Blockchain Application for Electronic Voting

JAVIER MANTILLA
BSc. (Honours) in Software Develpoment

April 26, 2019
Final Year Project
Advised by: Daniel Cregg
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology

Contents

Table of Contents

Introduction **4**

Methodology **6**
- **2.1 Principle of voting systems** **6**
- **2.2 Electronic voting systems** **8**
- **2.3 Solutions of electronic voting systems** **8**
  - 2.3.1 Encryption by public-key cryptography 9
  - 2.3.2 Homomorphic encryption 10
  - 2.3.3 Vote Mixing Systems 13
  - 2.4 Examples of voting systems 15
  - 2.5 Conclusions 15

Technology Review **17**
- **3.1 Bitcoin: P2P decentralized currency** **18**
- **3.2 Users - Addresses** **21**
- **3.3 Transactions** **23**
- **3.4 The chain of blocks (Blockchain)** **25**
- **3.5 Proof of work** **27**
- **3.6 Consensus** **29**
- **3.7 Other characteristics of blockchain implementation** **33**
  - 3.7.1 UTXO (Unspent Transaction Output) 34
  - 3.7.2 Merkle Chain 34
  - 3.7.3 Rewards: mining and commissions 36
- **3.8 Disadvantages, weaknesses and attacks on Bitcoin** **37**
  - 3.8.1 High latency 37
  - 3.8.2 Bitcoin software 38
  - 3.8.3 Double expenditure attacks [17] 38
  - 3.8.4 Scalability 39
- **3.9 Conclusions of the blockchain technology analysis** **40**

System Design **43**
- **4.1 Architecture** **43**
- **4.2 Setting up the Environment** **44**
- **4.3 Simple Voting Contract** **46**
- **4.4 Interaction with the Contract in the Nodejs Console** **50**
- **4.4 GUI Interface to Blockchain and Vote** **51**

System Evaluation **55**
- **5.1 Evaluation** **55**
- **5.2 Limitations & Opportunities** **56**

Conclusions **58**

Appendices **60**
- **Github** **60**

About this project

Abstract Governments, for the sake of innovation, transparency and citizen participation are interested in pilot voting projects based on blockchain technology. This innovative technology allows the decentralization of the government of operations, specifically the voting of citizens in an immutable, auditable, safe and reliable distributed registry. Throughout this study we will see the problems detected during its deployment phase, the solutions adopted and the conclusions for its use in institutions. The final result will be achieved with the elaboration of a proof of concept prototype.

Author Javier Mantilla

Chapter 1

Introduction

In recent years, the emergence of Bitcoin (the virtual currency created by Satoshi Nakamoto), has been a revolution not only in the economic field, in which it has had remarkable success, establishing itself as a "currency" alternative and generates even the emergence of other virtual currencies or cryptocurrencies based on the same concept (Ethereum. Litecoin, Dogecoin, PeerCoin, etc.), but also in the technological field where its blockchain protocol has been established as the reference of the concept of distributed registration.

The success of Bitcoin, and therefore of blockchain, as well as its testing during these years validate the concept of registration or distributed consensus, which allows registering efficiently and securely transactions virtual elements. Bitcoin was launched in 2009 in an environment as sensitive as the economic one, where it has been possible to verify its safety and robustness.

And this concept, which after all is solving a fundamental problem of the digital environment, copy and ownership, can be applied to other areas apart from the economic or cryptocurrencies. For example, digital art, rights of author, or electronic voting.

The purpose of this project is to study the application of technology blockchain to build an electronic voting system. One that offers higher security and more advantages than the current ones due to the security-based nature in cryptography and decentralization that this technology provides.

First, the electronic voting scenario will be defined, detailing all the needs and requirements that a solution of this type demands, taking as an example the current electronic voting systems already implemented, They mainly use technologies based on centralized systems. Next we will analyze the blockchain technology, its characteristics and main elements, taking as an example its implementation in cryptocurrencies (mainly Bitcoin and Ethereum).

From these two analyses, an electronic voting solution will be proposed, based on blockchain technology, which responds to voting requirements and could represent an alternative over current systems based on centralized storage and

transactions technologies. A detailed description of the solution will be made, as well of the implementation, deployment, and technologies and algorithms that would be used.

Once the solution is described, an analysis will be carried out, observing the weaknesses and disadvantages of the system, making a special emphasis comparing it to current electronic voting solutions. Finally, the conclusions of all this work, the proposed electronic voting system, its conditions and its suitability will be presented. Demonstrating the use of blockchain protocol to solve this type of problems.

Chapter 2

Methodology

Most current political systems (and some non-political ones) base their internal organization, and to a greater or lesser extent, its decision-making, on voting systems.

A vote is essentially a consultation of a number of people (voters) to choose between different options. Although rules may apply different in the voting, the spirit is usually the same, it takes out the option with more supports.

In fact there is a whole field of study of voting systems. The theory of voting, which encompasses subfields of political theory and mathematics. Although there are many types and voting systems (for example, just raising hands), this analysis will focus on the votes associated with political systems (elections and citizen participation), since it is one of the most complex (high number of geographically dispersed participants, need for privacy in the vote, need for voters identification, etc.) and the applied solutions to this type of voting would solve other simpler cases.

2.1 Principle of voting systems

The voting systems that are being analyzed (elections and citizen participation) have a series of characteristics and requirements that must be fulfilled. These characteristics and requirements are prior, universal, and independent of the technology and procedures that are going to be used to voting.

The following are the most important requirements for voting:
1. **Single vote:** It can only be exercised by the one person who has the right, and it can be exercised effectively once during the vote.
2. **Private vote:** The vote is secret. No one should be able to find out what a specific elector has voted.
3. **Integrity of the votes:** Once issued, no one should be able to change the meaning of a vote.
4. **Integrity of voting:** It should not be possible to modify the overall result of the voting event.

5. **Individual audit:** Ensure that an individual vote is counted correctly.
6. **Universal audit:** Ensure that the process and electoral counting is done correctly

These requirements are met in most democratic system choices in a similar way. Here is how they are met in most common electoral systems:

1. **Single vote**: This requirement is met thanks to the census, national identification document and a polling stations system.
    a. By having a census it is easy to limit the people who can participate (voters that meet the requirements of age, territory, etc.)
    b. The polling system obliges the voter to cast his/her vote in a specific place (an urn of an electoral college). This way, vote control is facilitated, since the members of the polling station have the list of people who can vote, the people who have voted and do not allow double voting.
        i. In other countries where they do not have these census systems and schools, the use other systems, like marking the voter with indelible ink, are applied.
    c. By having a unique identifier per citizen, it is easy to ensure the identity of the voter and avoid impersonations.

2. **Private vote**: This requirement is fulfilled thanks to:
    a. The voters cast their ballot on paper within an enclosed envelope. This way, the members of the voting jury and others stakeholders cannot know the contents of the envelope and therefore the casted vote.
    b. These envelopes are put in an urn, depositing all the envelopes. Once the urn is filled with envelopes it is impossible to distinguish what envelope belongs to each person who has gone to vote. Sometimes, this system has problems in constituencies where very few people vote, and it is possible to infer, although not assure, to whom each vote belongs.

3. **Integrity of the votes and individual audit**: It is ensured through the polling station systems, composed of several selected people through a draw, party proxies and controllers to supervise the process and ensure that votes of an urn are not manipulated and comply with established rules.

4 **Integrity of voting and universal auditing**: The system of counting and redundancy ensure:
a. Each table recounts the votes, annotates and transmits the results electronically to the central electoral board.

b. In addition, the votes and the paper count are deposited in custody in case it is necessary to make a later count.
c. The system of agents and controllers allows parties have access to the tally, and therefore any party or interested person could do his own recount to check that matches the official.
d. The whole system and process is public and transparent and can be supervised both by participating parties and by transparency associations, etc.

2.2 Electronic voting systems

By contrast to the traditional voting system, based on ballots and boxes, there has been introducing and testing new proposals applying new technologies. It can be defined as electronic voting the application of ICT to a process of vote.

In general, there are two types of electronic voting:

- **Onsite electronic voting**: these systems are continuities of the traditional voting system, in which each person must appear physically at a polling station. Technology is introduced into ballot instruments and ballot boxes (electronic or magnetic-band based ballots), electronic menus to select the candidate, etc.) Generally these electronic voting systems use technology to streamline the count and prevent access and manipulation of votes by unauthorized persons.

- **Online electronic voting**: in this case, voters can vote from their location, without having to travel to a specific position, using electronic means (Internet, sms, etc.) This analysis will focus on electronic voting by Internet (or telematic vote), since it would be the type of technological application that would offer a greater number of advantages over the face-to-face system. Therefore in the following paragraphs, when referring to electronic voting, it is talking about the electronic vote by Internet or telematic vote.

2.3 Solutions of electronic voting systems

Electronic voting systems must be able to offer the same guarantees that traditional voting systems, and the use of technology and computer systems should help to offer even more guarantees, such as for example, verification by the voter that his/hers vote has been posted correctly [1].

In general, telematic voting systems follow a similar sequence of the traditional vote:

1. **Registration / identification**: The system must identify all possible voters. Sometimes, depending on the system, prior registration it is not necessary. Nevertheless, it is necessary to specify the procedure of voter identification (e.g. digital certificate of the national ID card)
2. **Preparation of the vote**: The vote conditions must be specified: candidates, vote selections, dates, etc. Depending on the type of voting the voter must receive the codes corresponding to each candidate. Also he/she must receive the voting rules so that the vote is valid.
3. **Sending of the vote**: The voter makes his/her vote, prepares it and sends it to the voting system. At this point the telematic voting system must use algorithms that allow later vote verification preserving the voter's anonymity.
4. **Vote count**: The system must manage all the votes received, accounting and offering the final result.
5. **Verification of votes**: if possible, the system should allow voters verify his/her vote has been correctly posted (maintaining voter anonymity)

To achieve a telematic voting system with all the guarantees of security, different techniques and cryptographic algorithms are used. Below are some of the most used and have been considered most relevant for this study.

2.3.1 Encryption by public-key cryptography

It is the basis of most current cryptography and is based on the existence of a key pair (a public key and a private key). The characteristic main purpose of this encryption is that any message encrypted with the private key it can only be deciphered using the public key. Usually the users of this system publicize and make public their public key so that any interested party can send them an encrypted message with the absolute certainty that only he/she can decipher it.

These cryptographic solutions are very common and are used in most computer security systems: encryption of messages and emails, digital signature, communication protocols and identification between servers, etc.

In the context of telematic voting systems, the public key cryptography is used to encrypt the vote, so that only the holder of the private key can access it and post it:
1. In the preliminary phase of vote preparation, the key pair is generated, public and private.
2. The public key is distributed to all voters so that they encrypt their vote with it.

3. The private key is kept by the board or electoral committee. Generally this key is shredded and distributed among the members, so that a single member cannot decipher the votes.
4. Once the voting is over, the members of the electoral board or committee introduce the private key to decipher the votes and do the count.

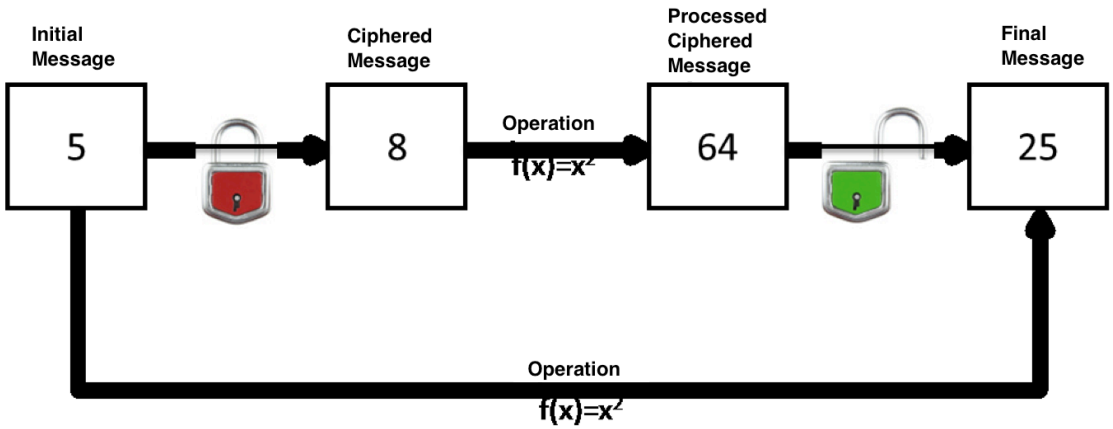
This way it is ensured that only with the connivance of the members of the board or electoral committee can access the voting data.

2.3.2 Homomorphic encryption

Homomorphic ciphers are a specific type of cryptographic algorithms. They have a very interesting property:
- Assuming a function $E(x, k)$ that is the algorithm that encrypts a message x using a key k.
- Assuming a function $D(x, k)$ which is the algorithm that deciphers a message x using a key k.
- If the encryption system is correct, it follows that:
$D(E(x, k), k) = x$

That is, if the message is encrypted with the function E, it can be deciphered with the D function and get the message back in plaintext.
- If the encryption is also homomorphic, it is true that:
$D(f(E(x, k)), k) = f'(x)$
That is, if an (f) operation is applied to the encrypted message the result deciphering is equivalent to having applied the plaintext message in a (f') operation.

Sometimes, depending on the system, these operations (f) and (f') they can be the same operation, although this is not always the case. The next schema represents a simple example where $f(x) = f'(x) = x^2$



This property of homomorphic cipher systems is extremely interesting, since it allows to operate safely in encrypted systems without knowing the key and could mitigate some considerations of security on the current trend of some companies and individuals of maintain private data in the cloud (i.e. in possession of other companies and / or countries).

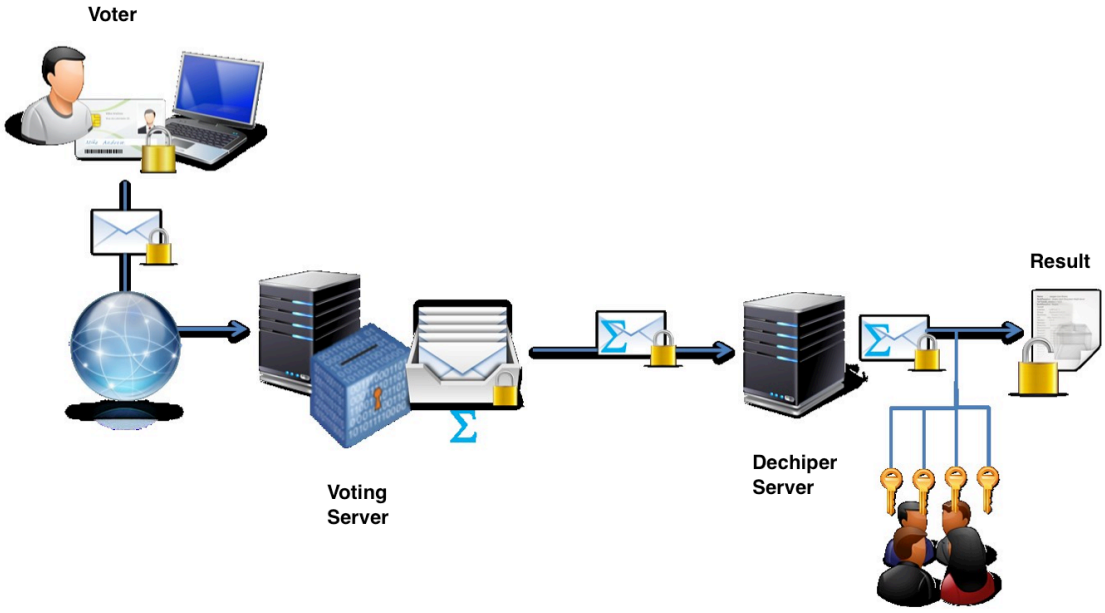
A clear example would be a company that has an encrypted database in an external provider. In that database there would be a "clients" table. If the the company needs a list of all its clients in Barcelona, it should do:
1. Download the table from the external provider to a local system
2. Decrypt the table
3. Conduct the consultation

If the database is encrypted with a homomorphic system that allows string text operations, the user should do:
1. Encrypt the keyword to search: `Galway` with the same password as the database is encrypted.
For the purposes of the example, it will be assumed that the result of the encryption would be "yawlaG".
2. Would perform directly against the database in the provider external query:
```
SELECT * FROM client WHERE locality = 'yawlaG';
```
3. Decrypt the result of the query.

This way, the database is in an external provider and use its computer services safely.

Currently there are not implemented complete homomorphic encryption systems [2] that can perform complex operations in a practical way like those in the example, but some systems allow, with certain restrictions, specific simple operations (called partial homomorphic systems):
- ElGamal [3]: In this algorithm, the encryption *(E)* of a message *(m)*
$E(x_1) \cdot E(x_2) = E(x_1 \cdot x_2)$
- Goldwasser-Micali [4]: In this algorithm, the encryption *(E)* of a bit *(b)*
$E(b1) \cdot E(b2) = E(b1 \oplus b2)$ where $\oplus$ is the XOR operator.
- Benaloh [5]: In this algorithm, the encryption *(E)* of a message *(m)* with a block size *(c)*
$E(x1) \cdot E(x2) = (x_1 + x_2 \bmod c)$
Other possible partial homomorphic systems are: Paillier, Okamoto-Uchiyama, Damgård-Jurik, etc.

These systems can be enormously useful in telematic voting, since they would serve to be able to count the total result of votes without having to decipher each vote individually [6].
For example:
1- An intermediate system is generated (virtual urn) that does not have the deciphering key that the board or electoral committee will use.
2- Voters send their encrypted vote to the virtual urn.
3- Once the virtual urn has all the votes (encrypted), it applies an addition operation (without deciphering the votes).
4- The result of the sum operation is sent to the central system.
5- In the central system, the board or electoral committee uses the private key to decipher the message that the virtual urn sends, and contains the aggregate sum result.
This way the central board does not have access to individual votes, while the middle urn does not have access to the content of each vote.



To implement such system, the voting message is constructed as a vector. This vector has as many positions as candidates, and the voter uses a 1/0 binary system to indicate their vote:

Vote Format	Candidate 1	Candidate 2	Candidate 3
Voter 1	1	0	0
Voter 2	0	1	0
Voter 3	1	0	0
Voter 4	0	0	1
Voter 5	1	0	0
Once votes are summed:			
Final result	3	1	1

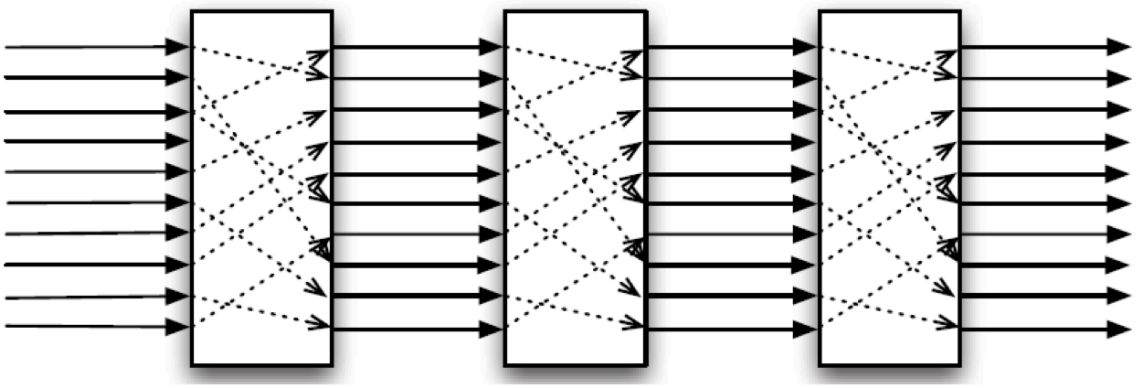
In this example, candidate 1 wins.

Keep in mind that this type of solution only works for a certain types of voting, which could be denominated simple voting, in which only some options must be chosen or can be resolved by simple operations such as the direct sum of the votes. Other types of more complex votes could not be resolved by this solution.

2.3.3 Vote Mixing Systems

It is based on the concept of mix-net invented by Chaum [7]. These techniques are used to break the correlation between the inputs to a system and its output, in this case between the casted votes and the counted votes.

Basically it is a series of chained servers (mix-net) that perform an encrypted permutation of the votes. This way on the final server, the order of the counted votes is not related to the order of the casted votes making it impossible to associate a vote to a user by means of techniques of monitoring (use of logs, channel monitoring, association of ips, etc.)



The operation (simplified) would be:
- Given a number of *n* voters, *n* votes are created *(V)*.
- Each voter votes and sends it. The encrypted vote is denominated *(C)*.
- The set of votes is created on the server that receives the votes
$C_{1,0}$, $C_{2,0}$, $C_{3,0}$, ..., $C_{n,0}$
(where the first suffix indicates the order in which the vote came, and the second suffix indicates the number of servers which it has passed. In this case, 0 indicates that it is the initial server)

To ensure anonymity, the system has separated the digital signature of the votes from the content of the votes.

- The system consists of *t* mixing servers S_1, S_2, S_3, ..., S_t
- On each server S_i a mixture is applied on the set of input votes:

$$\{C_{1,i-1}, C_{2,i-1}, C_{3,i-1}, ..., C_{n,i-1},\}$$

The mix consists of an order permutation on the above set using a secret algorithm and performing a sequence of re-encryption (or decryption) on the above set, to prevent linking entry data with the exit data.

$$\{C_{1,i}, C_{2,i}, C_{3,i}, ..., C_{n,i},\}$$

- Once the process of mixing votes is completed, they can deciphered (if they haven't been) and publish the results.

The use of these systems requires adding guarantees that the processes of mixed do not change the direction of the vote. Generally servers that perform the mixing they also perform a series of tests to check the consistency between input and output. Those checks are stored for audits later. Although it is true that these checks and audits could reduce the security of the system, an attacker should be able to access all the checks of all the mixing servers and all the input/output data from them to be able to relate an entry vote with the final vote.

There are different approaches to the implementation of these systems called mixing nets [8] [9]. Although their basic operation is the same, they have differences in the type of mixing protocols, application encryption and decryption, or generation of proof of work to validate the integrity of operations.

2.4 Examples of voting systems

Different telematic voting systems already exist. They use the previous or other technological solutions that have been used for real scenarios.
Some examples:
- ElectionBuddy. An online voting system (http://electionbuddy.com) that allows configuration and management for voting and elections that are totally on-line.
- Helios voting: An organization that has designed a verifiable system for online voting (https://vote.heliosvoting.org)

2.5 Conclusions

Work is being done to improve democratic capacities for several years through technology. Telematic voting systems could ensure equal or greater guarantee than presential voting systems, and also offer advantages in their condition as systems based on the use of new technologies, some of which are:
- **Cost**: The current elections have a very high cost due to all the material, logistics and personnel needed. A telematic voting system, once implemented, it would have a very low cost in each use.
- **Ubiquity**: Current systems require or physical displacement of the voter, or a series of cumbersome bureaucratic procedures that in occasions discourage or do not allow some people to vote. A telematic voting system would eliminate these problems.
-**Verification of the vote**: Currently once the voter casts the vote in the ballot box has no way to check it has been posted correctly. It must rely on the ballot verification and counting system. The cryptographic solutions of telematic vote systems can safely and anonymously verify each vote individually.

Currently there are solutions to build voting systems that apply the various technologies. Those technologies have been analyzed to guarantee the security and anonymity of the vote. Sometimes these solutions are combined (encryption + homomorphism + mix-net) to give more robustness to the system. However, these solutions suffer from a problem fundamental: trust

In the end the voting is processed on one or more servers with software and keys from the voting. This centralization of the process can generate mistrust, as for the possibility of an attack on any of the voting servers that can alter the voting, as by the ease of the voting manager to alter it in own benefit, despite the security and audit systems.

In addition, it is necessary to take into account the difficulty of people without technical knowledge in understanding these systems and therefore their distrust.

Chapter 3

Technology Review

The objective of the blockchain protocol is to achieve a decentralized system, that is, composed of several independent nodes following the peer to peer scheme (P2P), which manages a single record where all the transactions and operations that are carried out (in the financial field it would be the equivalent to the accounting ledger where all the movements, payments, collections, etc.).

The following sections detail the operation of the protocol and its main features:

• **Peer to peer (P2P) decentralization**. It is a peer network where all the nodes that make up the network behave as equal to each other, acting at the same time as servers and clients of the rest of nodes in the network.

• **Anonymity**. The protocol is used to make transactions between addresses (equivalent to what would be a bank account). But the users themselves generate these addresses, so there is no centralized registry that allows assigning an address to a concrete person.

• **The blockchain.** It is the main foundation of the system. It represents the record where all transactions are logged. It has all the information of all the elements and all the transactions that have been carried out since the beginning of the system.

• **Transactions**. It is an operation that we want to record in the system. For example, the transfer of an element (i.e. a Bitcoin) from one user to another. A transaction once is considered valid, it is included in the blockchain.

• **Distributed consensus.** It is the logic that makes the system work. In a distributed network, different nodes can be doing simultaneous operations that should be spread out through the network. Sometimes the nodes collide with each other, a well-defined system is established for determine which transactions are valid and are incorporated into the chain and how conflicts and collisions are resolved.

• **Cryptography.** The cryptographic algorithms are responsible for the security of the system as well as its robustness and reliability against malicious attacks as to compromised situations (double spending, duplicities, etc.). The blockchain protocol is based on:

- Digital signature.
- Hashes and Merkle Trees.

- Proof of work.

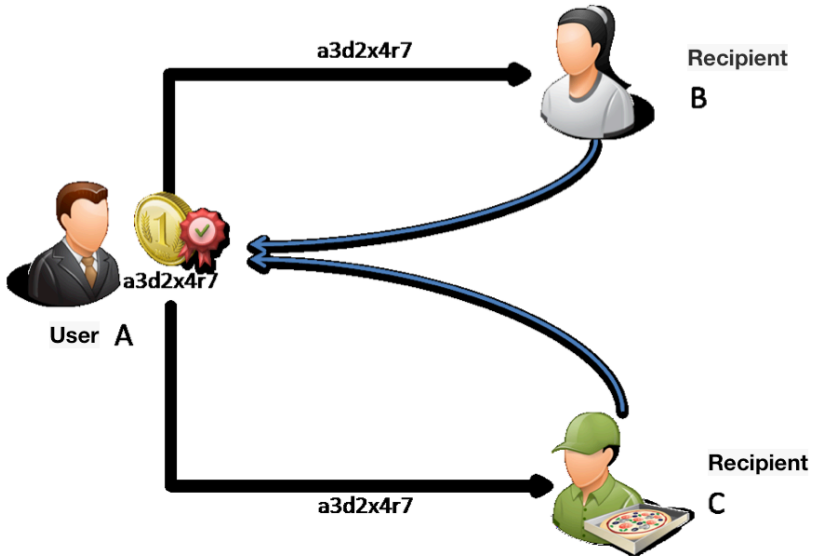
3.1 Bitcoin: P2P decentralized currency

Keep in mind that blockchain was born as a protocol to give support for a specific solution, bitcoin, which aims to solve a concrete objective, the creation of a currency and transaction system for such currency among several users without depending on a centralized entity.

In a digital environment it is difficult to assign value to a digital object. In the traditional economy, the value of an object is determined in a large extent by its shortage (law of supply and demand). But a digital object is not more (in its last instance) than a numerical sequence. Therefore a digital object can replicate as many times as needed at a ridiculous cost. This property subtracts value (in the traditional sense) from digital objects. Copies of software, songs etc., are a clear example of this problem, and DMR systems are examples of attempts to "make expensive" the copy system and maintaining the shortage of the product.

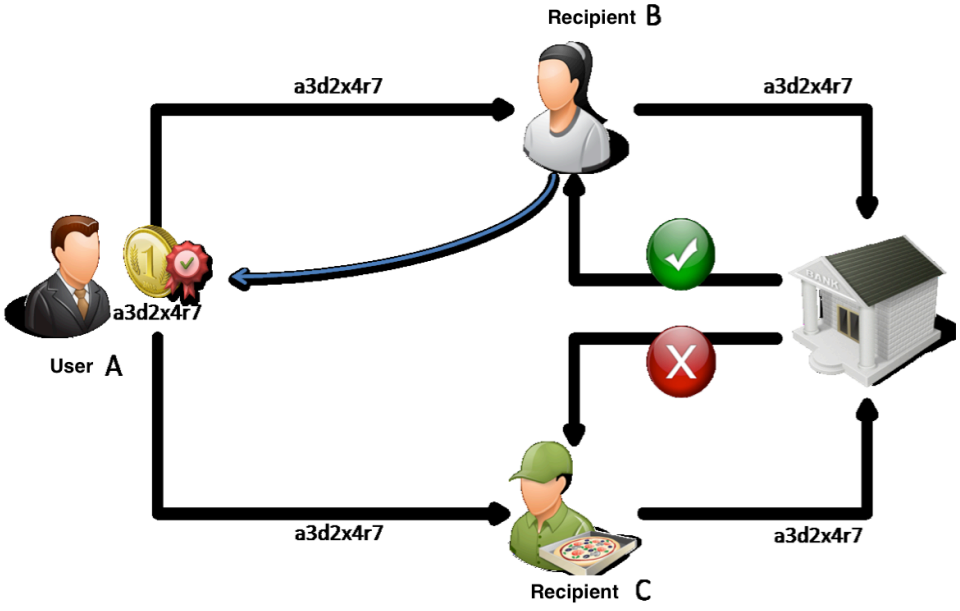
The case of coins is paradigmatic, since they are by definition scarce elements. The main use of the coins is the exchange: an amount of coins in exchange for a consideration (a service, an object, etc.). Therefore its main use is to make transactions: send units monetary from one user to another.

A coin in a digital environment must face the problem of double expenditure. Each monetary unit can have a unique code (010010110 ..... 001), and even a digital signature of an issuing entity, but it will be very easy to copy and use it two times.
1. User A makes a transaction to a recipient B by sending him a monetary unit in exchange for a service.
2. User A makes another transaction to a recipient C by sending him a copy of that same monetary unit in exchange for another service.
3. As the monetary unit is authentic and the two copies are identical, the recipients have no way of knowing what unit is being used more than once, thus user A receives two services.



The traditional solution to this problem is to use a centralized entity that controls transactions. In general, this is what currently happens with banks.

A bank would be a centralized entity, which keeps a record of the monetary units its users possess as well as the transactions that they perform. This way the bank would detect a double-spending attempt if a user tries to use money that he/she has already used and would not allow it.



1. User A makes a transaction to a recipient B by sending him a monetary unit in exchange for a service.
2. Recipient B confirms the transaction with the bank.
3. The bank, which has register of all currencies and transactions, validates that the transaction can be made.
4. Recipient B, upon receipt of the bank's confirmation, accepts the payment and gives the service to the user A.
5. User A makes another transaction to a recipient C sending him an exact copy of the same monetary unit in exchange for another service.
6. Recipient C confirms the transaction with the bank.
7. The bank rejects the transaction, since that currency has already been used previously.
8. Recipient C does not accept the payment and does not provide the service to the user A.

This solution has some drawbacks, the most obvious being:
- **Security**. Being a centralized architecture, if an attack occurs on the central node (the bank) the whole system is at risk.
- **Privacy.** Trust is deposited in the centralized entity. The users must register with that entity to operate with it, and therefore the centralized entity is aware of the funds and transactions of the users.
- **Dependency**. The user is totally dependent on the centralized entity, exposed to any change that it wants to be made (i.e. devaluation of the currency, freezing of accounts, etc.).

As a solution, based on the p2p [10] Bittorrent protocols, a decentralized database could be built. This database allows each transaction in multiple nodes. This option solves the previously exposed problems:
- The fall of a node does not have a great impact on the system that would keep working.
- Trust is not deposited in a centralized entity; it is deposited in the protocol, which must guarantee the proper functioning of the distributed system. It is not deposited in the individual nodes.
- There is no dependency, since a node cannot decide unilaterally changes in the protocol.

Even so, a decentralized database of this type also entails problems to consider:
- **Synchronization and latency**: Not all nodes will have the same information at the same time. It must take into account the propagation time of the information over the nodes.

- **Double expense**: A user could send two spending transactions using the same monetary unit at two nodes distant from each other, with the hope that these nodes validate the transaction before the information about that monetary unit has been used in another node.
- **Consensus**: When updating information from different nodes, cases of contradictory transactions (double spending) can occur. Who and under what criteria decisions about it should be made?

All these drawbacks (and some additional) are what the blockchain protocol solves, allowing the use of Bitcoin as cryptocurrency.

3.2 Users - Addresses

To solve the problem of users, the protocol that supports bitcoin it discards the user registry totally. Instead, it uses an address system based on public key cryptography or asymmetric cryptography [11].

Asymmetric cryptography is based on the generation of a pair of keys for each user, a key called public that the user can freely distribute, and a private key that only the user should know. To send a secure message to the user any other user can encrypt the message with the public key, so that only the holder of the private key (the initial user) can decipher it and read it.
Public key cryptography algorithms are based on difficult mathematical problems. These problems are used to generate the two related keys without making possible to infer one key from the other.

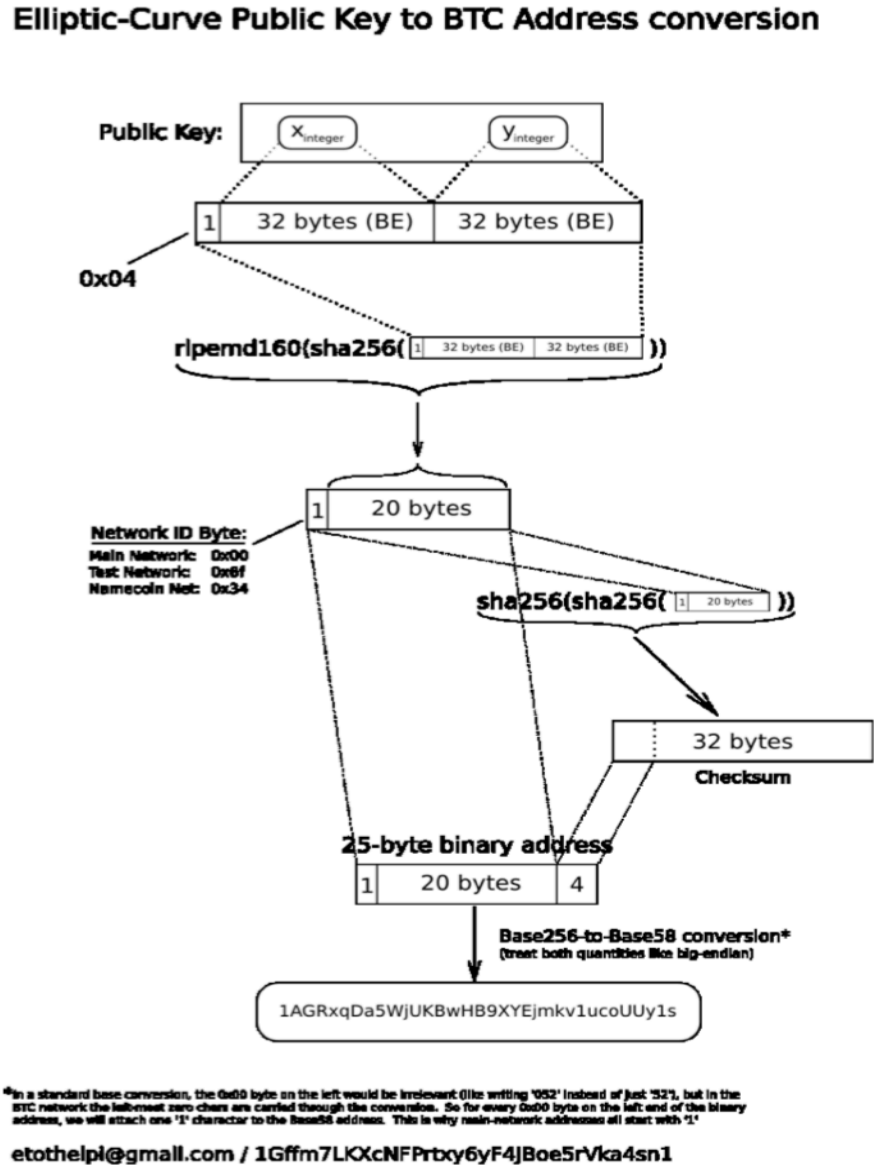
Specifically for Bitcoin, generating user keys the algorithm ECDSA [12] based on elliptic curve cryptography [13] is used.

Basically, any user who wants to use Bitcoin, using this cryptographic algorithm generates a pair of asymmetric keys. The user who wants receive a payment can send to the payer his/hers public key. The payer links the payment to this public key, so that only the user who has the private key can access the payment and therefore those funds.

Although Bitcoin allows payments to a public key, such public keys are long (ECDSA public keys have 64 bytes) and complex so it is easy to make mistakes when transcribing them. For this reason, the Bitcoin protocol usually uses payments to addresses instead of direct payments to a public key.

An address is nothing more than a summary of the public key. Applying various algorithms that transform the 64-byte public key into a shorter 25-byte address. That address incorporates some control codes and error correction mechanisms [14].

This model is completely open, so any user can build autonomously all the addresses he/she needs. Also being a process performed by the user, independent from the network, and without the need for a registration or centralized authorization, there is no way to link an address to the holder.



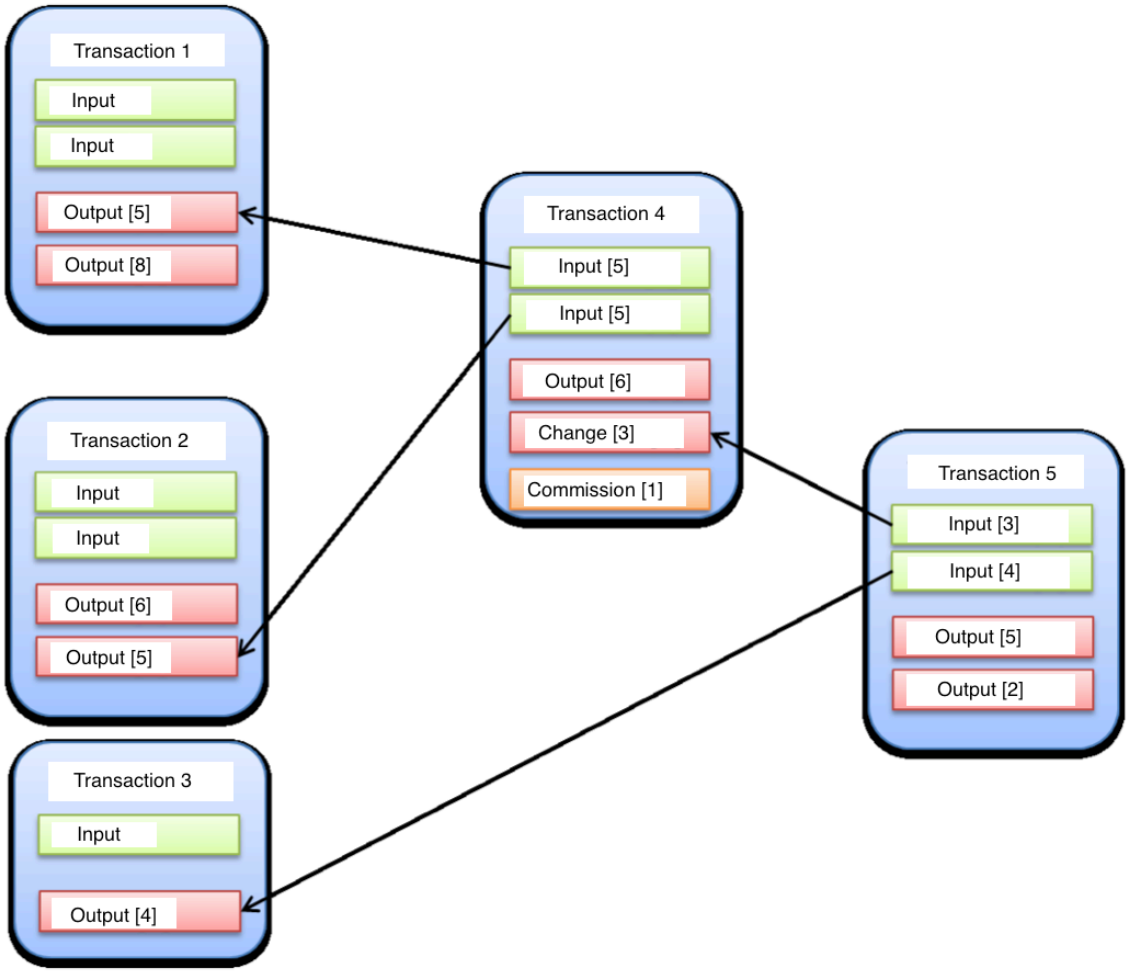
3.3 Transactions

Transactions are the central element of the system. The ultimate goal is that transactions of monetary units can be carried out safely between users.

In this protocol, a transaction is composed of two main elements:
- Transaction outputs. It is the transfer of funds. Indicates the amount of funds to be transferred and the recipient of the funds.
- Transaction inputs. They are the funds that will be used for the transaction. These funds must come from a previous transaction. Therefore, the inputs of a transaction are references to outputs of previous transactions.

In addition, the transactions have the following characteristics:
- They can combine different input and output transactions. That is to say, a single transaction can take funds from different previous transactions and distribute them in a way the user deems appropriate to different recipients.
- The sum of input amounts must be greater than or equal to the sum of output amounts. In most the cases, a user will not have previous transactions whose funds sum exactly the amount he/she wants to send in the transaction. What is normally done is to take previous inputs transactions whose amount is greater than necessary, and in the output transaction place the desired funds to transfer to the recipient. Plus another output with the remaining amount (the change) in which recipient is the user who is making the transaction.
- Commission. Any user, besides the outputs recipients and him/her self (the change), may leave a pending amount. This surplus is called commission and is totally voluntary (since the user who creates the transaction can establish the inputs and outputs as he/she prefers). This commission is for the node that process the transaction; an incentive for the different nodes that dedicate resources to process user transactions.

The following diagram exemplifies the transaction operation:



In this example it can be observed:
- Some initial transactions (transaction 1, transaction 2 and transaction 3)
- Transaction 4, in which the user has to pay 6 Bitcoins. For that the transaction:
    - Collects 5 Bitcoins from transaction 1.
    - Collects another 5 Bitcoins from transaction 2.
    - Makes the output transaction (payment) of 6 Bitcoins.
    - Makes another output transaction to himself (the change) of 3 Bitcoins.
    - The difference between the inputs (5 + 5 = 10 Bitcoins) and the outputs (6 + 3 = 9 Bitcoins) leaves a commission of 1 Bitcoin.
- Transaction 5 in which the user must make two payments, of 5 and 2 Bitcoins. For this the transaction:
    - Collects 3 Bitcoins of change of transaction 4.
    - Collects 4 Bitcoins from transaction 3.
    - Makes an output transaction (payment) of 5 Bitcoins.

- Makes an output transaction (payment) of 2 Bitcoins.
- In this case all inputs (4 + 3 = 7 Bitcoins) become output payments (5 + 2 = 7 bitcoin) so there is no commission left.

To make a transaction, it is necessary that the user uses funds which he/she has rights, that is to say, has the private key of the transaction input address.

As the source transaction is sent to an address, in order to dispose of the funds, the user must perform a digital signature process with the private key. This process ensures that the recipient is the legitimate owner of the address to which the funds were sent and ensures that funds are available to that address.

In reality, the system is more complex, since the funds verification is not a simple signature, but includes the execution of a script that is generated by unifying a resident code in the source transaction that has the public key, and a code generated by the recipient user with the private key. The script execution must give a concrete result to be considered valid. For the purposes of this document it is not considered necessary to further detail of this system; simply know that:
- It is a secure system that ensures that only possessing the private key of an address allows use of the funds that have been sent to that address.
- Being a scripted system, it can be used to allow different types of recipients. The most usual (about 99% of transactions) is the payment to a bitcoin "address", but there are also scripts that allow payments directly to a public key (not to the address generated from it), multifirm payments (which require authorization of several private keys), etc.

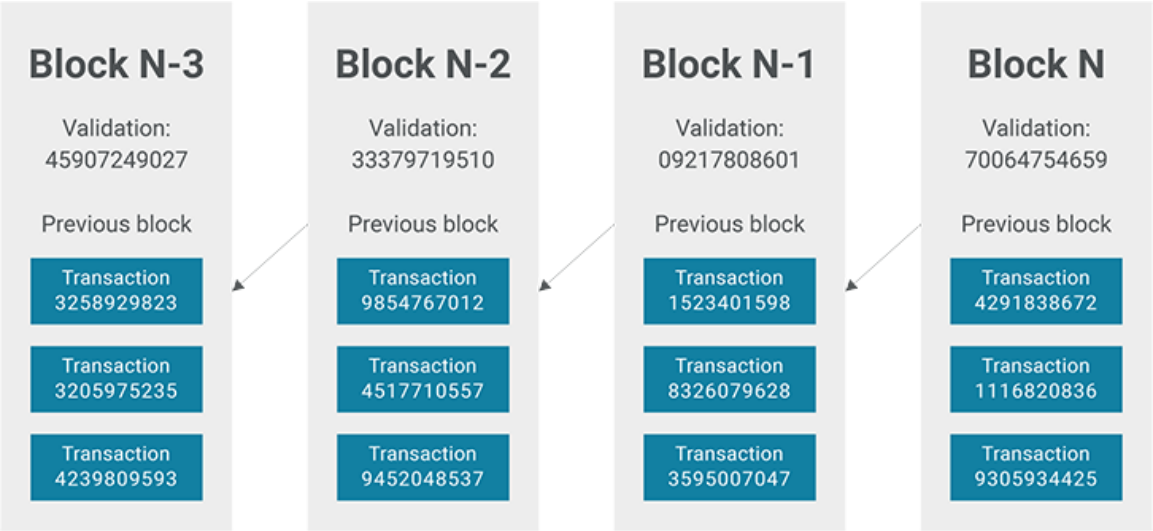
3.4 The chain of blocks (Blockchain)

Blockchain is the pillar of Bitcoin. Once an address system is established and transactions between addresses; a distributed p2p environment must solve the fundamental problem: how is it distributed among nodes the information of the transactions so that all the nodes have the same information and there are no problems of concurrency, security, double expense, etc.

This problem is solved by blockchain, and that's what it makes it possible for the currency to work. The blockchain is, in essence, a distributed database, which stores all transactions that have been made since the beginning of the system (2009).

As its name suggests it is a chain of blocks. Each block includes one series of transactions and the reference to the previous block of the chain. And one transaction is not considered valid unless it is part of the blockchain. This way the blockchain contains all the transactions considered valid.

The following figure illustrates the concept of transaction blockchain (with references between one block and the previous).



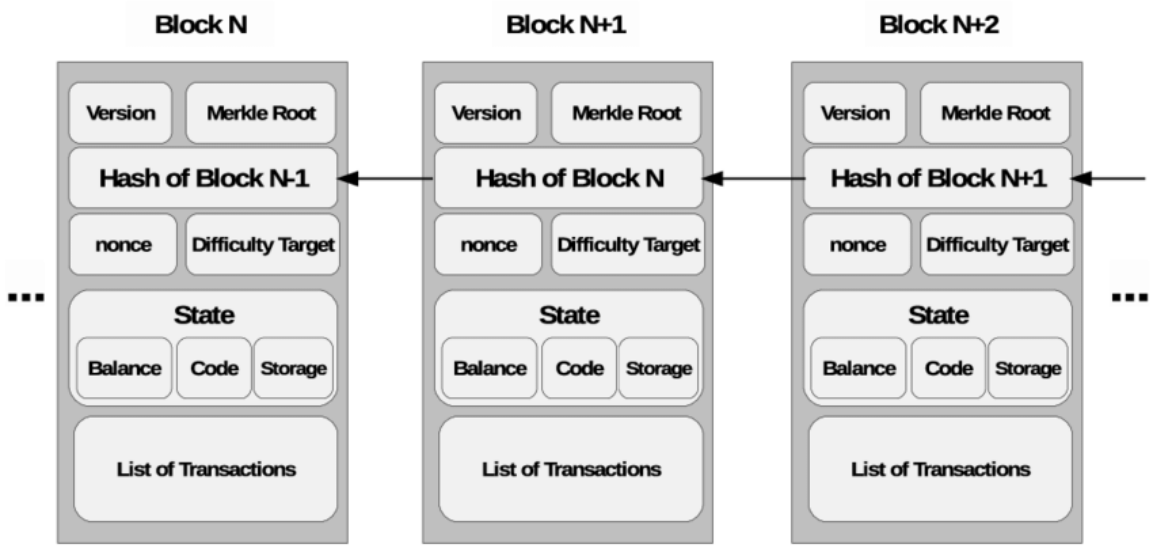
Therefore, the operation would be the following:
1. A user prepares a transaction with some inputs and outputs. With his/her private key he/she certifies the inputs and specify in the outputs the address of the recipient.
2. The user sends the transaction to a Bitcoin node to validate it.
3. The node validates the transaction: check that it is correct, that in the blockchain has not already included any previous transaction that uses the same funds, etc.
4. The node groups several transactions that have been sent to it and with them creates a new block.
5. The node adds the new block to the blockchain.
6. The node distributes the new string to other Bitcoin nodes so that they consider the new block.

This scheme of operation depends entirely on the honesty of the different nodes and their ability to agree on what is at any time the valid block, since different nodes can be incorporating to the same chain different blocks with different transactions. How is this distributed consensus problem solved?

3.5 Proof of work

Actually, the solution adopted is to introduce in the creation of blocks a proof of work that forces the node to make a computational effort to create a new block.
The block will actually consist of the following components:
- **Transactions**. The transactions are those that the node has received and are requested to enter in the chain. These are transactions it must validate previously to make sure they can be entered without creating collisions or inconsistencies that would cause the rest of nodes to reject the introduction of the block in the chain.
- **Coinbase (Base currency)**. It is a special transaction without origin. In essence is the generation of a new Bitcoin unit, and it is how monetary mass is generated in Bitcoin. Each block that is generated carries a new quantity for the node that generated the block.
- **Header.** A block header is generated with the following information:
    - **Hash of the transactions**. A hash of the transactions included in the block using the algorithm SHA256^2 (actually the SHA256^2 is simply applying two times the SHA256 algorithm). This hash is done using a Merkle tree or hash tree schema as detailed in Section 3.7.2.
    - **Hash from the previous block**. Each block includes the reference and hash of the previous block.
    - **Nonce**. It is a 32-bit block that can contain any value. It is the one used for proof of work.
    - **Block hash**. It will be the hash of the header. In the Bitcoin protocol it uses the SHA256^2 algorithm used by the Merkle-Damgard [15] and a double application of the algorithm.

Therefore, the blockchain would be more appropriate to represent it as follows:



But for a block to be considered valid, it must meet a condition special, and that is that its hash must start with a certain number of zero bits (0). The number of initial zero bits that the hash must satisfy determines the difficulty of creating a block, and it is something that the Bitcoin network adjusts for keeping constant the time it takes to create a block.

Therefore, assuming that the network forces the block hash to begin with 60 zero bits (15 characters '0' in hexadecimal), the block hash should be the following way:
0x000000000000000@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

How is it possible to generate a hash with a specific number of zeros at the start? Due to the very nature of the hash algorithms (and also the use of SHA256^2), there is no way to calculate or predict the result of a hash, therefore the only one way to get a hash with these properties is by brute force:
- The block hash is created from 3 elements: The hash of the transactions, the previous hash, and the nonce.
- The hash of the transactions and the previous hash are unalterable, but the nonce is a 32-bit number that can contain any value.
- Therefore, you must enter a nonce, do the hash of the header, and see if the result complies with the rule (starting with 60 zero bits). If not, the nonce is changed; the hash is re-done and checked again.
- The nonce is changed until a nonce is found which the hash has the number of desired initial zeros.

This method implies that a node, to construct a valid block, must use a relevant computing time. Although the time is totally random (a node could be lucky and find a valid nonce in its first attempt), the Bitcoin network tries the average time in generating a new block be 10 minutes. To maintain this average, and taking into account that the computational capacity of the nodes grows, adjusting the difficulty of the blocks increasing the number of initial zeros that the hash must have.

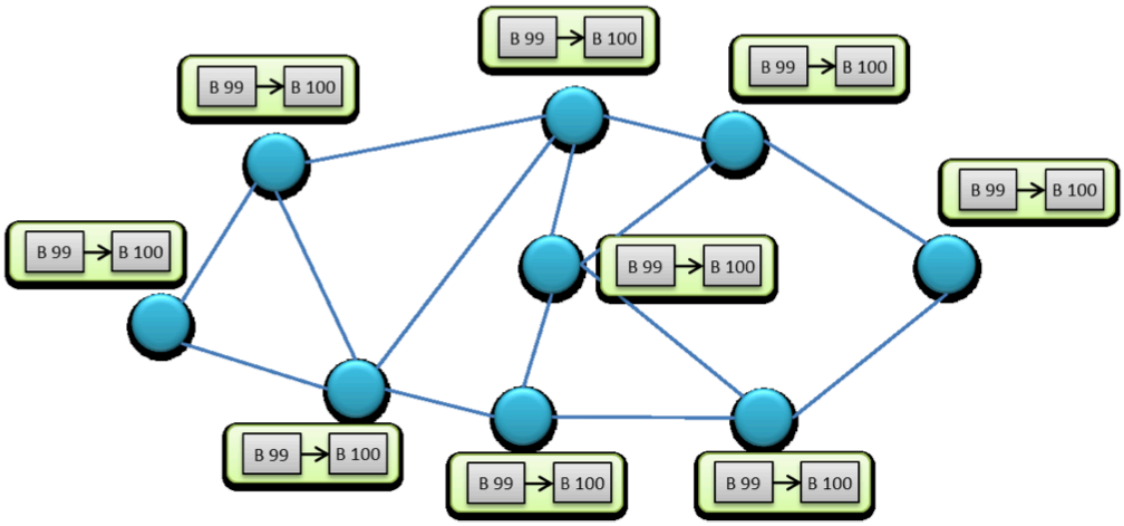
As an example, block 1 of Bitcoin, created the 2009-01-09 had a hash with 8 zeros in hexadecimal. However block 411932 created on 2016-05-15 already had a hash with 17 zeros in hexadecimal:
00000000000000000029277e2e42a48880461632c25512b1bd693ee32b6d6992d

This process of finding the right hash to generate a new block is usually known as "mining", and the nodes that perform this work as "mining nodes".

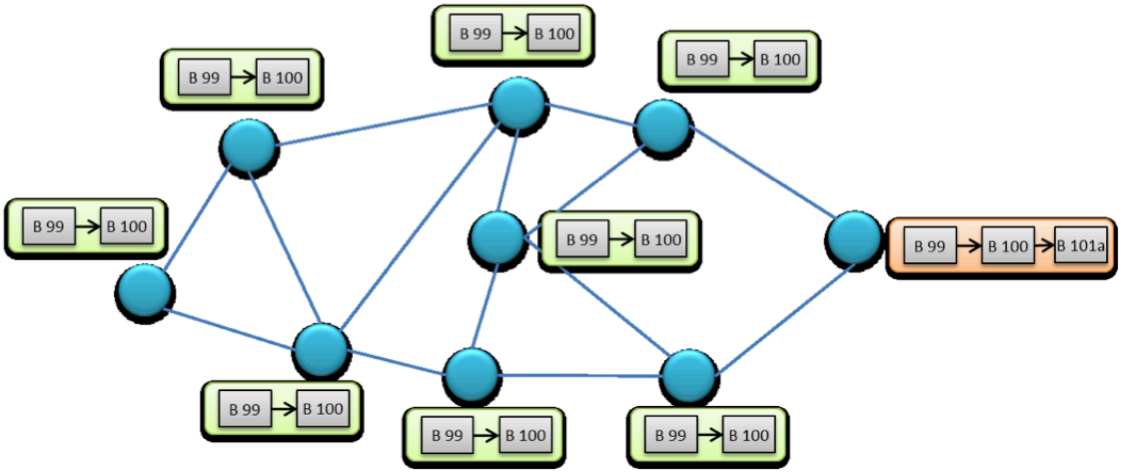
3.6 Consensus

Once the proof of work is introduced as a stabilizing factor for creation of the blocks, still to be established the mechanism to solve conflicts when two or more nodes create a new block and try to add it to the blockchain.

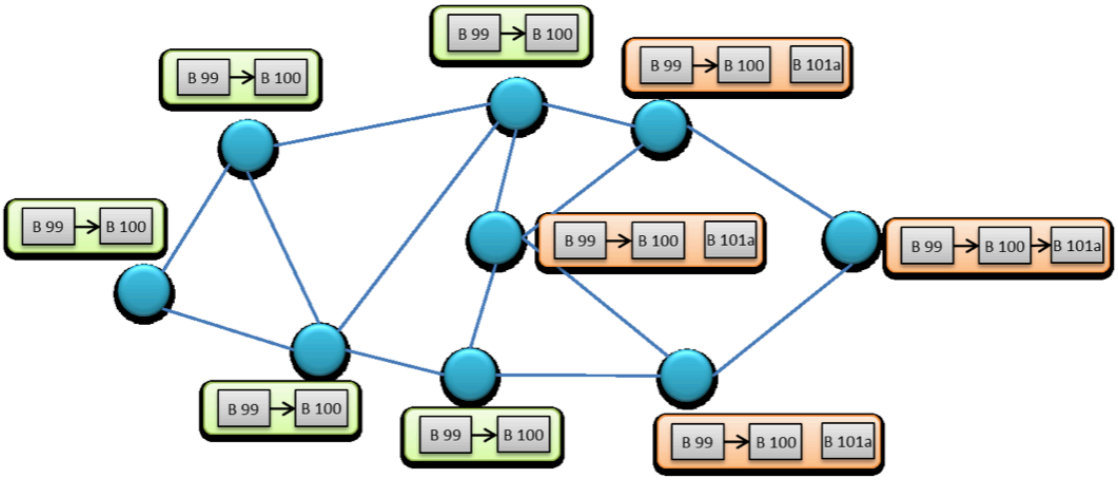
This situation is known as fork of the chain:
- It begins in a stable situation in which all the nodes of the network have the same chain (for example, 100 blocks).



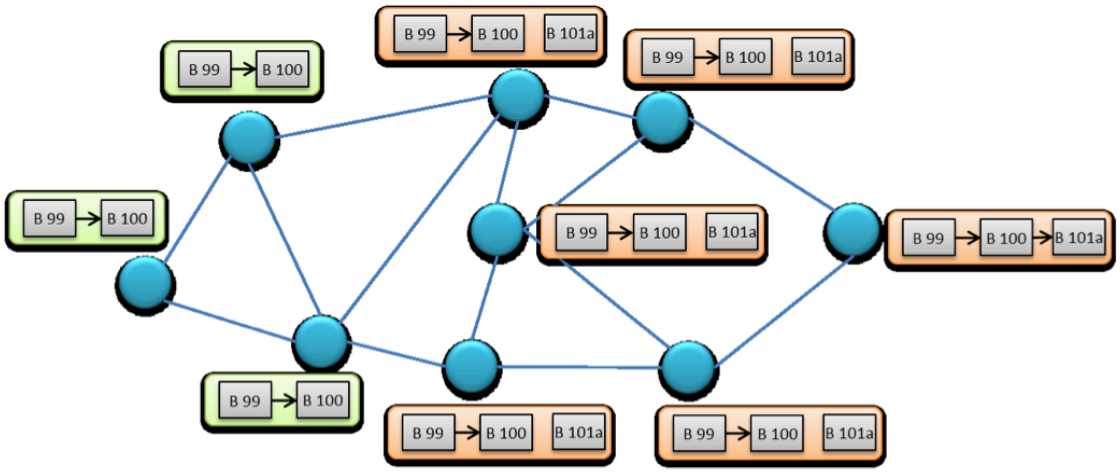
- Then a node manages to pass the proof of work and generates the block 101 (101a).



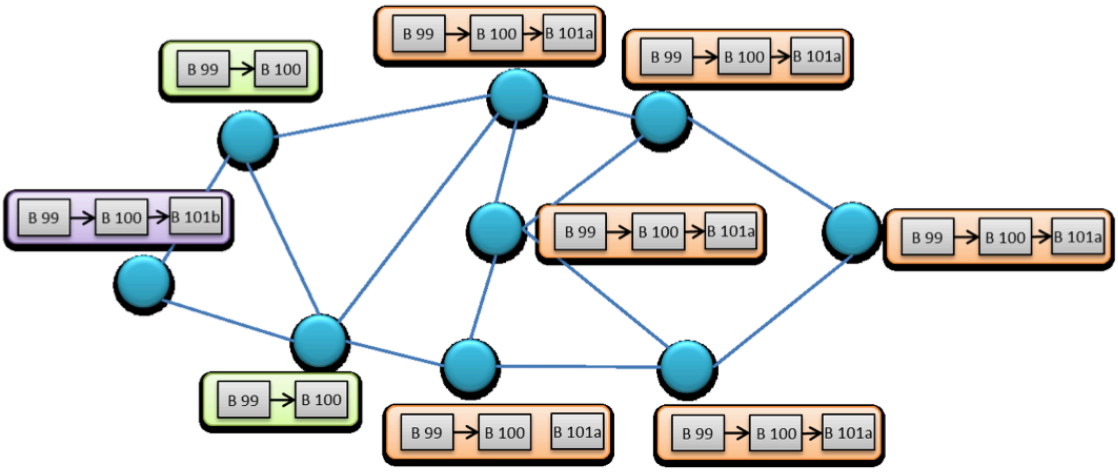
- The node adds the new block to the chain and transmits it to its closest nodes (those to which it is directly connected).



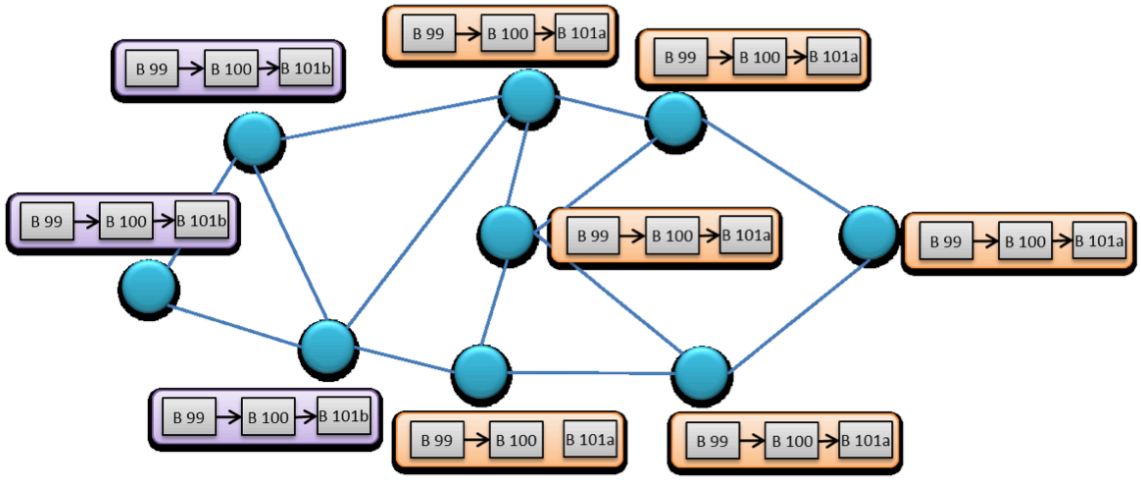
- These nodes verify the new chain, take it for valid and from now on they work with it trying to add new blocks. They relay it to other nodes by spreading them over the network.



- Another node, which has not yet reached the new chain and is still working with the chain of 100 blocks, it achieves the proof of work and also generates a new block 101 (101b).



- This node also adds it to the chain and retransmits it to its closest nodes.

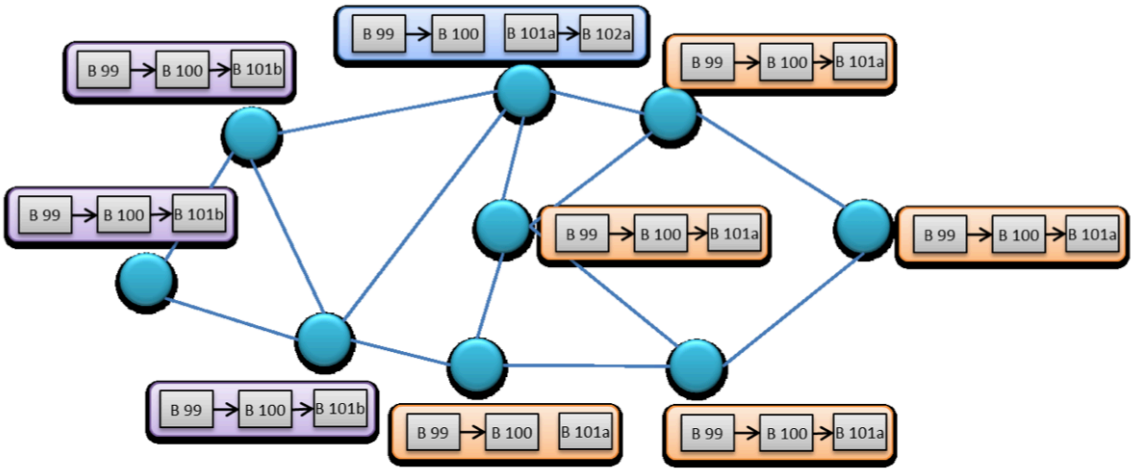


- At some point, the nodes start to receive two different chains of 101 blocks. The chain with 101a and the chain with 101b.

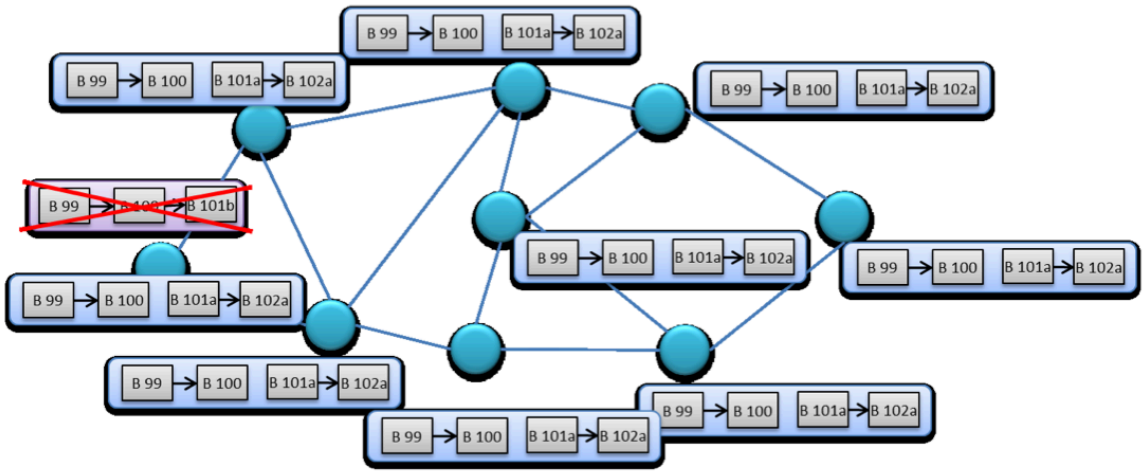
In this situation, it would be normal for some arbitration system to decide which is the correct string. In Bitcoin there is neither centralized system nor a centralized consensus. The consensus is determined by a single and simple rule: **the longest blockchain is the correct one**.

Therefore, in the previous case in which two nodes created a fork with two different chains 101a and 101b, some nodes will receive the string 101a and the they will take it as valid and work with it, others will validate 101b and work with it.

At some point another node will generate a new block and add it to one of the two chains (for example, 101a generating block 102a).



This new block will spread across the network and will reach nodes that are working with block 101b. Since block 102a is greater than 101b, these nodes should take it as valid, discard 101b, and start working from the 102a. Finally the network stabilizes with the chain 102a.



Block 101b is discarded (it is known as an "orphan" block) and has no validity. The transactions that were included in it are pending again (unless they have already been included in the new 102a block) and they will be included in the next block.

These situations do not happen very often, according to Bitcoin statistics, forks are produced only 2% of the time, which is approximately one fork every 50 blocks. Most forks are about one block and are resolved quickly. The forks that have more than one block (that is, a one-block fork and 2 nodes working with different versions of the chain; each one adding a new block at the same time continuing the fork) are very rare.

However, it must be taken into account that for a time and a group of nodes, block 101b was valid, and its transactions were validated. For this reason it is recommended to wait for chain validation where the transactions are by a sufficient number of nodes (usually 6) before declared committed.

3.7 Other characteristics of blockchain implementation

In the previous sections, a high-level description of the protocol and the solutions adopted by the Bitcoin implementation were portrayed. Even so, there are some peculiarities considered interesting that are described briefly below:

3.7.1 UTXO (Unspent Transaction Output)

To validate a transaction, you must validate that the source of the transaction is valid. In other words, the source transaction from which the funds are obtained is a transaction with sufficient funds and is valid.

This would mean that the node that wants to validate the transaction should go through the entire chain of blocks until it finds the block that contains the source transaction. Then go through all the subsequent blocks to make sure that transaction has not been used in another transaction.

Considering the size of the blockchain, and that it does not stop growing, this validation requires important computational resources. To optimize these resources the nodes create a special table called UTXO (Unused Transaction Outputs). What the node does is to perform the sweep of the block chain once. It goes through the blocks and it is verifying the transactions, noting in this cache the output transactions that have not been spent. For each transaction it deletes from the UTXO the input transaction and adds the output transaction.

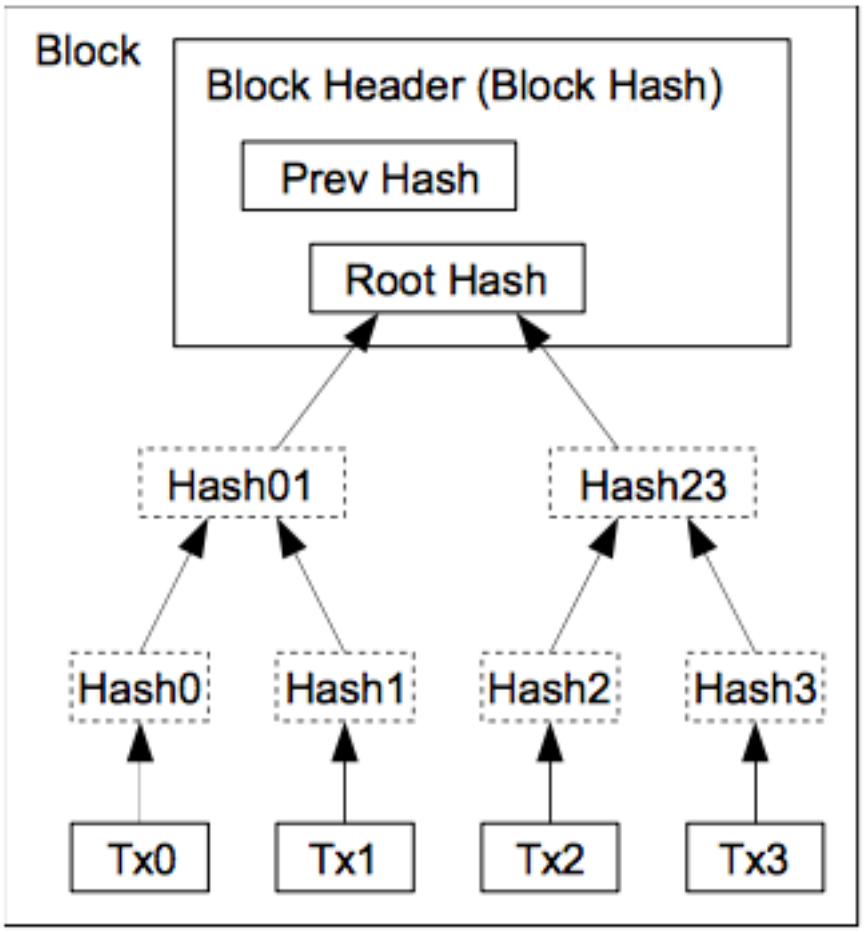
In this way the UTXO is an updated list of all the transactions of usable output. So to validate a transaction it is not necessary to scan again the whole blockchain; just check if the required output transaction is in the UTXO, greatly speeding the verification of the transaction.

3.7.2 Merkle Chain

As explained in section 3.5, at the header of each Bitcoin block resides a hash of all the transactions that the block incorporates. This way the hash ensures that no subsequent modification is made on the transactions, since any modification would alter the hash, and so for the header and the whole chain of blocks.

However, to perform this hash of all transactions will influence in the cost of validating them. For example if all the transactions are concatenated and the hash is generated, the verification would imply concatenate again all transactions, that is, to verify a transaction should calculate the hash of all transactions.

The Bitcoin protocol uses a more efficient algorithm that reduces the cost of calculating an individual transaction, the hash in the Merkle Tree [15].

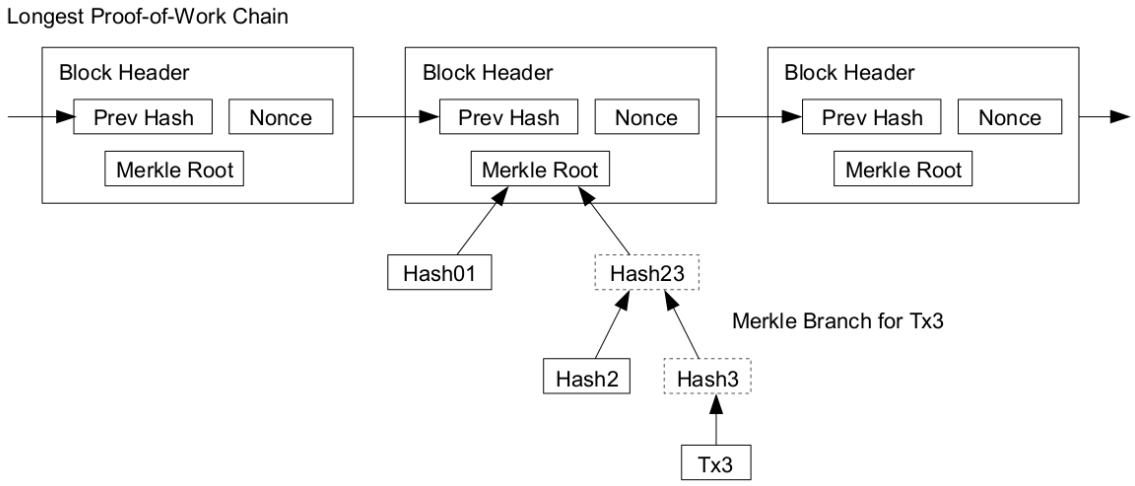


Transactions Hashed in a Merkle Tree

In essence, this hash is made up of a tree of hashes:
1. First perform the hash of each transaction, in what will be the most branched point of the tree, the leaf.
2. Next, each next parent level of the tree is a hash concatenating two hashes of the child level.
3. Finally, the root of the tree is where only one hash remains, the root hash that will be the one used in the block header.

This scheme is a great advantage when verifying a transaction, since it is not necessary to recalculate the hash of all the transactions of the block, but only the hash of the branch where the transaction to verify is. Similarly if a node

wants to modify a transaction of a block (before adding it to the chain) can recalculate the hash efficiently, just recalculating the branch of the transaction.



3.7.3 Rewards: mining and commissions

The security of the system relies on having a sufficient number of nodes with high computing mining power for generating new blocks. In order to happen, mining nodes must have incentives to spend their computational resources on this task. The Bitcoin network offers 2 types of incentives:
- Coinbase by mining. Each block can include a special transaction (coinbase) that has no intput and whose output can determine the mining node (usually assigns it to itself). It is as if the miner would have found a new currency (hence the simile of mining). This way, the mass of Bitcoins in circulation increases, but it is not infinite. The protocol is defined so that every 210,000 blocks (or every 4 years) the reward for each mined block is reduced by half. Initially each mined block generated 50 Bitcoins. As of May 2019, each block generates a coinbase of 12.5 Bitcoins and the mass of Bitcoins in circulation is approximately 17,600,000 Bitcoins. Following this model, the final estimated total mass would be about 21 million Bitcoins.

- **Commissions.** In each transaction, the user that generates it can leave a part of the funds without spending (sum of inputs > sum of outputs). In this case, the mining node that tries to incorporate the transaction into the blockchain can generate an output by assigning those remaining funds to it. This is known as commission, a payment is assumed by the mining node and as an incentive (mining nodes are interested in including the transactions with the highest commission inside the blocks).

Thanks to these two incentives, the mining nodes receive a reward for devoting its resources to the Bitcoin network. At the beginning of the network, and because there were no transactions, the majority of blocks were empty blocks in which the coinbase transaction simply appeared. That is, it was only to obtain new Bitcoins. As the network grows and starts to offer transactions commissions, the miners care more about including transactions. Due to the policy of decreasing the value of coinbase, the commission takes more and more prominence.

3.8 Disadvantages, weaknesses and attacks on Bitcoin

Like any system, although it is created specifically to be secure, certain drawbacks of the system have been detected as well as possible attacks and failures.

3.8.1 High latency

It is one of the problems of the Bitcoin network: it is not immediate.
To make a payment or a bitcoin transaction, it must:
- Generate a transaction.
- Send the transaction to one or more nodes.
- Some node must include the transaction in the blockchain (noting that the time between blocks is set to 10 minutes, the inclusion time for a new transaction will be at least 5 minutes, but it can be more).
- The node must replicate the new blockchain to the rest of the nodes.
- The blockchain should stabilize as a dominant chain (usually wait for a significant number of nodes have accepted it and are already working with it).

This whole process can last for tens of minutes, therefore a payment or Bitcoins transaction is not immediate.

This feature is intrinsic to its architecture and could not be changed without a profound change of the essence of Bitcoin.

For this reason the Bitcoin network cannot be used in environments that need a very high transaction speed and transaction validity (i.e. financial environments or stock markets in which prices fluctuate very fast and the purchase and sale of equity must be executed quickly).

3.8.2 Bitcoin software

Being a distributed network, and by consensus, each node is executing software independently from others. This software maintains business rules of the system that allow validations, rejection of double spending, etc.

A node could alter its software to change the rules to its favor, but if the rest of the nodes of the network continue working with the agreed software and rules, they simply reject the blocks generated by the attacking node, and this will not have any impact on the system. On the contrary, it will not receive compensation for its mined blocks, so in general, it is more convenient for the nodes to act in a legitimate way.

However, there have been problems due to updates of the Bitcoin software. Like any software sometimes errors are found or introduced improvements that are agreed by the community. These improvements are specifid in a new version of the software that is distributed to the nodes.

The problem here is that the different nodes may be using different versions of the software, and therefore apply different rules.

In 2013, due to a version upgrade, from 0.7 to 0.8, an involuntary change in the rules made the nodes with version 0.8 validate blocks that were given as invalid by version 0.7. This produced an important fork of the blockchain. Although, because most nodes were updated to version 0.8 the system was imposing this criterion, but as version 0.8 acted incorrectly it was decided by the community to downgrade of all nodes to 0.7 and take as base the blockchain of these nodes [16].

3.8.3 Double expenditure attacks [17]

A double-spending attack involves trying to launch two transactions that use the same source funds. The Bitcoin network solves the problem in a decentralized way: it takes as valid the first transaction that is included in the block and rejects the next one.

As the block containing the transaction is linked to new blocks, the transaction is more secure, because if an attacker wants to alter that transaction, he/she should alter that block and all the posterior blocks, and do it before some other node adds a new block, because if it adds new blocks, he/she must alter those new ones also.

As the way to generate a new block is mainly based on hash generation, for an attacker to be able to generate alone blocks faster than the entire network, its ability to generate hashes should be greater than the capacity of the rest of the network. This is known as 51% attack (since the attacker must have at least 51% of hash generation power of the entire network).

It would also be possible to achieve it with less than 50% power, but the probability of success of the attack depends on the hash rate percentage that the attacker controls and the number of blocks that must alter, decreasing exponentially with both factors.

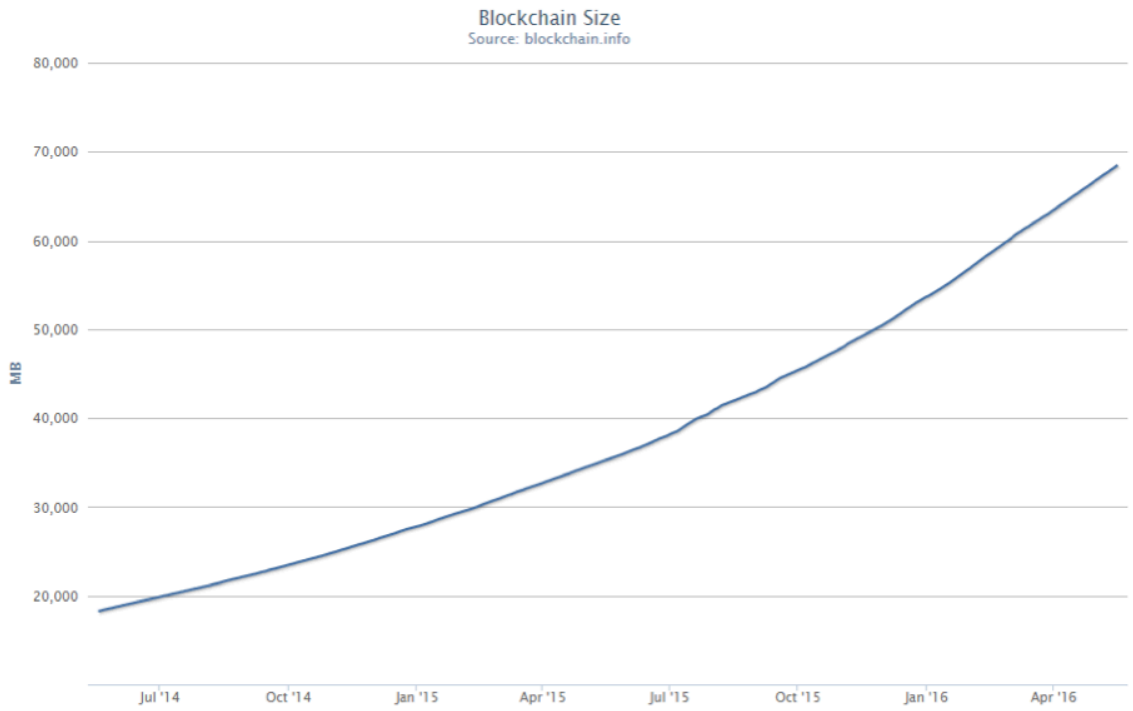
Keeping in mind that although this type of attack could alter the blocks, it could not alter transactions. As the transactions are secured by public-key cryptography the attacker could not alter them. What I could do is eliminate them from the block. So he/she could make a payment, wait for the receiver to confirm the transaction and get the service, then alter the chain to eliminate that payment and enter a new payment with the same funds.

In any case, the investment required to obtain the enough computing power is huge, and probably would not make up for the benefits obtained.

3.8.4 Scalability

The blockchain does not stop growing. Each new block is added to the previous one. Currently, each block is limited to a maximum of 1 MByte. But taking into account that on average every 10 minutes a new block is generated, and if we assume that on average each block occupies 500KBytes, it means that the chain grows 3MBytes per hour. In May 2019, the Bitcoin blockchain occupied already 252.41 GBytes.

In Figure 20 you can see the growth curve of the blockchain.



Another factor is the existing limitation in the number of transactions that the network is capable of processing. If a block has a limitation (currently 1 MByte) and therefore can store a number maximum of transactions (the average of a transaction is 0.5 KBytes, a block could store a maximum of 2000 average transactions)

Keeping in mind that the network ensures that there is a new block every 10 minutes, it turns out that Bitcoin is only capable of processing 200 average transactions per minute. It is a very low processing capacity for a payment system, but increasing the block size can regulate it, so that more transactions fit in.

3.9 Conclusions of the blockchain technology analysis

Thanks to its blockchain protocol, and despite some of its limitations, Bitcoin has become the most popular decentralized cryptocurrency. Being recognized actor in the financial markets.
Some information as of May 2019 [18]:
- Exchange: 1 Bitcoin (BTC) is exchanged to USD $5,000.
- Money supply: 17,600,00 BTC (approx.).
- Total capitalization: USD $94,212,497,477 (approx USD value of all Bitcoins in circulation).

- Volume of transactions: Approx. on average 244 transactions per minute.
- Value of transactions: Approx. daily transactions by value of USD $10,096,938,823.

From these data it can be inferred the success of Bitcoin, and therefore of its protocol blockchain and the applied solutions:
- No users, only addresses or accounts created using public key cryptography from outside the Bitcoin network.
- Transactions use keys address generated for security and anonymity.
- The transactions are added to the blockchain, which has the history of all transactions made on the network, so that it can always verify if the new transaction is valid.
- To add a new block to the chain, it must pass a proof of work, this way the rate of insertion of new blocks is regulated and the possibility of a denial of service attack is prevented.
- The way to achieve consensus in case of parallel blocks (fork) is simple: the longest block is the valid one.

In addition, for being the first case and more successful use of blockchain, we must bear in mind that this protocol is "neutral". It's simply a data logging protocol and distributed consensus. Although it has been applied on the field of Cryptocurrencies, it could perfectly be applied to other areas.

In fact, several initiatives have emerged that propose the use of the blockchain protocol in different areas. Some examples would be:
- Changetip (www.changetip.com), which is aimed at making micropayments using the funds deposited in Bitcoin.
- Ethereum (www.ethereum.org), which is a decentralized platform based on blockchain for the execution of smart contracts [19]. The contracts are coded in the blockchain and executed independently by the system. For example, it could be a scheduled payment from one account to another under certain conditions, and the own system would be responsible for checking the conditions and executing the payment autonomously.
- Ascribe (www.ascribe.io), uses blockchain technology for management of copyright of digital content.
- Eris (erisindustries.com), is also a platform based on blockchain for the management of smart contracts.
- Followmyvote (followmyvote.com), which offers a platform for online voting using blockchain technology to guarantee the integrity of the vote, in a centralized environment.

Taking into account all the characteristics, advantages and possibilities of the blockchain protocol, as well as the limitations of current systems for electronic voting, the development of an electronic decentralized voting solution is proposed based on blockchain technology.

Chapter 4

System Design

The proof-of-concept implementation is written in Ethereum's solidity language [8]. The system's user interface is implemented the using HTML5 and JavaScript. The application is extremely simple, all it does is initialize a set of contestants, let anyone vote for the candidates and display the total votes received by each candidate. The goal is not to just code an application but to learn the process of compiling, deploying and interacting with it.

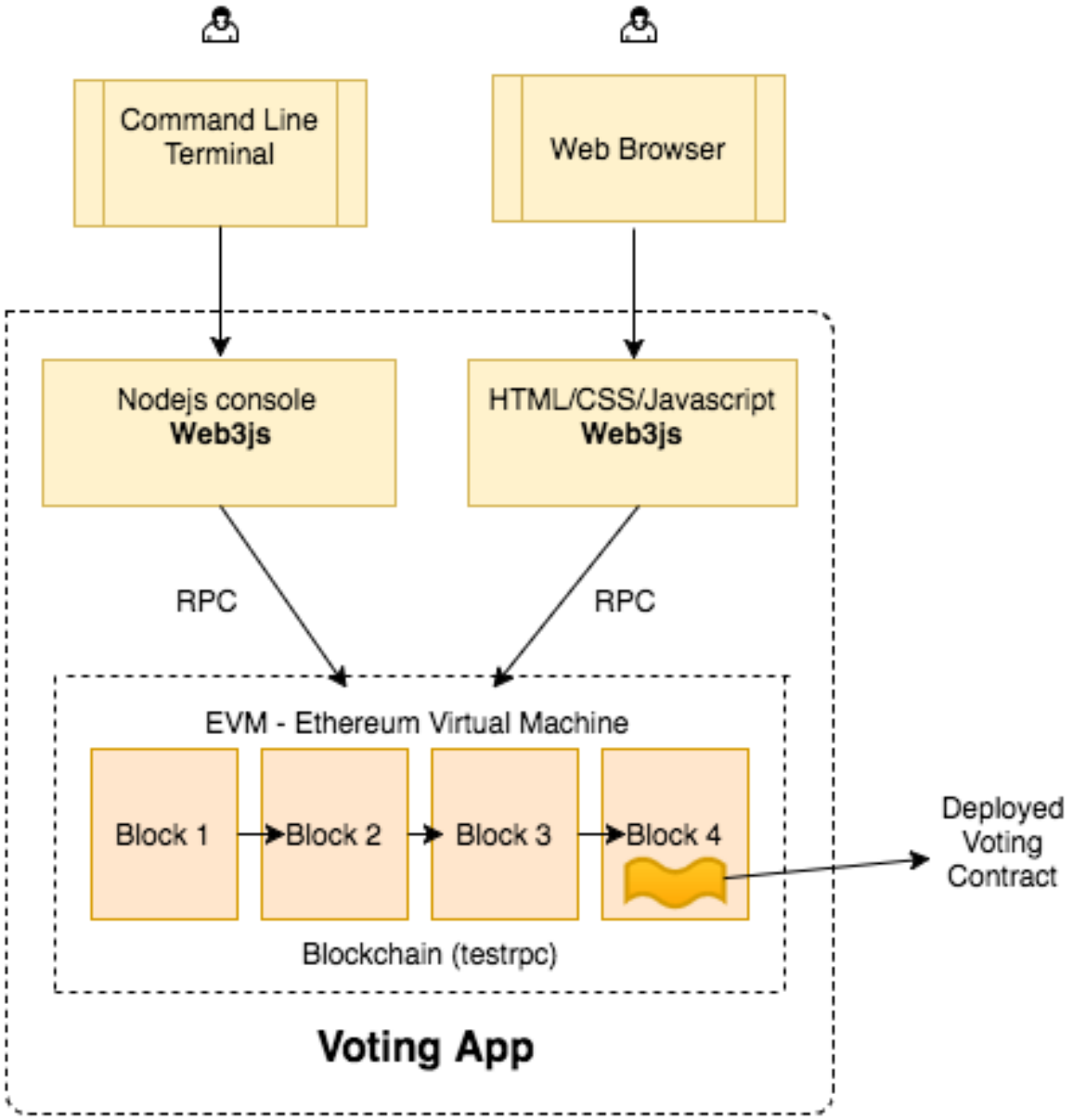
I have deliberately avoided using any frameworks to build this application because the frameworks abstract away lot of the details and hide the internals of the system.

The goal of this implementation is to:

1. Set up the development environment.
2. Learn the process of writing a contract, compiling it and deploying it in your development environment.
3. Interact with the contract on the blockchain through a nodejs console.
4. Interact with the contract through a simple web page to display the vote counts and vote for candidates through the page.

4.1 Architecture

The entire application set up and build was done on a fresh installation of Ubuntu 16.04 Xenial. I have set up and tested the application on Mac OS as well. This is the architecture visualization for this application.



4.2 Setting up the Environment

Instead of developing the app against a live blockchain, I will use an in-memory blockchain (as a blockchain simulator) called ganache. Below are the steps to install ganache, web3js and start the test blockchain on a linux operating system. The exact same instructions work on Mac OS as well.

```
mahesh@zastrin:~$ npm -v
6.1.0
mahesh@zastrin:~$ node -v
v10.4.0
mahesh@zastrin:~$ mkdir -p ethereum_voting_dapp/chapter1/
mahesh@zastrin:~$ cd ethereum_voting_dapp/chapter1
mahesh@zastrin:~/ethereum_voting_dapp/chapter1$ npm install ganache-cli
web3@1.0.0-beta.37
mahesh@zastrin:~/ethereum_voting_dapp/chapter1$
node_modules/.bin/ganache-cli
The output should look like below
Ganache CLI v6.3.0 (ganache-core: 2.4.0)
Available Accounts
==================
(0) 0x66d1dda957bc7a087324130c89547ce579cee563 (~100 ETH)
(1) 0xf98e7d03f312c8329d01ef0642b74e77740cc203 (~100 ETH)
(2) 0x71d4ee31cd935d17a2b00b7d8873651d6aefbf73 (~100 ETH)
(3) 0x2fdaa99748e6224e03e3e5a1a9a89abd906df671 (~100 ETH)
(4) 0xfb32d320a858747453ac1b4e23c849fecd6ebf33 (~100 ETH)
(5) 0x2c00cd8a488a8ae569553dfe90bf90d65af97630 (~100 ETH)
(6) 0xa4099d0d4a0e7ce5bf26c3f4078a001daf5cedff (~100 ETH)
(7) 0xa53c14d56683bdc775c19540679b0809bbf95a2b (~100 ETH)
(8) 0x3ba642d43a2a24a0160fa22c358cf697aa4a627f (~100 ETH)
(9) 0x81c43af78fdb4a2c8078ec37875e9616e3c5a3e4 (~100 ETH)
Private Keys
==================
(0) 0xee670c7915b5e73b6f5c82da3919701f19ed1f72a1d4f31007836930623f4a57
(1) 0xf40063c41978799ff6fa4b85d678b0f42ed6c774c35fa883a289fdc7fa0b65cb
(2) 0x5228a5e30d1be11885cd7d44d4a47b1dfa22db4033cd758a1c3677a3a0cb5383
(3) 0x3520c609b830538bd8c5d05c8f53338322411deb477308c5543f20a420fea5ec
(4) 0xa7728cbc1eee41d696cb1d252157fc66ab1726ff751f2132c387726b5c901c35
(5) 0xe459627dc49d59da76c393b40c36c674c8307ec52c24794a33ddc773f3c229fc
(6) 0xa65715e3bdc8236420491de88486ff301a19b624b006811a89742e74120eac50
(7) 0xc95ab6498d506f45b4b8ab2b04967d51e05af41059334df91d63097cb3254a3b
(8) 0xf0619031f2ba4319aae691ce34fe8cce136640fb715b70d5baa045c2d58f1b80
(9) 0x39a85154d16a1eba4a9f8514f187bcb6c8b867b790035afa9f92bf2731aebc5b
HD Wallet
==================
Mnemonic:      rather online beyond apology whale cheese game ankle
together share potato leisure
Base HD Path:  m/44'/60'/0'/0/{account_index}
```

Notice that the ganache-cli creates 10 test accounts to play with automatically. These accounts come preloaded with 100 (fake) ethers.

4.3 Simple Voting Contract

I use the solidity programming language to write our contract. If there is familiarity with object oriented programming, learning to write solidity contracts should be easy. I write a contract (like a class in OOP paradigm) called Voting with a constructor, which initializes an array of candidates. I write two methods, one to return the total votes a candidate has received and another method to increment vote count for a candidate.

Note that the constructor is invoked once and only once when deploying the contract to the blockchain. Unlike in the web where every deployment of code overwrites the old code, deployed code in the blockchain is immutable. i.e, if there is an update on the contract and is deployed again, the old contract will still be in the blockchain untouched along with all the data stored in it, the new deployment will create a new instance of the contract.

Below is the voting contract code with inline comment explanation:

```
pragma solidity >=0.4.0 <0.6.0;
// We have to specify what version of compiler this code will compile with

contract Voting {
  /* mapping field below is equivalent to an associative array or hash.
  The key of the mapping is candidate name stored as type bytes32 and value is
  an unsigned integer to store the vote count
  */

  mapping (bytes32 => uint256) public votesReceived;

  /* Solidity doesn't let you pass in an array of strings in the constructor (yet).
  We will use an array of bytes32 instead to store the list of candidates
  */

  bytes32[] public candidateList;

  /* This is the constructor which will be called once when you
  deploy the contract to the blockchain. When we deploy the contract,
  we will pass an array of candidates who will be contesting in the election
  */
  constructor(bytes32[] memory candidateNames) public {
    candidateList = candidateNames;
  }

  // This function returns the total votes a candidate has received so far
  function totalVotesFor(bytes32 candidate) view public returns (uint256) {
    require(validCandidate(candidate));
    return votesReceived[candidate];
  }

  // This function increments the vote count for the specified candidate. This
  // is equivalent to casting a vote
  function voteForCandidate(bytes32 candidate) public {
    require(validCandidate(candidate));
    votesReceived[candidate] += 1;
  }

  function validCandidate(bytes32 candidate) view public returns (bool) {
    for(uint i = 0; i < candidateList.length; i++) {
      if (candidateList[i] == candidate) {
        return true;
      }
    }
    return false;
  }
}
```

The above code is copied into a file named Voting.sol in the ethereum_voting_dapp/chapter1 directory. Next is to compile the code and deploy it to ganache blockchain.

To compile solidity code, installation of an npm module called solc is required. Then contract is compiled.

```
mahesh@zastrin:~/ethereum_voting_dapp/chapter1$ npm install solc@0.5.3
mahesh@zastrin:~/ethereum_voting_dapp/chapter1$
node_modules/.bin/solcjs --bin --abi Voting.sol
mahesh@zastrin:~/ethereum_voting_dapp/chapter1$ ls
Voting.sol              Voting_sol_Voting.abi   Voting_sol_Voting.bin
```

When the code is successfully compiled using the command above, the compiler outputs two files that are important to understand:

1. Voting_sol_Voting.bin: This is the bytecode you get when the source code in Voting.sol is compiled. This is the code which will be deployed to the blockchain.
2. Voting_sol_Voting.abi: This is an interface or template of the contract (called ABI), which tells the contract user what methods are available in the contract. Whenever there is an interaction with the contract in the future, this abi definition is needed.

Web3js is a library that allows interaction with the blockchain through RPC. I use that library to deploy the application and interact with it.

First, the 'node' command is run in a terminal to get into node console and initialize the web3 object. All the code snippets below need to be typed in the node console.

```
mahesh@zastrin:~/ethereum_voting_dapp/chapter1$ node
> Web3 = require('web3')
> web3 = new Web3("http://localhost:8545")
```

To make sure web3 object is initialized and can communicate with the blockchain, a query to all the accounts in the blockchain is made. An output like shown below should result:

```
> web3.eth.getAccounts(console.log)
['0x9c02f5c68e02390a3ab81f63341edc1ba5dbb39e',
'0x7d920be073e92a590dc47e4ccea2f28db3f218cc',
'0xf8a9c7c65c4d1c0c21b06c06ee5da80bd8f074a9',
'0x9d8ee8c3d4f8b1e08803da274bdaff80c2204fc6',
'0x26bb5d139aa7bdb1380af0e1e8f98147ef4c406a',
'0x622e557aad13c36459fac83240f25ae91882127c',
'0xbf8b1630d5640e272f33653e83092ce33d302fd2',
'0xe37a3157cb3081ea7a96ba9f9e942c72cf7ad87b',
'0x175dae81345f36775db285d368f0b1d49f61b2f8',
'0xc26bda5f3370bdd46e7c84bdb909aead4d8f35f3']
```

To compile the contract, load the bytecode and abi from the file system in to a string like below.

```
> bytecode = fs.readFileSync('Voting_sol_Voting.bin').toString()
> abi = JSON.parse(fs.readFileSync('Voting_sol_Voting.abi').toString())
```

Next is the deployment of the contract. First is to create a contract object (deployedContract), which is used to deploy and initiate contracts in the blockchain.

```
> deployedContract = new web3.eth.Contract(abi)
> listOfCandidates = ['Rama', 'Nick', 'Jose']
> deployedContract.deploy({
  data: bytecode,
  arguments: [listOfCandidates.map(name =>
web3.utils.asciiToHex(name))]
}).send({
  from: 'ENTER 1 OF 10 ACCOUNT ADDRESSES like 0xfb3....',
  gas: 1500000,
  gasPrice: web3.utils.toWei('0.00003', 'ether')
}).then((newContractInstance) => {
  deployedContract.options.address =
newContractInstance.options.address
  console.log(newContractInstance.options.address)
});
```

I use the web3 deploy function along with send function to deploy the contract to the blockchain. Let's see what are all the arguments we pass to deploy and send functions:

1. **data**: This is the compiled bytecode which is deployed to the blockchain.
2. **arguments**: These are the arguments passed to the constructor of the contract. In this case we pass an array of candidate names. Note that it is explicitly converted from string to bytes32, that's why web3.utils.asciiToHex is called on each candidate name (using map function).
3. **from**: The blockchain has to keep track of who deployed the contract. In this case, the first account that gets back from calling web3.eth.getAccounts is picked and is the owner of this contract (who will deploy it to the blockchain). Remember that web3.eth.getAccounts returns an array of 10 test accounts ganache created when the test blockchain started. In a live blockchain, you can not just use any account. The user must have ownership of that account and unlock it before transacting. A passphrase is asked while creating an account and that is what the user uses to prove ownership of that account. Ganache by default unlocks all the 10 accounts for convenience.
4. **gas**: It costs money to interact with the blockchain. This money goes to miners who do all the work to include your code in the blockchain. The account that deploys has to specify how much money he/she is willing to pay to get his/her code included in the blockchain and does that by setting the value of 'gas'. The ether balance in the 'from' account will be used to buy gas. The network sets the price of gas.
5. **gasPrice**: Each unit of gas has a price associated with it. That is set in the gasPrice field.

The contract is deployed and there is an instance of it (variable deployedContract above), which we can use to interact with the contract. There are hundreds of thousands of contracts deployed on the Ethereum blockchain. So, how do we identify the contract in that blockchain? Answer: deployedContract.options.address. When the user has to interact with the contract, he/she needs this deployed address and the abi definition mentioned earlier.

4.4 Interaction with the Contract in the Nodejs Console

The following commands in are executed in the node console and it should increment the vote count.

```
>
deployedContract.methods.totalVotesFor(web3.utils.asciiToHex('Rama')).c
all(console.log)
>
deployedContract.methods.voteForCandidate(web3.utils.asciiToHex('Rama')
).send({from: 'YOUR ACCOUNT ADDRESS'}).then((f) => console.log(f))
>
deployedContract.methods.totalVotesFor(web3.utils.asciiToHex('Rama')).c
all(console.log)
```

Every time a vote is casted for a candidate, a transaction id gets back. For example:
'0xdedc7ae544c3dde74ab5a0b07422c5a51b5240603d31074f5b75c0ebc786bf53'

This transaction id is the proof that the transaction occurred and it can be referred back any time in the future. This transaction is immutable. This immutability is one of the big advantages of blockchains such as Ethereum.

4.4 GUI Interface to Blockchain and Vote

Now that most of the work is done, the development of the user interface is about creating a simple html file with candidate names and invoke the voting commands (which were already tested in the nodejs console) in a js file. Below is the html code and the js file. Both of them are in the ethereum_voting_dapp/chapter1 directory and the index.html is opened in a browser.

index.html

```
<!DOCTYPE html>
<html>
<head>
 <title>Voting DApp</title>
 <link href='https://fonts.googleapis.com/css?family=Open Sans:400,700'
rel='stylesheet' type='text/css'>
 <link
href='https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css'
rel='stylesheet' type='text/css'>
</head>
<body class="container">
 <h1>A Simple Hello World Voting Application</h1>
 <div class="table-responsive">
  <table class="table table-bordered">
   <thead>
    <tr>
     <th>Candidate</th>
     <th>Votes</th>
    </tr>
   </thead>
   <tbody>
    <tr>
     <td>Rama</td>
     <td id="candidate-1"></td>
    </tr>
    <tr>
     <td>Nick</td>
     <td id="candidate-2"></td>
    </tr>
    <tr>
     <td>Jose</td>
     <td id="candidate-3"></td>
    </tr>
   </tbody>
  </table>
 </div>
 <input type="text" id="candidate" />
 <a href="#" onclick="voteForCandidate()" class="btn btn-primary">Vote</a>
</body>
<script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js@1.0.0-
beta.37/dist/web3.min.js"></script>
<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>
<script src="./index.js"></script>
</html>
```

index.js

```
web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:8545"))
var account;
web3.eth.getAccounts().then((f) => {
 account = f[0];
})

abi =
JSON.parse('[{"constant":true,"inputs":[{"name":"candidate","type":"bytes32"}],"na
me":"totalVotesFor","outputs":[{"name":"","type":"uint8"}],"payable":false,"stateMu
tability":"view","type":"function"},{"constant":true,"inputs":[{"name":"candidate","ty
pe":"bytes32"}],"name":"validCandidate","outputs":[{"name":"","type":"bool"}],"pay
able":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[{"na
me":"","type":"bytes32"}],"name":"votesReceived","outputs":[{"name":"","type":"uin
t8"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inp
uts":[{"name":"","type":"uint256"}],"name":"candidateList","outputs":[{"name":"","ty
pe":"bytes32"}],"payable":false,"stateMutability":"view","type":"function"},{"constan
t":false,"inputs":[{"name":"candidate","type":"bytes32"}],"name":"voteForCandidat
e","outputs":[],"payable":false,"stateMutability":"nonpayable","type":"function"},{"in
puts":[{"name":"candidateNames","type":"bytes32[]"}],"payable":false,"stateMutab
ility":"nonpayable","type":"constructor"}]')

contract = new web3.eth.Contract(abi);
contract.options.address =
"0x71789831d83d4C8325b324eA9B5fFB27525480b5";

candidates = {"Rama": "candidate-1", "Nick": "candidate-2", "Jose": "candidate-
3"}

function voteForCandidate(candidate) {
 candidateName = $("#candidate").val();
 console.log(candidateName);


contract.methods.voteForCandidate(web3.utils.asciiToHex(candidateName)).sen
d({from: account}).then((f) => {
  let div_id = candidates[candidateName];

contract.methods.totalVotesFor(web3.utils.asciiToHex(candidateName)).call().th
en((f) => {
   $("#" + div_id).html(f);
  })
 })
}
```

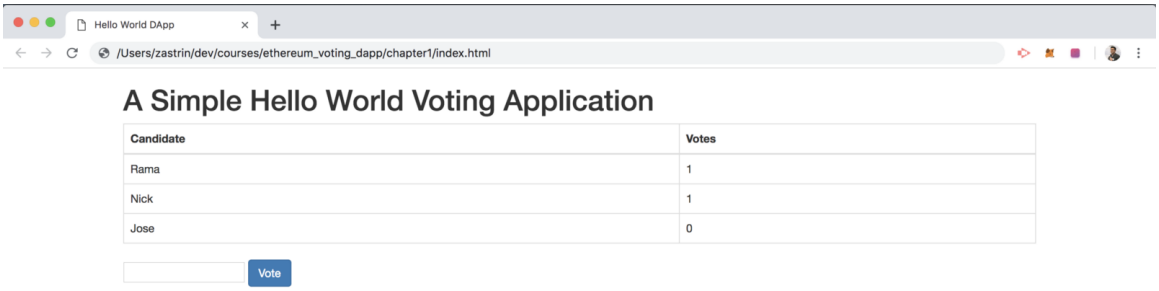
```
$(document).ready(function() {
 candidateNames = Object.keys(candidates);

 for(var i=0; i<candidateNames.length; i++) {
 let name = candidateNames[i];

 contract.methods.totalVotesFor(web3.utils.asciiToHex(name)).call().then((f) => {
  $("#" + candidates[name]).html(f);
 })
 }
});
```

As said earlier, the abi and the address are needed to interact with any contract. They can bee seen above in green in the index.js file and how they are used to interact with the contract.

Opening the index.html in a browser and it should render something like this:



The user is now able to enter the candidate name in the text box and vote and see the vote count increment. To summarize, the environment was set up, a simple contract coded, compiled and deployed on the blockchain. The user can interact with it via nodejs console or through a webpage.

Chapter 5

System Evaluation

5.1 Evaluation

From the initial planning of this project, I can say that I have met the objectives I have set out for this project.

At the start of this project my idea was to research and develop a custom blockchain suited for voting. But as my researched progressed on the subject, I noticed that there are already mature and well-developed blockchains that can carry a voting application. Such is the case of Ethereum, the blockchain I selected for this project.

Initially there were three blockchain candidates to develop this project: Bitcoin blockchain, Ethereum and EOS. The reason that I discarded Bitcoin was because it is specifically designed for cryptocurrency transaction, not giving any chance to create more elaborate applications on it. The Ethereum blockchain incorporates an Ethereum Virtual Machine (EVM) that allows execution of deployed code in the Ethereum blockchain. This opens the possibility to use blockchain technology for a lot of applications.

As for the remaining option, EOS blockchain, it has most of the features that come with the Ethereum blockchain like smart contracts and it has a better transaction throughput than the Ethereum. It uses Delegated Proof of Stake (DPoS) to incorporate new blocks into its blockchain. This mechanism allows EOS to outperform Ethereum's throughput. The downside is that DPoS is achieved using a semi-centralized network. Such infrastructure could be biased to particular voting results. Deciding between the application's performance and the vote integrity, I decanted to use the Ethereum blockchain.

Other set of objectives for this project was to learn and improve my coding skills and time management skills. I have improved my coding abilities during the course of the development of the project in the specific areas, which are all of the technologies I used in this project such as Solidity, JavaScript, HTML, Web3, Nodejs and geth.

I have also gained a lot of experience in time management, meeting goals and deadlines and managing software on Github. As I followed a weekly development plan for my project I made sure that I stay on track and meet all of my weekly plans. I also gained some additional knowledge with using software-managing tools such as Github.

5.2 Limitations & Opportunities

Scaling up the solution to national elections requires addressing limitations in both Ethereum's blockchain and the e-voting contract.

Using Ethereum as deployed today, only one vote can be cast in one block. Given that each block is generated every 12 seconds, this means only five votes per minute can be cast over the blockchain. Take the 2016 UK Referendum an example. For 5.2 million votes (the number of postal votes in that election), it would require 722 days for all votes to be recorded into Ethereum's blockchain. To support national scale elections, a dedicated Ethereum-like blockchain will be required. Such a blockchain will provide a consistent global database that all voters have access to and guarantee that all inserted data remain immutable. Election audit data sent to the blockchain will be verified by independent validators who act as 'miners' and get awarded for verifying the audit data and maintaining the blockchain. Each block should allow storing more votes by increasing the gas limit and new blocks may be generated at a faster speed than the current 12 seconds per block.

There exist few debugging tools for these smart contracts. The best practice is to create an 'event' that logs data along with the contract. These events need to be incorporated into the program before compiling the contract, and they do not allow running the code step by step.

Since blockchain is decentralized, there is more incentive to make the voting protocol more decentralized as well, while not making a particular government a trusted central party issuing and monitoring the vote tokens and the votes. Complete decentralization of the protocol would be the desired outcome, however the lack of an acceptable transaction rate of Ethereum platform currently poses as an issue to the fundamental concepts of voting. Steady advancements in development of Ethereum platform bring the creation of this protocol closer. Buterin, the co-founder of Ethereum, has given some updates on the direction of the Ethereum and its pending upgrade named "Byzantium".

Byzantium is an update which aims to improve Ethereum, and although the exact details of the update are unconfirmed

The other potential for improvement of the proposed contract is for the registration side. In accordance to the security considerations mentioned above, it is possible to request more private information during the registration stage, or create two-factor authentication process with user's mobile device of choice.

However, with the Ethereum protocol, perhaps, this centralized aspect of the protocol can be removed completely by implementation of a contract. If a government body was to issue a voting contract with the candidates defined within, any person who wishes to vote would be able to vote, and the rules of the election will be enforced by how the contract is created

Chapter 6

Conclusions

The success of Bitcoin and the protocol that sustains it, blockchain, has awakened interest in the uses that can be made from this database technology and distributed consensus technologies.

From this interest arise some initiatives such as other cryptocurrencies, Ethereum (which registers self-executing contracts), ChangeTip (micropayments) or even in the field of voting (FollowMyVote).

In the field of voting, although in recent years there are attempts to apply new technologies to electoral processes, it has not been managed yet to implement a system that offers the same security to voters than the traditional system.

Although there are good solutions based on secure technologies, there is distrust from the elector, and even of experts, on technologies for telematic voting that would control a centralized system, possibly managed by some of the participants in the electoral contest.

A system of this type would be easy to manipulate if the interested party has direct access to the server where the electoral processes are executed.

In this sense, the solution proposed in this document is a system that allows managing small types of voting, being:
- Secure. Thanks to the use of advanced cryptographic techniques.
- Decentralized. Avoiding the possibility of match fixing by the organizer.
- Flexible. Allowing any type of voting.
- Transparent. Allowing constant monitoring of the vote.
- Economical. In comparison to the infrastructure necessary to carry out a poll-site vote or by mail.

The implementation of this solution would not be excessively expensive, and its open character would allow any interested party (political formation, association, journalist, etc.) add their own node to the network, forming part of the system as well auditing it.

As any solution, it has some limitations to take into account, being possibly the most important the low speed at which it can process the votes, but knowing these limitations it can be optimized.

Therefore it is a valid option for the problem that was desired to solve and a functional example of blockchain technology in a voting environment.

Chapter 7

Appendices

Github

The Github link to the project repository can be found below:
`https://github.com/cadel560x/AppliedProject.git`

Bibliography

[1] S. Guasch, "Practical considerations for the implementation of end-to-end verifiable eVoting" by *International eVoting PhD workshop* , Vienna, 2011

[2] C. Gentry, "A FULLY HOMOMORPHIC ENCRYPTION SCHEME," 2009

[3] AJ Menezes, PC v. Oorschot and SA Vanstone, «Chapter 8.4 ElGamal public-key encryption, "by *Handbook of Applied Cryptography*, CRC Press.

[4] SMS Goldwasser, "Probabilistic encryption," from *Journal of Computer and System Sciences 28* .

[5] L. Fousse, P. Lafourcade and M. Alnuaimi, of *Benaloh's Dense Probabilistic Encryption Revisited* , 2011.

[6] B. Adida, "Homomorphic Public-Key Encryption," in *Advances in Cryptographic Voting Systems* , MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 2006

[7] D. Chaum, "Untraceable Electronic Mail, Return Addresses and Digital Pseudonyms, »from *Communications of the ACM - CACM '81* , 1981.

[8] P. Golle, M. Jakobsson, A. Juels and P. Syverson, "Universal Re-encryption for Mixnets, »by *Topics in Cryptology - CT-RSA 2004* , Springer Berlin Heidelberg, 2004.

[9] D. Wikström, "A Sender Verifiable Mix-Net and a New Proof of a Shuffle, »by *Advances in Cryptology - ASIACRYPT 2005* , Springer-Verlag Berlin Heidelberg, 2005.

[10] Sanobis, "Topologies of P2P Networks (Peer To Peer)," [Online]. Available: http://www.redeszone.net/redes/topologias-de-redes-p2p-peer-to-peer /.

[11] P. Franco, "Public Key Cryptography," by *Understanding Bitcoin: Cryptography, Engineering and Economics* , Wiley, 2014.

[12] National Institute of Standards and Technology, «Digital Signature Standard (DSS), »July 2013 [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[13] MV Hankerson, Guide to Elliptic Curve Cryptography, Springer-Verlag, 2004

[14] «Technical background of version 1 Bitcoin addresses,» [Online]. Available: https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses

[15] G. Becker, "Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis, »2008.

[16] P. Franco, "7.4 The blockchain," by *Understanding Bitcoin: Cryptography, Engineering and Economics* , Wiley, 2014.

[17] Bitcoin Wiki, «Double spending,» [Online]. Available: https://en.bitcoin.it/wiki/Double-spending.

[18] Blockchain info, «Bitcoin Monetary Statistics,» [Online]. Available: https://blockchain.info/es/stats.

[19] N. Szabo, "Formalizing and Securing Relationships on Public Networks, » *First Monday,* vol. 2, No. 9, 1997.

[20] E. Dawson and D. Donovan, "The breadth of Shamir's secret-sharing scheme, » *Computers & Security,* nº 13, 1994.

[21] D. Chaum, "Blind signatures for untraceable payments,"*Advances in Cryptology Proceedings of Crypto,* No. 82, 1982.

[22] MG,. IM Christina Garman, «Decentralized Anonymous Credentials, »The Johns Hopkins University Department of Computer Science, 2013