.

$$VRF_{SK}(seed||role) =< Hash, \pi >$$

The SK is the node's private key, the seed is a value shared between all the nodes, in particular it's the hash of all the previous messages shared during the last block election. If this value is lower than a certain threshold, the node become part of the committee group. To ensure that all the committe are self-elected in the correct way, each node can verify the result of the VRF:

$$VerifyVRF_{PK}(Hash, \pi, seed, role) = true\ or\ false$$

The probabily that a node become part of the committee group increses with the increasing of the amount of money held by the node.

### 5.4 Safeness

Algorand is safe since the blockchain forks are very improbable, since the committee group changes at every round. Algorand is also safe against byzantine failures, this beacause if a malicious node become part of the committee group, it can:

- **Vote against the block**, this can lead the block to be rejected, but with an high probability the next round this malicious node will be not part of the committee so we can start a new round.
- **Vote for the block**, no problem since if the block is also voted from the other committees it will be chosen, else it will be rejected.

## 6 Failure Detection

### 6.1 Description

Suppose that each process in a distrbuted system holds a D module for the crash detection. So the process P can ask to D if the process Q is crashed or not: $Q \in D_P(t, \sigma)$ means that D believe that Q is no longer active at time $t$ in the run $\sigma$. We can define a two disjointed sets:

- **Crashed**$(t, \sigma)$, each process in this set is crashed at time $t$ in the round $\sigma$.

- **Up**$(t, \sigma)$, each process in this set is live at time $t$ in the round $\sigma$.

Based on these definions we can also define some properties:

- **Strong Completeness**, $\forall\sigma\forall P \in Crashed(t,\sigma)\ \forall Q \in Up(t,\sigma)\ then\ \exists t'\forall t > t' P \in D_Q(t',\sigma)$
- **Weak Completeness**, $\forall\sigma\forall P \in Crashed(t,\sigma)\ \exists Q \in Up(t,\sigma)\ then\ \exists t'\forall t > t' P \in D_Q(t',\sigma)$
- **Strong Accuracy**, $\forall\sigma\forall t\forall P, Q \in Up(t,\sigma)\ then\ P \notin D_Q(t,\sigma)$
- **Weak Accuracy**, $\forall\sigma\forall t\exists P\forall Q \in Up(t,\sigma)\ then\ P \notin D_Q(t,\sigma)$
- **Eventual Strong Accuracy**, $\forall\sigma\forall t\exists t' > t\ \forall P, Q \in Up(t,\sigma)\ then\ P \notin D_Q(t,\sigma)$
- **Eventual Weak Accuracy**, $\forall\sigma\forall t\exists t' > t\ \exists P\forall Q \in Up(t,\sigma)\ then\ P \notin D_Q(t,\sigma)$

We can define a taxonomy between accuracy and completeness:

| | Strong Acc. | Weak Acc. | Eventual Strong Acc. | Eventual Weak Acc. |
|---|---|---|---|---|
| Strong Comp. | P | $\diamond$ S | $\diamond$ P | S |
| Weak Comp. | $\theta$ | W | $\diamond\ \theta$ | $\diamond$ W |

Where:

- P = Perfect
- W = Weak
- S = Strong
- $\theta$ = Theta

If a system is P (perfect), then we are able to make Paxos live. To do this we need to elect a leader among the proposers, a simple way is to follow the rule "the process with the smallest ID alive becomes the leader", since the system is complete. In some timestamps, it can happen that there are no leader elected (the strong completeness works eventually, so in some timestamps I can believe that a process is alive even if it's dead) but we are sure that exists a timestamp $t'$ where only one leader is elected (we have strong accuracy, so if a process belives that another process is dead, we are sure that is true). However, since the system is asynchronous, Paxos is not live.

## 6.2 Building a Failure Detector

Suppose that we are in a synchronous system, we can build a failure detector like this:
$\forall Q$ ping $Q$
if $Q$ does not reply in time $2\Delta T$ then append $Q$ in $D_Q$
With $\Delta T$ the default delay. This sistem is P because:

- if a process crashes, it will not reply so it will be appended do D. **Strong Complete**
- if a process is up, it will reply in $2\Delta T$ so it's not possible that a process that is still up can be added to D. **Strong Accuracy**

If we don't know the default delay, we can use a generic delay $\Delta T$:

- if $\delta \leq \Delta T$, then the system is still P.
- if the $\delta > \Delta T$, then the system is still SC but becomes ESA. In particular, if a process Q replies after $2\Delta T$ we can update the new $\Delta T = 2\Delta T$.

If the system is asynchronous we cannot achieve any type of accuracy, but we can achieve a Strong Completeness.

# 7 Bit Torrent

## 7.1 Description

Bit Torrent is a decentralized system, used to download large sized files. Each file is splitted into blocks of 0.25MB, when a peer (user) has downloaded a block, he will share this block with other users. The server's role is to distribute blocks among the peers. In order to incerase performance, Bit Torrent allows to download in parallel more blocks from different peers. When a user wants to download a file by using Bit Torrent, he need the .torrent file which contains all the usefull informations:

- **Name**
- **Total file size**
- **Hash of each block**
- **URL of the tracker**

The tracker is a peer which knows the list of all the other peers that hold at leat one block of the file. When the user asks for the list of the peers, the tracker will reply with a random subset of that list.

## 7.2 Select the first block

There are two ways to select the block that should be downloaded first:

- **Random**
- **Rarest first**, used to ensure a more balanced block distribution among the peers and guarantee that if a peer leaves the network, the file remains available.

Typically if we download the rarest block, the process can take a long time to be completed because may the peer is faraway from us. To avoid this problem, Bit Torrent use a mix between these two techniques.

## 7.3 Endgame Problem

Typically the last block is probably the slowest to be downloaded, so we can split this block into more little blocks and download them in parallel from different peers.

## 7.4 Choking Algorithms

When we act as a peer that is sending its blocks to other peers, may we have to choke some incoming connections to avoid the overloading. We can choose amgong three different ways to choke the connections:

- **Unchoke the peer with the largest bandwidth**
- **Choke the "free riders"**, free riders are those peers that do not hold any block to share.
- **Optimistic choking**, we choke randomly among all the incoming connections.

Bit Torrent uses a mix between these three techniques because if we unchoke the peer with the largest bandwidth, may some peers with a small bandwidth will be always choked. If we choose to choke the free riders, all the peers that have not already downloaded any blocks will be always choked.

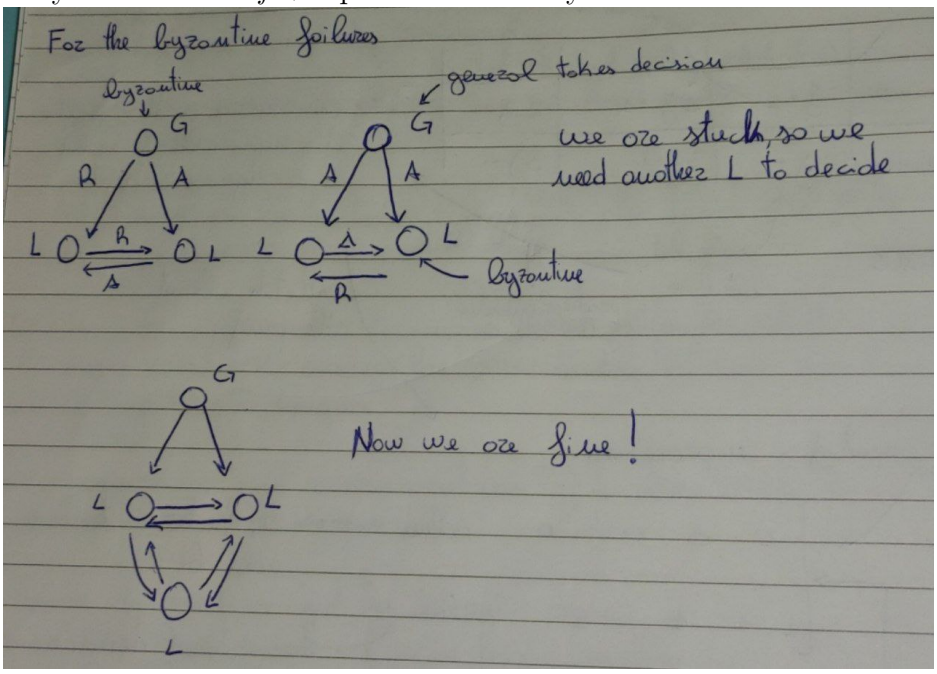
# 8 Lower bounds in consensous

## 8.1 Description

Here we will analyze how many processes do I need in order to tolerate $f$ faults.

| | Crash Failure | Byzantine Faults | Byzantine Faults with signatures |
|---|---|---|---|
| Syncronous | $f+1$ | $3f+1$ | $2f+1$ |
| Asynchronous | $2f+1$ (Paxos) | $3f+1$ | $3f+1$ (PBFT) |

In "Crash Failure" and "Byzantine Faults" synchronous systems, the system can be both safe and live.
In "Crash Failure" and "Byzantine Faults" asynchronous systems, the system can be safe but not live.
**PBFT** stands for "Practical Byzantine Failure Tolerance protocol".
Why do we need $3f+1$ process for the byzantine faults?




We can use public-private keys to sign the messages, in this way the byzantine process cannot modify the content of the message. So we need simply $2f+1$ processes.

## 8.2 FLP Theorem

Consensous with faults is impossible to solve in asynchronous systems. Sketch about the proof:

Suppose that we have $n$ processes in an asynchronous system. Suppose that these processes can vote for a binary number (so each process must choose 0 or 1). So these are possible inputs:

- 000...0 (n bits), the protocol decision must be 0
- 000...1 (n bits), the protocol decision must be 0

- ...
- 111...1 (n bits), the protocol decision must be 1

Now if we use the Hamming code to rapresent the possible inputs, we can identify two border inputs that differs from one bit on the process $i$ where:

- 000...0...0 (n bits), the protocol decision must be 0
- 000...1...0 (n bits), the protocol decision must be 1

If the process $i$ crashes or its message is very slow, the other process must agree without its vote on 0 or 1. Since in an asynchronous system we are not able to distinguish between a crash or slow message, if the processes agree on 0 and the message arrives late with value 1, the protocol has chosen the wrong value (same if the processes agree on 1 and the message arrives late with value 0).

# 9 AKAMAI

## 9.1 Description

AKAMAI is a content delivery network (like CNN) used to perform load balancing in the client-server protocol. In particular, when a client makes a request to a server that uses AKAMAI' service, the server will reply by sending the HTML document. Each multimedia inside the document, is a link to an AKAMAI server, so when the client makes a request for a specific content, it will be redirected to one of the AKAMAI server which holds the requested content. AKAMAI perform, for each websites that uses this service, an analysis to understand how many copies of each content should be done. These servers are distrbuted around the world in order to get a better worldwide coverage.

## 9.2 Why AKAMAI is usefull

Typically in the client-server protocol, if we want to perform load balancing to a specific website, we have to run multiple servers with the website. Then we should use a DNS to redirect the clients to the nearest server (or the to the less busy server). Moreover, browsers typically store the address of a website inside the cache, to bypass the whole translation process of a hostname. When this process is bypassed, also the load balancing is bypassed. With AKAMAI we need only one server which runs the website and AKAMAI will use its own DNS to perform the load balancing.

# 10 GDPR

## 10.1 Description

The GDPR is a law created by the European Union that must be followed by all the companies inside the EU. It defines the rights of the users on the internet, in particular:

- **Art. 15: Right of access by the data subject**, the data subject (user) must be able to obtain from the controller all the data concerning him and, where is that case, access to the personal data.
- **Art. 20: Right to data portability**, the data subject have the right to receive the personal data concerning him in a structured and commonly readable format.
- **Art. 12: Time to send the data**, the controller must must provide informations within one month.

There are several problems with this law, after few experiments the researchers has found some websites in wich:

- The GDPR was not followed.
- User rights were not exposed in clear.
- There were no contacts for the DPO (*Data Protection Officer* is a person who cares about the users rights inside a company).

After that, they also have sent the request for obtain the personal data. This is typically done by sending an email or by fulling an online form. Some of the companies did not send any reply or were having techincal issues related to the request. The companies that have replied in some cases have replied with an identification request (for example with an ID document, knowledgeable questions, email or phone call), but in other cases no identification was required (this is very dangerous!). However, some of the companies has accepted an obfuscated document or a different email, so the identification was not secure. Most of the companies that have replied, did it in time. One of the problem is the file format used to send data, there is no standard format. Moreover, most of them have sent the data as a plain text or by encrypting them but putting the password in clear inside of an email. This is very dangerous since out data can be stolen very easily in this case. Only a few companies have sent the password by using a different channel.