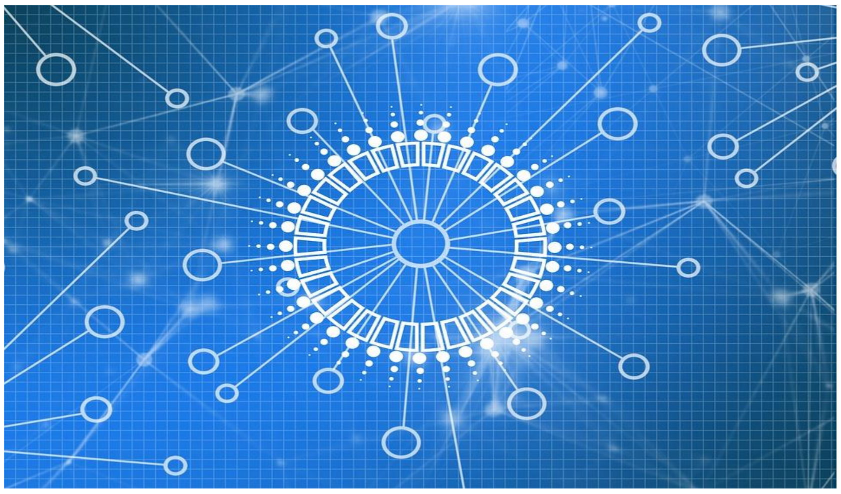
University of Piraeus

Master’s degree in Cyber Security

thesis

‘Creation of educational material on Blockchain environment’



Student: Sideris Angelos-Marios

Supervising Professor: Christos Xenakis

Athens 2023

Thanks

First of all, I would like to thank my family and friends for their support and patience during the creation of the project. I also thank the members of the University of Piraeus for the knowledge and experiences they offered me during my years of study in the Department of Digital Systems. Finally, I would like to thank in particular the supervising professor Mr. Christos Xenakis for assigning the dissertation as well as for the guidance and useful advice he offered me for the elaboration of the thesis.

Contents

Thanks 3
Abbreviations 6
Keywords 6
Introduction 7
Chapter 1: Blockchain 8
1.1 Definition and operation of Blockchain 8
1.2 Blockchain structure and types 9
1.2.1 Blockchain Architecture 9
1.2.2 Blockchain Types 12
1.3 Blockchain use cases 13
1.3.1 General uses of Blockchain 13
1.3.2 Internet of things 13
1.4 Blockchain evaluation 14
Chapter 2 Smart contract 15
2.1 Structure and operation of smart contract 16
2.2 Comparison of smart contracts with typical contracts 18
2.3 Evaluation of smart contracts 19
Chapter 3 Ethereum 22
3.1 Ethereum features 23
3.2 Structure of Ethereum 23
3.3 Platform operation 28
Chapter 4: Security in Blockchain 30
4.1 Principles of Blockchain 30
4.2 Security mechanisms and methods 31
4.3 Personal data security 34
4.4 Blockchain security vulnerabilities 35
4.5 Security in Ethereum 37
4.6 Access control 39
4.6.1 User-Managed Access (UMA) 40
Chapter 5: Smart Contract Lab Environment 43
5.1 Tools 43
5.1 Smart contract Testing example 45
5.2 Development AccessControl smart contract 54

5.2.1 Partial analysis of the smart contract 55
5.2.2 AccessControl smart contract results 59
5.3 Remix Tool toturial 63
5.3.1 Smart contract panel 63
5.3.2 Compiler Smart contract 65
5.3.3 Deploy Smart contract 66
Conclusion 69
Bibliographical references 69

Abbreviations

P-2-P:	Peer-to-peer
PoW:	Proof of work
PoS:	Proof of stake
EVM:	Ethereum virtual machine
UMA:	User-managed access

Keywords

Blockchain, Ethereum, smart contract, Solidity, access control, UMA

Introduction

The goal of the thesis is creating the basic environment for understanding Blockchain technology, especially the Ethereum network. The use of methods that guarantee the security of personal data is essential when managing data on distributed ledger. Also, this thesis aims to familiarize the reader with the innovative technology of Blockchain and the benefits offered by its adoption in the field of security. Initially will be refer to the definition and basic concepts of technology as well as its operation. A special analysis will be made in the concept of smart contracts and the role they take part in the operation of technology and in the implementation of applications. The Ethereum as an open-source platform based on Blockchain technology will be used to design and implement of the solution. It will take advantage of the possibility offered by the platform for the writing and operation of smart contracts as the user desires. The main object of study in this project is the security of Blockchain technology and how it works. The benefits of a Blockchain-based framework create a reliable and transparent data storage and mangling environment. Security will be assessed based on the personal data of users stored within the network. In particular, it will be examined in the light of medical data of patients who are sensitive personal data and require a high level of security. It will also present the logic of data access controls through the UMA mechanism that gives the user the choice of who has access to the data they hold. Based on the above criteria, a Remix environment it is created with smart contracts of the basic operation of solidity program language.

Chapter 1: Blockchain

Lately, the term Blockchain has been constantly appearing in our daily lives, although it has established as a technology using cryptocurrencies, its practice is aimed at a wider range of applications. The benefits of this new technology as well as its ability to be applied to a variety of applications force some to talk about an innovative idea like the internet. It is also a given that it will change our daily lives in terms of organizing and transferring data and information.

1.1 Definition and operation of Blockchain

A Blockchain is a distributed database of files or digital events that have been executed and shared between users inside a network. Typically, we can define technology as a public online ledger (block) that stores information and data in order to create chains between interacting blocks. Blockchain is accessible to all in encrypted form to ensure user anonymity. No human intervention is needed for its operation. The information stored on it must be verified by the majority of users. Once information is stored, it cannot be deleted or modified without the users' knowledge. The goal of the technology is to automate actions in the digital world but also to eliminate intermediaries in electronic transactions between client and server.

The way a Blockchain works is reminiscent of the way a data and information ledger work, but it differs from the conventional means used. For example, we are accustomed to having a competent authority in a database, which users trust, manages the data and distributes them accordingly to the users. In contrast to Blockchain technology, the nodes are responsible, i.e., the users of the network. Users who have installed the software at the same time update the ledger for the changes so that always there is the same state and data in the network.

To make it easier to understand how it works, we will cite an example of a transaction with conventional means, a bank without the use of the internet, and compare it with Blockchain technology:

Suppose you want to transfer an amount of money to an account in a bank outside the city where you are. You will first need to go to the bank's branch and ask for the transfer. The employee should check the account that the balance is sufficient to transfer the amount and then deduct the amount from your account and add it to the other.

The problem with this scenario is that users have to trust the banks for their transactions. The bank will keep an amount to become the intermediary in this transaction. But most importantly, we provide our data and it keeps a transaction diary that we sent an amount to someone and when it was done. The question is why trust our data to third parties is there another solution?

Let's compare the same transaction with the use of Blockchain between two internet users. Again, there are some similar steps in the transaction as both parties know and agree with the transaction as well as the balance of the account that wants to make the transfer must be sufficient. If these criteria are met then a smart contract is created between the two users (nodes) and the money is transferred almost automatically and without additional fees. Finally, all the nodes of the network are informed about the transaction and after the data are checked, they are saved.

It is observed that the transfer of money through Blockchain is much cheaper and faster, but more importantly it does not need the intervention and supervision of third parties. The two nodes communicate with each other and make the transaction without a third-party involvement. So, the transaction is anonymous and we do not trust our data to anyone without that being necessary.

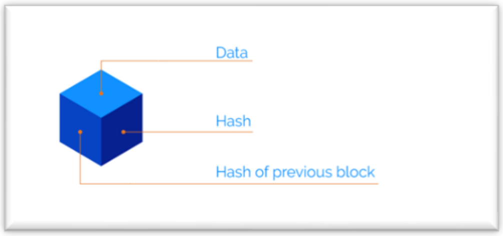
1.2 Blockchain structure and types

First of all, let's mention the basic elements that make up Blockchain technology: block, chain, digital signature, and P-2-P network. The following will be briefly analyzed one by one.

1.2.1 Blockchain Architecture

Block

Blockchain technology consists of blocks that interact with each other to form chains. The blocks contain information in the form of code and encrypted data, the data stored is immutable, cannot be deleted or modified if it is stored in the ledger. Blocks are the structural component of the technology that in themselves are not important but when combined with others they create chains of data.

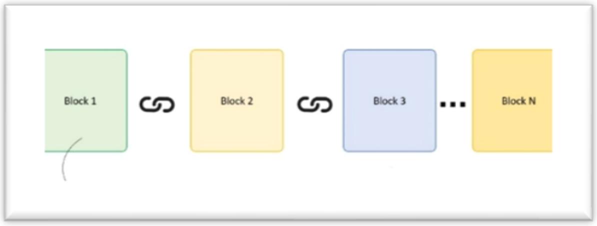


Picture 1.1 Structure of a block in Blockchain.

Each block of the chain has data and the address of the previous block to ensure that the network state is the same at all times. Since each block has the key to the previous block, it ensures that no other blocks are entered between them, thus ensuring the security of the data and the sequence of the chain at all times. The key to each block works like a digital fingerprint and is unique and cannot be forged. The blocks are defined by a Hash created by an encryption algorithm. This Hash exists in the header which has elements for the block such as size, time stamp and the address of the previous block. A block has only one parent block, previous block, but it can have many child blocks if a change is made to the parent block should change the children as well. Finally, in order to add a new block, it must be accepted by all the blocks of the network, this ensures the security and transparency of Blockchain.

Chain

Chain is the connection of the blocks and mainly the way they are sorted in a Blockchain network. Chain is a sorted list of back-linked blocks since each block in Hash refers to the previous one. With Hash, which is created by the encryption algorithm, the blocks are joined together. The algorithm functionality creates the digital fingerprint that cannot be modified or encrypted.

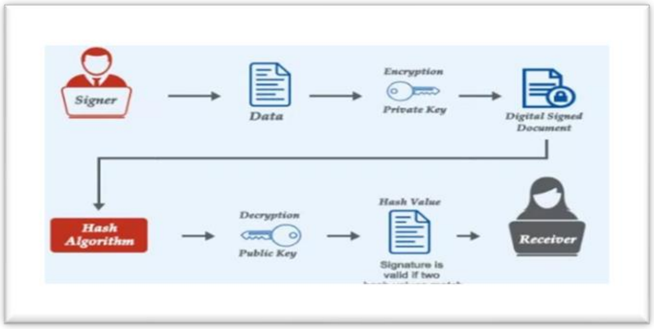


Picture 1.2 A N-block chain in the network.

The data is stored on the network and use pointers and linked directories to create the chain. directories store addresses in a form of variables that indicate the location of another variable. Connected catalogs are a series of blocks that are linked to them by pointers, generally the format of the network is linked catalogs that show each other back using hash.

Digital signature

The operation of the signature as we know it does not differ even in Blockchain technology with which a contract or a transaction on the network is sealed. Digital signature is required to check and validate a transaction. The signature requires 2 keys, one public, visible to all and used to verify the data, and one private, which only the user knows and is used to create the signature.



Picture 1.3 Digital signature validation process.

The following steps must be taken to validate the contract.

- A Hash is first created by user A with the contract data using an encryption algorithm.
- The digital signature is then created after encrypting the Hash value using the private key.
- Finally, the user sends to the user B who is for the contract the digital signature as well as the initial data of the contract.
- B user in sequence will decrypt the digital signature using the public key of user A to create a Hash.
- If the two Hash are the same then the transaction is considered valid.

Peer-to-Peer Network

Blockchain technology users interact through a decentralized P-2-P network and each is represented by a node in it. In these networks, users all communicate with each other at the same time without a central administration. When a transaction or data exchange takes place between two nodes, the data is distributed to all users of the network. This way you control the validity of the data without the need for third party intervention. Finally, using a P-2-P network if a node temporarily stops working does not affect other users who continue to communicate with each other, unlike conventional networks where if the main server crashes, access to the data stored there is unreachable.

In most network models there is a central administration that approves the trust and integrity of the data and information distributed. But Blockchain technology uses P-2-P networks that do not include a central admin. There must be trust between users for the integrity of the transaction data. To achieve this, some algorithms have been created to validate transactions on the network. Typically, we mention the Pow and PoS algorithms that aim to secure the data and prevent attacks on the network.

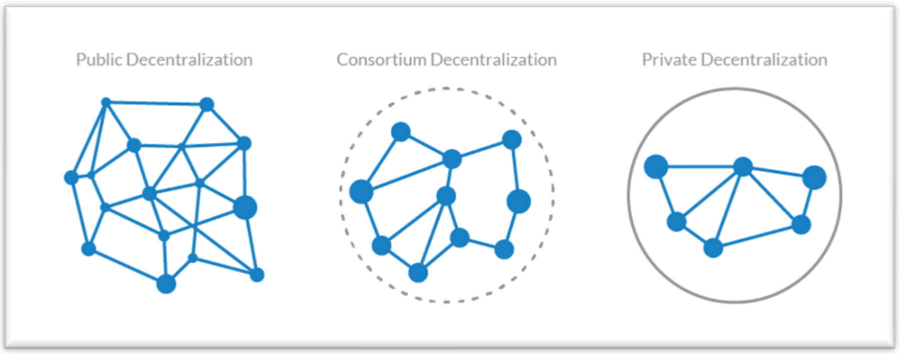
1.2.2 Blockchain Types

Blockchains are divided into three main types: public, private, and permissioned. The differences are mainly related to data and how open it is for users.

- Public: In public Blockchain anyone can read the data and write on the network although some may limit the ability to read or write only. Anyone can send data to a public Blockchain as long as their validity is checked and accepted by all network users. It provides security through encryption algorithms; they are open and everyone can take part in them. They are very large in number of participants but at the same time this makes them much slower than other types of Blockchain. Since they do not have a central administration and do not control their users joining, they allow the anonymity between users to be maintained. It is understood that it requires a lot of computing power to achieve synchronization and maintenance of the registry. A simple example of a public Blockchain is the Ethereum platform.

- Permissioned: It is a Blockchain that the consent process is approved by a pre-determined node team. Reading and writing rights may be public or restricted to network participants. This allows members of the public to ask a limited number of questions and receive encrypted Blockchain sections in response, which allows them to be faster than public ones.

- Private: In private Blockchain all participants are known in advance and are considered reliable by the network. Only a central authority has the right to write to this network, controls and selects the data to be stored. Reading rights are usually public or limited, but this depends on the nature of the network. It is the fastest type of all and may not take any time to process. They have low cost of creation and maintenance, unlimited user capacity and can be created in a short time.



Picture 1.4 The different types of Blockchain.

It is understood that there is no better type of the three but based on these characteristics, whether positive or negative, anyone can choose the Blockchain type that is suitable for each problem they have.

1.3 Blockchain use cases

As mentioned at the beginning, the technology was created with the main focus on the creation of cryptocurrencies and transactions with them. In the process, however, it has evolved into a technology that is increasingly being used in more and more areas, from online media transactions to healthcare technology. Some uses of the technology will be summarized below to understand its usefulness.

1.3.1 General uses of Blockchain

The purpose of the technology as mentioned above is to automate processes and reduce bureaucracy this is achieved through the use of smart contracts and some common examples are mentioned.

Digital identity

Multiple services and institutions require different data and information from users depending on the procedure’s requirements. The user must have them with him either digitally or in printed form and their verification by the services is required. But by creating a smart contract, the data is collected in an online digital document that is accessible from anywhere. Once saved, they have already been verified by all users and cannot be deleted or modified unilaterally. This allows the user to access their data whenever they want without having to verify it and keep the user anonymous.

Automatic transactions

Whether we are talking about banking transactions or a public service, there is a problem in processing them. The bank enters as an intermediary, holds a commission for the transaction but has access and manages the user's data. Public services, respectively, require a number of different documents and time to complete the transaction. With the use of a smart contract, transactions can be made almost automatically, without requiring verification of user data, no extra cost and more importantly without third party intervention. Due to the transparency of smart contracts, anonymity is ensured and it becomes almost impossible to deceive the user.

Copyright

It is common for copyright infringement on the internet and its creators are trying to find a fair solution. A smart contract can ensure the creation of a transparent network registry that holds the data of anyone who uses a copyrighted file and informs its creator.

1.3.2 Internet of things

An important use of Blockchain technology is related to the concept of Internet of things it refers to objects of our daily life from production machines to devices that we wear with built-in sensors. The idea behind this is to continuously collect data in order to improve the user's daily life. It indicates to countless devices around the world that collect data and communicate with each other whether online or not. Defining the exact function of the internet of things is difficult because we are talking about dissimilar devices in different areas that collect different data, but the logic remains the collection of data to improve everyday life. Always targeting the development of applications that facilitate the everyday life of users let us mention a few examples in order to be perceived.

Originally the concept of the smart home that always seemed an image of the future can now be a reality. A series of smart devices that communicate with each other and exchange information in order to create an automated home for the owner. Communication is done through the use of smart contracts and they can create a very familiar environment for the user.

Even in the market of products through a smart contract we can know the whole past of food products to technology products. Many devices store data about the product from when it was created, how it was packaged, and the route it took until it reached the store or the user's hands. In this way, the consumer does not need to simply trust the seller to know everything about the product, thus reducing the risk of fraud.

Even in the health sector, with the use of the internet of things and a smart contract, we can literally save lives. With devices worn by the patient who collects data on the patient's health such as his blood pressure or according to his health problem. If this information is sent to a nurse or doctor, it will be easy to see if there is a problem and thus prevent a crisis in the patient's health.

1.4 Blockchain evaluation

Like any new technology there is a skepticism from the user because it is difficult for user to change a technology he is using. The purpose of any new technology is to completely or partially improve the existing system. As we mentioned, an important change is that it makes the need for third parties in transactions outdated, it is important to mention that the term transaction does not only mean financial transactions. Blockchain technology transactions are not only about money transfer but also about creating smart contracts for everyday problems. The advantages of the technology have to do with a variety of improvements and will be analyzed below.

- Decentralized: It is one of the most important advantages of technology because it allows two nodes, two users, to make a transaction without the supervision or mediation of third parties.
- Transparent: Changes to Blockchain data are visible to all and must be approved by all users of the network, the data after being verified and stored cannot be deleted or modified.
- High quality data: The data in a Blockchain is complete, accurate and available to everyone.

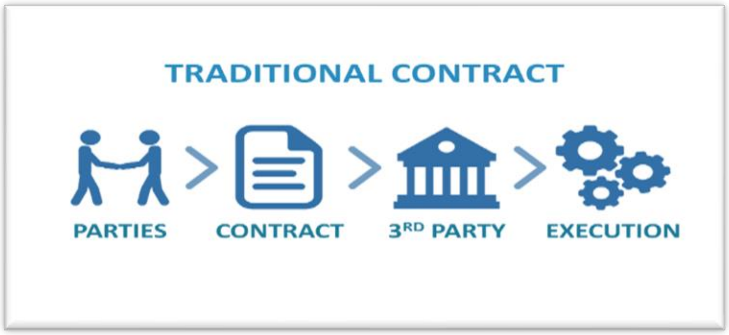
- Authorized users: Users have access to the data at all times as well as have the ability to manage their data and transactions on the network.
- Reliability: Due to the decentralized networks used in technology, there is no central management point that in the event of an attack would be catastrophic, so it could well cope with a possible attack.
- Cost Reduction: Since users are responsible for running Blockchain, third parties do not need to intervene and this results in a reduction in operating costs.
- Accessibility and ease of use: Accessing a Blockchain is easy because you just need Internet access. Also, its use is understandable and easy to teach everyone.
- Fraud Resistance: Since network data is visible everywhere and cannot be deleted or modified by a single user then all its nodes must be compromised in order for the system to be compromised.

A technology cannot always be useful and only have advantages, it is natural to have areas that need improvement. The purpose of the technology is to automate online transactions as well as to secure data on the internet using encryption. As expected, with any new technology users do not trust it enough to store their data in a Blockchain network. There is a prejudice against it and they prefer the old technologies that know their function rather than something new and foreign. Also, an adaption on a decentralized transaction system requires the consent of both users and service providers for a practical solution. Finally, although Blockchain offers cost reduction and time savings, the relatively high initial capital required is a subversive factor.

Chapter 2 Smart contract

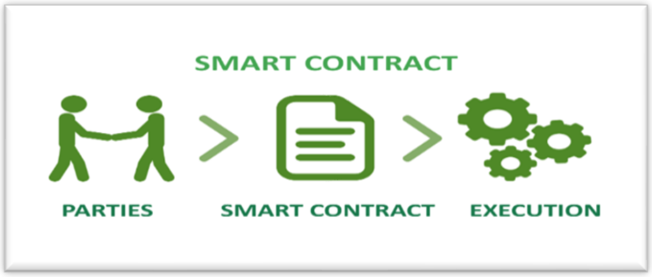
A smart contract can be defined as an electronic computer code that facilitates or executes a transaction using Blockchain technology. It is automated and recorded in a programming language as a set of instructions. Smart contract is probably the most important developing tool in a Blockchain environment.

First let's look at the logic of a non-digital contract, it is usually a written agreement between at least two parties and includes commitments for actions between those parties. The involved parties rely on each other to comply with this agreement and to comply with the defined conditions. The contract is usually validated by the signature of the parties presented to an intermediary who guarantees its execution.



Picture 2.1 Typical contract validation process.

Smart contracts have the same logic as non-digital contracts, but they eliminate the need for trust between the parties and they are validated by a digital signature. It is essentially a piece of code stored on the network and executes or imposes an agreement between two or more nodes on a Blockchain network. Once a code is saved, it cannot be modified or deleted by third parties and is activated when the parties wish or when certain pre-determined conditions are met.



Picture 2.2 Smart contract validation process.

The Blockchain environment is ideal for storing and executing smart contracts because of the security and stability it offers. Smart contracts are encrypted and distributed through a network ledger, making it impossible to lose information stored in blocks. Also, Blockchain environments introduce smart contracts to the concept of flexibility, they offer the ability to store any type of data in the nodes of the network. Smart contracts operating in a Blockchain environment serve organizations and users to provide more secure and efficient online transactions while reducing operating costs.

2.1 Structure and operation of smart contract

Smart contracts to be useful to users for transactions in a Blockchain environment must be determined by certain features. As mentioned, contracts are a piece of code that is enforced when the terms of the contract are met.

When writing, smart contracts required to meet certain conditions in order to be accepted by network users. The subject of the contract must first be clarified, in order to have access to goods and services that it can carry when executed as specified. It is also necessary to include the digital signatures of the involved users; the users sign the contract with their private keys. Finally, the terms of the contract must be understood by the involving parties and agreed in advance, the terms being a precise sequence of operations to be performed.

The process of recording the contract in the ledger is simple but includes peculiarities when storing it. The smart contract is listed in Blockchain as a piece of code, the parties involved may remain anonymous but the contract and its terms are established in the public ledger. Activation facts, such as expiration or receipt of data or money, as well as the conditions for its execution, come into force and are accessible to all. This makes it easier for everyone to interpret the contracts easily but also to execute them automatically as described in the code. The data of the contract are visible in the public ledger so that the regulators control the status of the contracts while the parties involved maintain their anonymity. Smart contracts can verify a transaction by interpreting the data. Each contract is executed independently at a network node to ensure that it is executed as it is defined, this allows its creators not to worry about whether the contract is executed correctly or not because it is guaranteed.

Smart contracts although they are called contracts, are not exactly contracts they are a kind of unilateral agreement that is fulfilled when certain terms of a computer program are met. In smart contracts the conditions are coded in programming language and converted to machine language. The peculiarities depending on the problem that needs to be solved presupposes a different way of writing a smart contract.

There are four ways to write a smart contract:

- Natural language with some functions encoded in digital form. This is a common way of writing a paper contract in the form of a comprehensible language with simple actions such as payment written in code form.

- Natural language using coded performance mechanisms. Also, this form is usually written on paper but the coded functions are not limited to payment. Contract operating commitments are automated to run on a digital platform autonomously.

- Contract written in automated form but completed with a document in natural language. The conditions and procedures for execution are written in code but a diagram written in natural language is in order.

- Contract written entirely in code. The procedures of the contract are fully automated and are performed without the need for a supplement in natural language or human intervention.

The last way to capture the contract is for those who are more interested in the developers because it is faster in performing the functions and offers autonomy to the contract.

After analyzing the conditions and the different ways of writing a smart contract, let's talk about how a contract works in a Blockchain. The way smart contracts work can be compared to the operation of a vending machine. Smart contracts receive an entry, usually an amount of cryptocurrencies, on the machine or in the case of a smart contract in the register of bonds. When the entry satisfies the terms of the contract code, the procedures agreed upon by the parties are automatically applied. The commitments written in the code are in the form of "when case 1 applies, perform the procedures". For example, "If person A executes event 1 then amount of money from person B will be transferred to person A". Smart contracts after they are written are stored in a ledger and allow users to transfer assets or data to each other whenever they wish. The way they are stored ensures that the data cannot be modified or corrupted after being registered. Finally, the encryption methods used in the process of writing the contract guarantee anonymity between users.

The easiest way to understand how a smart contract works is an example of a two-way transaction. Suppose the scenario is that user A wants to send an asset to user B who in turn will send an amount of money to user A as payment. User A wants to make sure that he receives the payment when his property is delivered, respectively user B must make sure that he receives the object in time after making the payment. Both users need an external factor that will ensure that the transaction will be executed as specified by both parties. In addition, the transaction must be carried out when certain conditions are met, this is achieved through the use of smart contracts. The transaction process using a smart contract ensures that user A will not send the item before it is paid also user B will not send the amount before receiving the item. The way it is executed ensures that both users will be satisfied and that the transaction will be executed as defined from the beginning. The transfers of the object and the money will take place at the same time when the terms of the contract are met. The contract will receive the amount of user B and will keep it as a guarantee until the item can be transferred. Only then will the amount be transferred to the user A. The example of the transaction is simple to understand how contracts work, usually the functions of smart contracts are more complex and are not limited to exchanging material and money.

2.2 Comparison of smart contracts with typical contracts

Smart contracts are not really contracting because they have many differences in structure and functionality, but they are named so because they have the same ultimate purpose. Initially, smart contracts are written in code form, while standard contracts are written in a natural language like English, which allows them to be understood in real terms. Smart contracts are executed by a computer that decodes software, contracts executed by humans using special terms and legal frameworks. The smart contract code is executed as defined without the intervention of a human factor or the possibility of modification, in contracts the parties may overcome problems that may arise by reevaluate the terms of the contract. If the coder of a smart contract makes errors while developing then the code may not be executed or executed incorrectly contrary to the contracts the errors may be eliminated if the parties agree to modify the terms to be adapted to the conditions. Smart contracts are intended to adopt contractual procedures and their binding nature but in digital form

If anyone wonders why users demand the use of binding methods such as contracts or smart contracts for their transactions then the answer is a simple lack of trust. People have always distrusted each other. Lack of trust is usually resolved with the intervention of third parties in transactions such as banks or lawyers. External factors provide users with the trust they need to perform the transaction, but this offer also has the corresponding price. In addition to the financial fees received by intermediaries, they manage user data and records, especially when it comes to online transactions. Transactions usually require a plethora of documents and signatures that greatly increase the execution time of the contract due to bureaucracy. In addition, for the creation of a valid contract, users authenticate the documents, a time-consuming process that requires the approval of a lawyer or notary.

Smart contracts are the solution to these problems, they are the digital version of contracts. Procedures such as authentication of data and documents are now done directly with the use of smart contracts as they automate functions, that usually take days, within minutes. Smart contracts provide the user with a level of autonomy, security and speed of transactions as well as offer relationships of trust without third party intervention.

Contracts have the potential to be binding due to the presence of third-party individuals in transactions as well as legal and judicial entities. But this raises the question of whether smart contracts are binding or whether they can replace contracts or whether they can virtually eliminate third party intervention in transactions. With regard to third party intervention, the answer is easy as smart contracts are intended to eliminate third party intervention in transactions apart from occasions that require legal advice when writing the contract. On the other, in order for a contract to be legally binding, certain conditions must be met, but there are also peculiarities that make it difficult for the binding property to be legally binding.

In order for a contract to be legally enforceable and binding, such as typical contracts, it must meet certain conditions during the process of writing and execution. First of all, one of the parties must offer goods or services to the others and the details of the contract must be available for everyone to be reviewed. Transactions made in a digital platform usually use the public key structure, which is a process of securely exchanging messages via the internet. The reason for a transaction between two parties should be clear so that it can be determined by the system, we usually refer to a money exchange but this is not absolute. Perhaps the most important criterion is that the parties are willing to enter into legal relations with each other. In a document in natural language it is easy to determine its terms in a courtroom, but in a contract written in code it is not easily interpreted on the basis of legal framework. A contract that is fully codified and does not contain human intervention is difficult to define as legally binding since the parties cannot be identified by name. In addition, the contracts fall under an uncertain legislative status because no government has legally regulated the sμart contracts or created a framework for their operation. However, a contract in digital form or in a piece of code and not in natural language does not mean that it is invalid, only that users must agree to enter into a collaboration in this way. In essence, the parties must be willing to enter into binding agreements whose terms are absolute and clear to all parties from the start so that the contract does not become unenforceable. Contracts using the data provided to them may automate processes related to their binding capability but also to legal issues.

According to Nick Szabo, who first came up with the term smart contract and their operating process, the ideal smart contract must meet certain rules. First of all, the content and terms of the contract must be verifiable and pre-determined by all parties. The positive and negative consequences of the contract must be enforceable preferably automatically after the contract is ‘signed’. The smart contract should be secure against attacks and strict versus violations of the terms and conditions. Encrypted procedures are used to improve the confidentiality and anonymity of the users of the contract but also for its object. If the criteria mentioned are met then a secure and effective contract is created with regard to its subject matter and binding status

2.3 Evaluation of smart contracts

How a Blockchain environment works is inextricably linked to operating smart contracts, especially when it comes to platforms like Ethereum. Smart contracts are the driving force behind technology, without them it would be a virtual money storage platform. The decisive procedures are performed using smart contracts aimed at the proper functionality of the platform. This makes it normal to have a resemblance between the advantages and disadvantages of technology and smart contracts. Let's evaluate the pros and cons from the point of view of smart contracts.

- Transparency, perhaps the most important advantage of technology and smart contracts, requires that certain terms and conditions be fully met in order for contracts to be executed. The terms and conditions are clear in advance and the contracting users know and agree to them. The contract is executed only when the conditions are met and this eliminates the possibility of problems in the cooperation of users at later stages.

- Processing time, procedures such as data verification require time to complete this is affected by intermediaries involved as well as unnecessary steps performed. Relevant actions using smart contracts are done almost automatically because they are done online.

- Precision, smart contracts are coded in specified steps that will be executed when they are put into operation. The conditions must be met before they take effect because if the conditions are not accurate, a problem can arise.

- Security and efficiency, smart contracts are automated instructions in code form and are the safest option in encrypted data environments. They are the ideal choice as they meet the highest security standards and allow their safe use to automate processes. Their precise operation and the security they provide during the writing and execution process result in their efficient operation at the application level.

- Data storage, smart contracts store data and transaction details in a way that allows the parties involved to access them at all times. Transactions are stored in Blockchain as future files or blocks that are particularly useful in case of ambiguity in the terms of the contract.

- Costs, the use of traditional contracts requires the presence of intermediaries such as lawyers and notaries as well as stationery resources to conclude the agreement. Smart contracts eliminate intermediaries in transactions and do not require documents for the agreement, so money is saved.

- Trust, transparency and security make smart contracts reliable for businesses and users. Since they remove the case of data manipulation while eliminating the human errors during their execution. Following the agreement on the terms, the contract is executed automatically as specified. Smart contracts significantly reduce litigation as the terms are set from the beginning and the parties are committed to abiding them.

- Digital form, since smart contracts are pieces of code written in digital form and not printed, it is very difficult to be corrupted over time.

Smart contracts have some principles and peculiarities in the way they work that make them very useful. Initially, they are autonomous, after their ratification the contracting parties do not need to communicate with each other again smart contract run on their own. They are also performed independently within a network without the need for third parties to intervene in their execution. In other words, they are self-sufficient, they operate with their own resources and once activated, they cannot be modified or deleted by third parties. They are decentralized, they do not have a central authority that operates them, but they can act on many nodes of a Blockchain network. The goal of smart contracts as well as Blockchain technology in general is to automate the processes and eliminate the intermediaries whenever is possible. Smart contracts eliminate a lot of operating costs such as staff required to monitor their execution, this makes them an affordable and advantageous service solution.

Smart contracts are the most important tool in Blockchain technology that guarantee the best operation of the technology. Although they have many advantages in their implementation process and make the most of the capabilities of the technology, there are malfunctions that users should consider. The main disadvantage of smart contracts is the involvement of the human factor in their implementation. People may make mistakes when writing smart contracts, perhaps because of fatigue, the consequences of mistakes can prove disastrous in the operation of contracts. Smart contracts are usually stored in a public Blockchain this makes it impossible to modify or delete them after they are saved in the registry. It is therefore necessary that the terms and conditions of the contract be carefully analyzed. Contracts are executed automatically in a node network when the conditions set by the parties are met. The instructions that will be executed must not contain errors so as not to create a problem in the network. Autonomy of contract execution makes them vulnerable to attack and allows them to operate, if they are attacked, in accordance with the intentions of the attacker. Due to the problems we mentioned it is necessary to hire experienced developers who are expensive in order to ensure proper operation. In the end, it is very important the development process of contracts so that they do not contain errors that will threaten their proper functionality on a network and not to be vulnerable to attacks by malicious users.

Smart contracts are applied and controlled automatically without human intervention, which allows them to use Blockchain technology in full force. At the same time, however, their mode of operation creates special operating conditions that the user must consider before adopting in his daily life. Initially it is a piece of code so it is as useful as the data that it gets as an input. They are absolute in their function and cannot cope with the ambiguities caused by shortcomings in the code during the writing process. They also cannot be adapted to situations in the environment created and performed when the conditions are met without the possibility of modifying them once stored Usually the purpose of contracts is not simply to automate processes so human intervention is valuable and crucial. In conclusion, smart contracts are not useful in all cases, but they are beneficial in situations that require automation of processes and elimination of third-party intervention.

Chapter 3 Ethereum

Picture 3.1 Logo of the Ethereum platform.

Ethereum is a public platform that manages transactions, smart contracts and allows any user to use decentralized applications through Blockchain technology. Like any platform based on Blockchain technology it does not have a central administration but unlike other Blockchain based platforms it is open-source platform used by many users around the world. Ethereum is a software friendly platform for developers, because instead of offering a set of pre-defined functions it offers the user the possibility to create the functions they want. It is practically a programmable Blockchain that allows the creation of decentralized applications and smart contracts that are capable of solving any computer problem. The applications on the platform are of any complexity that can solve from simple to complex procedures related to the operation of the platform. This feature makes them very useful in a variety of scenarios that involve the transfer of cryptocurrencies but are not limited to it. The main difference of the platform is that it allows the creation of smart contracts to the user without restriction. Similar platforms with Ethereum have offered a limited number of commands and tools to create smart contracts for cryptocurrency management. In contrast, the Ethereum platform uses a programming language that allows the creation and execution of applications as defined by their creator.

3.1 Ethereum features

The platform, as mentioned, is decentralized and runs applications based on Blockchain technology. Like any platform based on Blockchain technology, its main purpose is to automate processes and eliminate intermediaries. In traditional systems there is a central authority that determines the operation and guarantees the security of the system. The central authority has possession of the data they manage within the network, coordinates and validates the transactions. For example, a bank has a database of user data and performs the functions it desires. Ethereum, on the other hand, eliminates the concept of central administrator and uses a P-2-P network in which network nodes communicate with each other. The records are distributed to all users at the instantly, and all nodes hold a copy of the transactions that have been made

In addition to the mentioned feature, Ethereum has some features that contribute to the creation of a secure and interoperable environment while at the same time the concept of a central administration is abolished. Initially, the network is decentralized and consists of thousands of nodes (computers) that operate continuously and distribute the data to everyone in the network at the same time. The Ethereum network is public so that everyone can participate and create applications on it or make transactions. Applications are executed on the network automatically without the intervention of a human factor through the communication between the nodes. Data encryption improves their storage and distribution on the network. Finally, the use of cryptocurrencies allows payment within the network and due to the autonomy of the procedures the payments can be made when certain conditions are met such as a specific date.

The above features contribute to the way the platform works which create many benefits compared to conventional media. Based on cryptography, a secure environment is created for data storage and transaction execution. The network and its users are also protected from malicious attacks by users. Transactions and data are validated by thousands of users, this ensures their reliability and reduces the possibility of their alteration. Since there are thousands of network nodes and they are constantly working, they are more efficient, if we consider that there is no central administrator, which if violated, creates a problem, the network is practically unstoppable. Nodes have the ability to communicate directly with each other without the interference of third-party communication models.

3.2 Structure of Ethereum

Ethereum platform consists of several "layers" that participate key actions within the platform to guarantee its proper operation. Initially there are three levels of hardware, software and applications that are responsible for the operation of the platform. Then there is the EVM which is the link between the user and the platform to automate necessary actions. The platform also introduces an original way of storing user data and transactions. Finally, there are the accounts that determine the execution of transactions on the network and are distinguished in external and contracts.

Layers

Ethereum can be characterized as a stack consisting of from top to bottom layers that interact with each other. The main level of the platform that allows others to operate is the level of hardware. Then there is the level of software that deals with smart contracts and enables developers to run them on the network. Finally, there is the level of applications that offers different services for the decentralized use of applications.

The hardware level is the Blockchain that consists of many interconnected computers called nodes. Each node organizes transactions and data and transmits them to the rest of the network. The number of nodes confirms the data and transactions made and stores them in a common register. This allows the creation of a distributed database that can hold a file with the data transferred to the network as well as the transactions validated by the nodes. Transactions can transfer ether, the currency of the platform, or information in the form of code or encrypted data.

Within the network nodes can have multiple uses such as simple nodes, customers or miners. Nodes are responsible for overseeing transactions and data being transferred. Customers connect the nodes to each other to receive the information and data they want about the current state of the network. While miners are responsible for confirming the correctness of transactions and data.

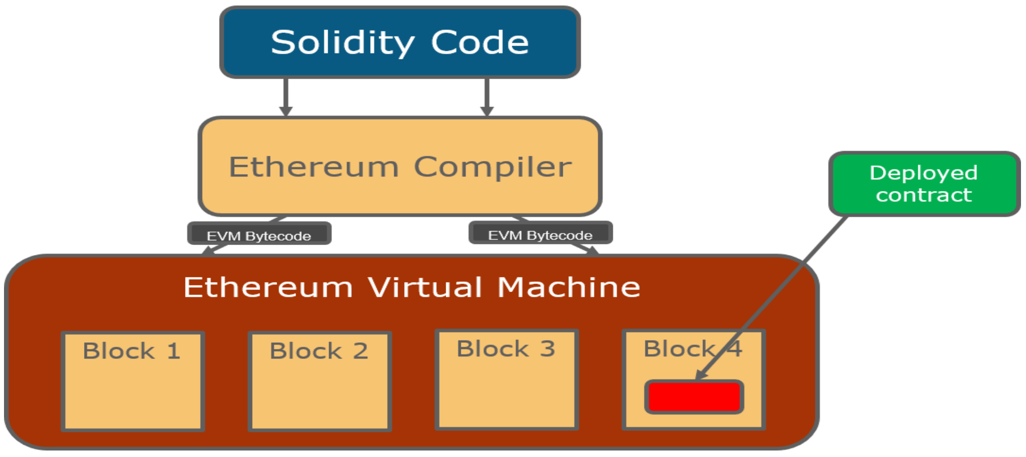
Anyone can join the network and become one of their nodes simply by running software on their computer. Once it becomes part of the network, it can simply offer the computing power it has to speed up the process or make transactions with other nodes. Whichever of the two options you decide will have access to the network data and information.

The level of software in Ethereum refers to the code that is executed and allows the transaction of any type of ‘value’ such as money, data, assets and information. The programming language used is called Solidity which is similar in spelling and operation to JavaScript. Users can create smart contracts that can implement the basic logic of transactions. Solidity is capable of performing any process and is available to anyone wishing to engage in smart contracts. In the Ethereum network, all the codes of the smart contracts are public asset of the software, so everyone can watch how the programs are executed.

The last level refers to applications running on the Ethereum network, application layer corresponding with hardware and software layers acts as a decentralized global network. The platform runs applications without third party intervention and once they start working, they cannot be stopped. The transparent nature of the platform encourages all developers to work together to create a strong community. The code is public, so it is controlled by all network users to improve the functionality of the platform.

Ethereum Virtual Machine

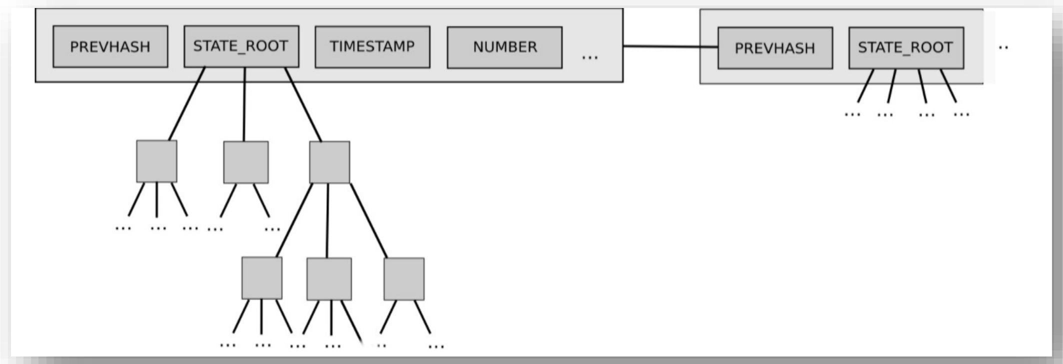
In addition to the three levels of Ethereum, we owe to mention the mechanisms that operate in the background so that applications run continuously on the network. Older technologies based on Blockchain technology did not allow the user to create applications from scratch or offer a limited number of predefined commands. This was the main reason for creating this platform, a programmable Blockchain that offers the user the ability to run applications regardless of complexity and computing power required. It is understood that a mechanism that acts as a link between the user and the network in which the applications are running is required to reduce the complexity of the actions. The mechanism used by Ethereum is EVM. EVM essentially creates a level of "logical subtraction" that performs actions related to how the network works. Its functions have to do with the conversion of the code and the actions that the user wishes to perform in a form that is understandable by the network and vice versa. The network cannot understand the code in a programming language or messages in a standard physical language, so the EVM converts them into a machine language. This allows the user to create applications in any language he wishes or is familiar to, without calculating whether it is compatible with the network. This allows the creation of a common network for every application and not multiple networks depending on the writing and execution environment. Finally, EVM is responsible for converting data so that it can be stored in the Blockchain.



Picture 3.2 Communicate between platform layers.

Architectural data storage

The platform stores and manages data related to transactions and smart contracts, the data usually relates to account balances, addresses and transactions performed by the account. How to store data as in any platform based on Blockchain contains a chain of blocks. Initially the chain contains only one block which is called parent, it is essentially the starting point of the chain on which all activities are based. All activities performed such as transactions, contract execution and data mining modify the chain. The transactions and actions of the users in the network after they are verified are stored in a new block. The data generally stored on the network is of two types, permanent and temporary, determined by their ability to be modified. A classic example of permanent data is the recording of a transaction that, once stored on the network, cannot be modified or deleted. Respectively, a temporary fact is the balance of an account that can be constantly changed depending on the transactions made by the accounts. The platform uses data structures to store permanent and temporary data, it is understood that the two types of data are stored separately. To store the data, the EVM uses fragmentation methods to secure the data and then stores it in tree form. Permanent data such as transactions are stored on the network after first being validated, while temporary data such as account balances are stored and updated constantly. The data is converted into smaller parts in a format that can be stored on the network. The mechanisms succeed with simple machine commands (such as push, pop, add and others) to create a more simplistic form. The different nature and usefulness of the data requires their management and storage in three different tree forms.



Picture 3.3 Tree form of data storage within Ethereum.

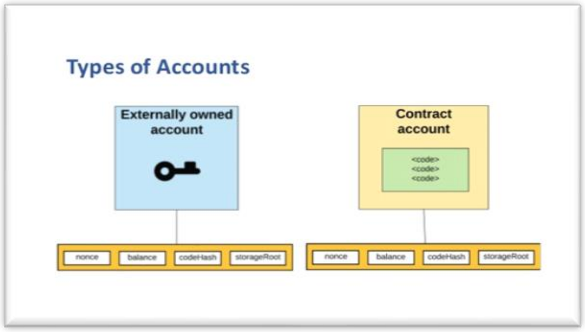
Tree forms are state trie, storage trie and transaction trie (trading tree) which store different types of data and present peculiarities in terms of their operation. State trie is unique to each Blockchain network, is constantly updated and includes a key for each account consisting of 160 bits. The storage trie stores transaction data for each node, every node of the network has one. Each network account contains a key containing a 256 bits address. The root node for each storage trie is stored in the state trie. A transaction trie is included in each block of the network and contains the transactions performed by the block that is stored and extends the corresponding transaction trie. Finally, the important data of the users and the contracts are not stored directly in the network but only a hash that is created.

Accounts

Perhaps the most important building component of the Ethereum network is the accounts that are entities within the platform that interact with each other and communicate via mean. Network accounts are characterized by a byte sequence that is their address. There are two types of accounts on the network that are related to whether they are managed by a user or are autonomous.

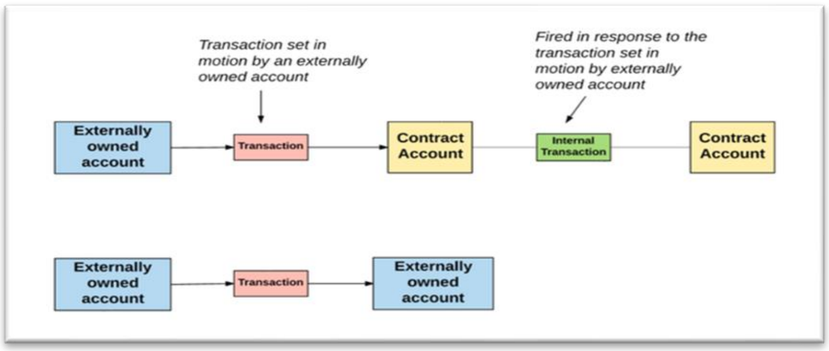
Initially there are external accounts controlled by the user via private keys. In external accounts there is no code related to their operation and they have the ability to communicate with other external accounts but also to create contracts. Also, with the use of private keys, they can sign transactions that take place within the network. These accounts contain a nonce that measures the number of transactions associated with the respective account address. It also contains the balance of the account, its address and a hash associated with the tree where the account data is stored.

The second type of account is contracts that are not related to a user but are autonomous and governed by the account code. Contract accounts cannot create conditions for performing new transactions on their own, such as external accounts. They execute the code contained either in the account or in other contracts within the network depending on the commands they will receive. They receive messages either from external accounts or other contract accounts and interact with each other. The contract accounts contain a balance as well as a hash related to the tree that stores their data. The nonce contains the number of transactions that interact with the account. Unlike external contract accounts, contract accounts contain a codehash that is associated with the code they contain and the actions that EVM performs when the account receives or sends a message.



Picture 3.4 The two types of external and contract platform account.

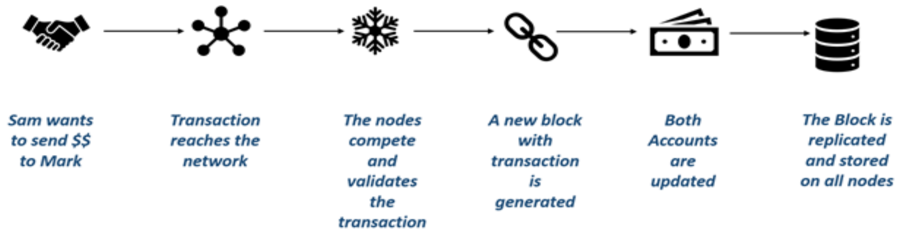
Accounts can communicate with each other as mentioned above to perform actions within the network. The communication between two external accounts involves the execution of a transaction and the transfer of an asset or data. When an external account communicates with a contract account it gives him the signal to execute the contract account code or to communicate with another account. Respectively, when communicating two contract accounts, it is because there is an order in the first to execute the second's code.



Picture 3.5 Procedure for calling and executing transactions within the platform.

3.3 Platform operation

Before referring to the smart contracts of the platform and the way they are executed, we will refer to the transactions within the network. Transactions are the way accounts and contracts communicate with each other in order to execute the commands contained in the message. Network transactions from their creation until they are available for use go through many stages. Suppose two accounts communicating with each other will send a message will be send from one account to another containing both addresses and transaction's commands. The transaction is not completed immediately nor is it considered valid by the network, therefore is not yet stored on it. It is initially stored temporarily in a block like all transactions awaiting approval of their data. The network's miners then compete over who will be the first to approve the transaction records. This process is called PoS Once the data is approved, a new block is created that stores the transaction and is sent to the remaining nodes of the network to update their status. The transaction is then available for use by all network nodes. Finally, the two contracting parties execute the transaction and update their data accordingly and store them in the respective blocks.


Sam wants to send $$ to Mark
Transaction reaches the network
The nodes compete and validates the transaction
A new block with transaction is generated
Both Accounts are updated
The Block is replicated and stored on all nodes

Picture 3.6 The life cycle of a network transaction from creation to execution.

In similar platforms such as Bitcoin, transactions and consequently smart contracts have very limited functions that can perform usually only money transfer. This is related to the way they are built as they contained fields that only concerned the sender and recipient address and the amount being transferred. In Ethereum, on the other hand, there is another field of data that can contain any kind of message. This increases the number of different types of network transactions.

Depending on the content of the transaction fields, there are three different types of transactions that take place on the network. Initially there is a transaction where money transfer is the same structure as in Bitcoin it contains recipient and sender's addresses and the amount transferred to, the data field remains blank. Then there is the transaction to create a contract that contains the sender's address (the one who creates the contract) while the recipient's address is blank, the data field contains the contract code in machine language. Finally, there is the transaction that calls a contract, there is the sender address while the recipient address is the address of the contract in the network, the data field contains the name and the parameters which the contract is called. The last two transactions in the amount field may contain money that was sent to the contracts but is not necessary. In this way, the accounts within the network are communicated so that the functions of the network can be performed.

The term smart contract is now known since it was mentioned in a previous chapter but Ethereum platform innovated in the area. It offers the user the ability to create smart contracts that are autonomous using programming languages. Contracts on the platform inherit that we have mentioned above about properties and features, but the way it is written and executed is different. The process of writing the contract by the user until it is stored on the network has three steps. Initially the contract is written in a high-level programming language usually for the platform is Solidity. After the contract is written by the user then it is converted from EVM to bytecode so that it is understandable and executable by the system. Finally, it is stored on a node in the network and "waits" until an account sends it a message to perform its operations.

Mainly the state of the network is inactive, the accounts remain the same without changing their status and data. But every user has the ability to trigger actions by making a transaction. If the transaction concerns two external accounts, it was stated that it is limited to money exchange, so it will be executed and the account balances will be renewed. If it refers to the call of a contract by an external account, then the situation changes to active and the code concerning the respective contract is automatically executed. The contract accepts a message with some parameters and executes them automatically or it may communicate with other contracts to execute commands. As soon as the contract performs its actions, it stops working until an account calls it back.

When a contract receives a message it automatically executes its code and the purpose for which it was created, the contracts serve four main purposes. They contain stored data related to applications or are useful for network operation. They work as a kind of account with multiple users managing messages and data under specific conditions. It manages transactions that are enforceable and offers security in user agreements. Finally, it can expand the capabilities of another contract and offer operating tools such as language libraries.

In the latter case, the contracts interact with the transmission of alternate messages, the messages contain the addresses of the two contracts and usually an amount related to the execution of the functions. When a contract receives a message, it can send data to the sender who in turn uses the data directly, like the operation of functions of a programming language. Contracts communicate with others because of the different purposes that each fulfills within the network, essentially complementing each other's weaknesses. Even at the level of code commands, contracts use each other so that the commands for operations are not written in each contract separately.

Contracts on the Ethereum platform can perform functions regardless of cost, but it is understood that this is not done for free. Smart contracts running on a network require gas to "fuel" the platform. It is a type of payment for executing code commands as well as actions required to store data. Each contract contains a maximum limit of resources that can be allocated for its operation, the limit cannot be exceeded so that there is a check on how much the contract will cost and how many times it will be executed. The cost of each action performed by the contract is predetermined by the nodes of the network that are miners. Once a contract has been stored on the network it can theoretically be called by everyone, although there are usually security measures so that it is not done unnecessarily. Like the way of writing, the calling of contracts, it is reminiscent of the way an object-oriented programming language. Calling objects requires the name of the object and specific parameters, perhaps maintaining a value that is displayed. The process of calling a contract is more complicated but is based on the same logic. The user calls the contract with the desired parameters, this call is converted into a hash that is entered in a transaction tree as well as the address of the contract called and an amount if needed. The transaction is then signed by the user and sent to the network of nodes to be validated, if it approved, it is executed and stored by changing the state of the entire network.

Chapter 4: Security in Blockchain

Perhaps the most important part of Blockchain technology is benefits it offers in security matters. One of its advantages is the transparency from which it is governed. with the use of encryption for user records. The security it provides is originally derived from some of its operating principles. Also, with the use of safety algorithms and mechanism, the quality of security offered is high level, but there are also some problems in which solutions must be found.

4.1 Principles of Blockchain

The whole essence of Blockchain, which makes it safer to conduct transactions and smart contracts, are some of the principles by which it is regulated.

- Open ledger principle

  Everyone in the environment has access to all the data (open and public information). However, the information is not complete, but in sections that alone do not make sense. As a result, everyone has the information but no one's data is public

- Distributed ledger principle

  The principle of public ledger by itself does not mean much without the principle of distributed accounting. This ensures that everyone who wishes can keep a copy of the diary, the chain of blocks stored.

- Common ledger principle

  If someone wants to make a transaction within the network, they must first make their intention public. This transaction is not valid and is not added to the network. It must first be validated by other network users. Then a hash is found so that it can be registered in the Blockchain.

Based on these principles, a highly secure environment is created because it is easier to violate a central authority that manages everything, like a bank for example, than countless nodes in a network.

4.2 Security mechanisms and methods

Blockchain technology, apart from the principles it uses to create a safe and transparent environment, utilizes mechanisms and algorithms to ensure the efficiency of the system. Mechanisms such as cryptography are widely used on platforms based on Blockchain technologies and the use of private keys. In addition to the mechanisms used algorithms such as hashing, PoW and PoS that help in the transparency of the stored data.

Encryption

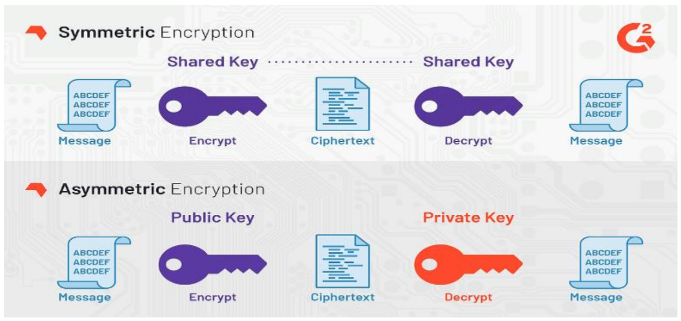
Blockchain technology uses encryption in a variety of ways to secure data transferred between nodes. Encryption is a method that has been used for centuries by mathematical authorities to store and transmit data in a specific form. The purpose of encryption is to ensure that the only person who reads or processes the data is the one for whom it was intended and no one else.

Simple example of encryption function:

- **Secret**= "Demiro Massessi blog post".
- **Function**= Swap each letter in the secret with a new letter according to the Key.
- **Key**= "+3".
- **Cipher**= "Ghpnur Pdvvhvvn eorl srvw".

The example above shows a simple encryption form for sending a secret message. Using the encryption function the original message is converted to a new one resulting from changing each letter to another. Specifically, each letter is converted to the letter that is 3 places later in the alphabet, ie a becomes d and b becomes e and so on, the recipient of the encrypted message will do the reverse process (decryption), each letter will change with the one three places before in the alphabet to get the original message sent.

Encryption is divided into symmetrical and asymmetric.



Picture 4.1 Differences in coding and decoding processes
between symmetric and asymmetric.

Symmetric cryptography is a simple form of cryptography that uses a single key to encrypt and decrypt data. This key can be a number, a word or a random sequence of characters. Once this key is set it is used to encrypt the data and send through a network securely. The same key used by the sender to encrypt is used to decrypt the data from the recipient. The main disadvantage of symmetric cryptography is that the key must be shared with all those who need access to the data, which can be difficult.

Asymmetric cryptography has a similar function to symmetric, it's just a little more complicated and offers a solution to the basic problem of symmetric. The main difference is that it uses a pair of keys, one public and one private. The public key can be considered as a username is available and shared on the network, while the private key plays the role of the password for the public key. It is impossible to manage the account without the public key. The private key is not available to everyone but only to selected users the sender decides who owns it and is used to approve actions on the account. The encryption process begins with the sender encrypting the data through the recipient's public key and then sending the encrypted message. The message is secure because only the receiver can decrypt it using the corresponding private key. Asymmetric encryption is considered to be the most secure encryption method because only the receiver can decrypt the data. Finally, the key pair can also function as a way of verifying an identity, which we will mention below.

The two versions of cryptography have similarities and differences, which means that they solve different types of problems. Asymmetric cryptography is preferred in cases where the user wishes to send encrypted data to some without giving away his private key. In contrast, symmetric cryptography is selected when the user has no problem assigning the private key or encrypting data for private use.

Digital signature

Digital signature is essentially a signature that provides integrity in two-way online transactions using asymmetric cryptography. Digital signatures are an inviolable and easily verifiable means with the use of asymmetric cryptography, they also use the concept of non-disclosure and are as binding as a regular signature. A pair of keys is used as in any method that uses asymmetric cryptography to ensure integrity and transparency between the two parties. The sender encrypts the data using the private key and sends it to the recipient who, when receiving it, uses the public key to decrypt it.

Hashing

Reliability against incorrect or malicious transaction data in a Blockchain environment is based on the use of a hashing algorithm. The hashing algorithm is the process of getting an input of an independent mean and outputting an encrypted sequence of fixed lengths created by a mathematical algorithm. The nature of the input to the hashing algorithm can vary form a message, a piece of information, a block of the Blockchain network or generally any information on the internet. But how the hashing algorithm ensures the data of a Blockchain network, the answer is using a hash. The algorithm dramatically increases the security of data stored and distributed on the Internet. This is due to the output of the hash algorithm which is a constant length regardless of the length of the input. It is impossible to calculate the length of the input using only the algorithm output.

Hashing algorithm must meet certain conditions in order to be considered useful to the user:

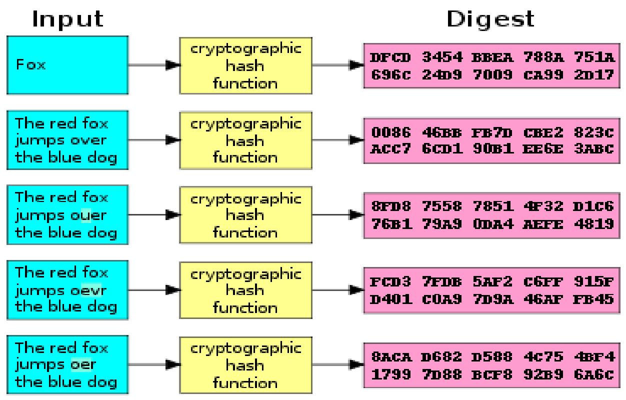
- It does not produce the same output for different input values, if this happened it would be difficult to decrypt the input value of the algorithm.

- The same input always produces the same output, similar to the above it would be difficult to authenticate the input value to the algorithm

- You need to be quick to generate an output price, otherwise it would not be reliable or useful for the user.

- It must be impossible to determine the input value to the algorithm using only the resulting output value, as it is perceived to be the most important condition for the operation of the algorithm.

- Finally, the slightest change in the input value should affect the output dramatically, it is a prerequisite for the security of the data managed by the network. If a small change in the input represents a small change in the output then it would be easier to determine the input of the algorithm.


Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C
Picture 4.2 Convert text to a Hashing algorithm.

The hashing algorithm ensures the data but also assures the user that the data has not been altered until it reaches the specified recipient. For example, if you download a file and use the algorithm and compare the output with the one sent to you by the file owner, it is clear if the file has been altered in the process. If the two hash, ie the algorithm outputs, are not the same then the data has been modified until it reaches the recipient.

In Blockchain environments, hashes are used to define the current state of the network. It practically includes everything have been done on the network so far additionally the data that is waiting to be added to the Blockchain. The output of the algorithm depends on the current state of the Blockchain network. As mentioned above, the slightest change in the input to the algorithm drastically affects the output, this ensures the transparency of data that is distributed through the network. Changing the slightest thing that has already happened on the network will affect all the hash but this will make them wrong and outdated. Network errors are immediately apparent because they affect the transparency of Blockchain technology, the changes in the chain must be verified by the network users to be considered valid.

The first block created in the network is called the birth block, it contains validated data and a unique hash is created. All the new blocks that enter the network create a hash that contains the hash of the previous block in order to create the chain of blocks. In order for the hash of each block to be valid, the other nodes of the network must agree, the blocks of the network communicate with the use of indicators. The indicators of the network blocks define its security because they contain the address of the previous block and its data. The hashing method is fundamental to the security of the network, it authenticates the data recorded on the network but also ensures the integrity of the network.

.

Proof of work

The PoW algorithm aims to ensure data validation and prevent attacks on the network. To understand the function of the algorithm we need to know how the data are stored in the blocks. When we want to enter data into the network they are first recorded in a temporary block, when this block is filled with data they are transmitted to all nodes of the network. At this stage of the process the nodes will check the validity of the data, then the PoW algorithm is used. Each node adds a piece of data to the block called 'nonce' and the 'block + nonce' is formed. This is then placed in an encryption mechanism that generates a hash of alphanumeric characters. The nodes compete with each other for which node will be the first to find the hash smaller than or equal to the one set from the beginning. Nodes can only change 'nonce' and create a new 'block + nonce' input which is placed in the same encryption mechanism. The first node that finds the correct result will have the data validated and rewarded by the network. Then they will send the alphanumeric 'block + nonce + hash' to the other nodes and if they validate it then the data will be entered in the chain.

4.3 Personal data security

The rapid development of digital technologies in recent years creates new challenges in terms of privacy and security of personal data. Especially if we consider the amount of data that is transmitted through the internet, since our daily life is now interwoven with it. Even the value of the data managed by companies has risen sharply as a result the benefit of the attackers is constantly increasing. Now the companies that dominate the business sector are social media and IT companies that manage a large volume of user data. Cases of violations and interception of user data are now daily cases, of which not even giant companies are excluded. Data storage in conventional media or cloud technologies did not have the desired effect. These storage methods proved to be inferior in terms of security, and also required the intervention of a third party to secure the data. It is obligated to find a solution that is characterized by data security and transparency is no longer a luxury offered but a necessary move.

Data security requires a more rational and modern approach, while at the same time giving the user access and control constantly. Blockchain technology meets these requirements, it is essentially a distributed registry that manages and stores data and transactions. The data stored is unchanged, once recorded it cannot be modified or deleted thus satisfying the transparency property. User anonymity and manageable access to data within the network is ensured through the use of private and public keys. Since, only the user has access to the data while he has the ability to provide access when he wishes.

The principle of data security in Blockchain is in the structure and mode of operation of the network. The technology is based on a decentralized network without a central administrator, the network stores and distributes data at all nodes. This network structure does not have a point where all the data is stored which would be its vulnerability to violations and attacks. The data is stored in a Blockchain where each block is related to the previous and subsequent blocks creating a more secure storage environment. To attack the network data, attackers must violate either all nodes or the majority of them. This is why we can say that the power of the network is in the numbers since in other networks if a specific node (the central administrator) is violated then the network data is exposed. The network also reduces access by third parties who want data for advertising purposes. This is achieved by rewarding users with "coins" when they are active or honest with the network and other nodes. Essentially, nodes become shareholders in the network and it is more profitable for them to work in favor of the network than against it.

The way data is stored and distributed within the network is an important factor in ensuring its security. The technology for storing them uses methods and algorithms that verifying them while guaranteeing their integrity. The use of a pair of encrypted keys ensures that access to the respective data will only be with the approval of their owner. Also, the use of public keys to validate data ensures their integrity. When the data is validated this is done using a coded key and then distributed to the nodes, all nodes have the same state of the chain of data. If the decoding of the data results in a different key, this means that the data was altered.

The user can utilize the technology in multiple ways to store and secure their data. Initially the user can split their files into pieces to store them partially to the network. Use encryption methods to store them so that only the owner has access to them or can provide access to them when it seems necessary. The owner can distribute the data to nodes so that everyone has access to them even if part of the network is not available. Essentially, instead of the user trusting his data to a third-party provider, he publishes it at the nodes of a network. The storage network is shared by users without anyone having access to the sensitive data of others, the owner of the data decides who will have access to them and under what conditions.

The technology offers a secure and efficient environment for storing data through the security methods and mechanisms used. The distribution of data through the network is instant while the cost of use is not high for the use of the environment. It also offers the user security in personal data since they do not have to worry about unauthorized access to them.

4.4 Blockchain security vulnerabilities

Blockchain technology is often presented as a secure solution for our data and transactions. But there are some technology issues that have not yet been fully resolved, as well as a hesitation from users in terms of trust. There are various examples where technology has created problems that have led to questioning the reliability of technology. Some US states have introduced bills to secure data stored or secured by distributed media such as Blockchain. These bills came into force after attacks on platforms using Blockchain technology were observed. However, if environments such as Blockchain need to be legally "fortified" for their security, this contradicts the purpose of their creation. Definitely, technology is a useful tool in the field of security, however there are areas that still require improvement to trust its users.

Decentralized network

An important advantage of Blockchain is that it is decentralized and retains data on a widespread network. Since the data is on the internet and is consequent there is a vulnerability in the security sector against threats. The network is accessible to all and everyone can join it then what ensures the security of the network against malicious attacks. However, in order to create a problem in the reliability of the network, many nodes must be violated at the same time. But how users are checked in terms of the trust they provide in the network, a solution to the problem would be an authorized system where members determine the permission of users to enter the network.

Smart contracts

Smart contracts are the most important tool in technology, if implemented properly is the answer to a plenty of problems. However, an error in the creation of the smart contract may jeopardize the involved parties and their data. One such example occurred on the Ethereum platform, which jeopardized the data of network users. The attack was quickly dealt with by the system and the developers rewrote the history of the blocks so that it seemed like the attack never happened. However, that way of dealing with the attack and erasing its existence is controversial. If the history of the blocks can be changed even to counter an attack, it does not contradict the reliability of the technology. This strict and perhaps 'dishonorable' solution counters the transparency of technology, which is one of its most important advantages. A more permanent solution for smart contracts unfortunately does not depend entirely on the network but on the smart contract creators. The ideal scenario would be to create contracts that operate as defined without the need of interference from third parties.

51% attack

When a new import to a Blockchain is made, it must be approved by the majority of nodes to be valid and then stored. But if an individual or group of individuals possess 51% of the computational power of the network, they have the majority. Then in theory they are dominant in the network and can approve any new entry they want according to their desires. While they can prevent imports into the network, they do not serve them or are against their desires. In other words, they become the determinant factor of the operation of the network but also for the storage and distribution of data within it. They could also modify as many blocks as they want and create a new chain. This would lead to a chaotic and unreliable network for users. This attack is seen in small networks. There is practically no solution for this problem only monitoring that 51 % of the computational power of the network is not concentrated by someone.

Network complexity

We mentioned above that in small networks it is easy for a group of individuals to maintain 51% of the computing power of the network. In large networks, such phenomena are not easy to happen but are vulnerable to deferent types of attack. In large networks that are constantly growing and increasing in volume, no one is aware of the threats that exist as well as the technological weaknesses that are created. In a network with many users, sharing encrypted keys can cause problems. An error by an individual user can cause a problem throughout the network and affect its security. Operation and maintenance of the security of a large system requires strong equipment and continuous development of the system infrastructure at least as soon as the network grows.

Security in public and private keys

It has been reported that entering a Blockchain environment requires the use of a public and a private key. The keys are encrypted sequences of characters, which are virtually impossible to guess. The access to the data of a Blockchain network without the use of these keys is impossible so the attackers aiming to possess these keys. As we said, it is impossible to guess these keys, then the only solution is to steal them from the user. The key spying process aims to compromise the security of the user's device, which is the most vulnerable part of the Blockchain environment. For the attackers, the possession of the keys is synonymous with their ownership since it is no different for the system. Smuggling can be prevented by the user by using appropriate security tools but also by storing the keys in encrypted form on their device. The system must change or modify its keys at regular intervals to ensure attacks.

4.5 Security in Ethereum

In addition to the principles and security methods used in every platform based on Blockchain technology, Ethereum adds different mechanisms and methods. These mechanisms are either improvements to standard technology mechanisms or innovations such as the use of smart contracts in a more rational way.

Initially the platform uses the PoS algorithm is a variation of the PoW algorithm. The two algorithms have the same purpose of validating data and transactions, but the methodology in which they reach the final goal is completely different. Unlike the PoW algorithm where the data validation node is the fastest in the PoS algorithm, the data validator user is selected in a deterministic way. All network nodes offer an amount as a collateral that the data will be validated. The network checks and maintains user data about the amount offered and selects the user who offered the largest. The algorithm does not require the solution of complex mathematical operations but is based on the financial offer of the users. The basic idea of the algorithm is that the use of an economic factor to achieve data validation is more efficient than the use of mathematical concepts. The use of the algorithm creates a safer environment since in order to violate the network the attacker must own 51% of the money the network manage.

Network security is achieved as users are 'shareholders' in the network so malicious actions on it would reduce the value of their share. Also, the algorithm is more efficient and needs fewer resources than PoW which requires a lot of computer energy which makes it expensive and malfunctioning.

Moreover, the way the platform manages the data stored on it affects its security. The platform forces the nodes to store all the data of the network but it was noticed that it creates problems in the way it works. The network is slower to perform actions but mainly makes it more vulnerable to attacks as the data gathers at the nodes. This has led to research into new network operating methodologies that are slowly being adopted. The division of the Blockchain was studied, which is the process of separating any information stored in the chain into smaller parts before storing them. Virtually every node contains a portion of the network's information and data depending on the transactions and actions it manages. If the node needs additional data to perform its functions, it must contact the corresponding node that holds them. It was also planned to conduct transactions outside the chain, the process of verifying transactions and data outside the main chain of blocks. Transactions are performed before they are validated in a temporary chain which is in full communication with the basic one using a communication protocol. At any time, the participants in the temporary chain can ask the main chain to confirm the blocks they manage and to update the status of the network. With the adoption of the two methodologies, the platform hopes to create a more efficient and secure environment for the users and the data being transferred.

4.5.1 Smart contracts

Ethereum has often been accused of not being a secure platform due to violations that have caused great damage to users. The truth is that the network itself is secure and not responsible for violations. The problem is usually caused by malfunctions that occur in the way smart contracts are executed. Smart contracts are pieces of code that promise autonomous and automatic execution as defined without third party intervention. The vast number of smart contracts on the platform and the constant need to create new ones have attracted the interest of a large number of developers. Most of them are not properly trained or disregard basic elements to create effective contracts that meet the requirements. Contracts often show errors in their writing that cause performance issues as well as security vulnerabilities. Experienced and fully trained developers are expected to develop smart contracts that contain errors it is obvious that the probability increase when contracts are created by inexperienced users.

Errors in developing contracts cannot be eliminated, but they can be minimized by checking the process of writing contracts. The code is checked either manually or automatically. Manual control is done by the code development team which will check the code line-by-line and simulate its operation in a virtual environment and different scenarios. It is undoubtedly the most effective type of control but it is very time consuming. While it requires the existence of an experienced team of developers who are trained in the subject matter while the hiring cost is expensive. In contrast, automatic control requires less time and is cheaper for the development team. Developers use programs to check for syntactic errors in the code but also to simulate it in different scenarios. This method saves resources and time for the development team but loses performance quality. Programs may ignore code errors or misrecognize a contract situation as problematic. It is understood that there is no control method that is effective in a full range so the choice should be made according to the requirements.

Even the programming language used to develop the platform's contracts creates security gaps. The smart contracts of the Ethereum platform are created with the Solidity language which allows the creation of applications without restrictions on complexity. Flexibility in how to write applications offered by a language makes it more accurate in terms of its maintenance and execution. Solidity creates a familiar development environment that attracts developers who want to experiment with something innovative. However, the environment still has gaps in security that, combined with the inexperience of the developers about the proper methodology, may create problems. Solidity requires the use of specific concepts and tools for the writing and checking of contracts in order to be evaluated according to the user's requirements

The programming language has a part for any malfunctions observed in the execution of contracts on the Ethereum platform. However, this does not mean that they only create problems on the platform and should be removed. On the contrary, it is a very important tool of technology since they can create applications from scratch. Creating contracts that work as defined and without errors in the code can be beneficial to the efficiency and security of the platform. Appropriate contracts can guarantee the anonymity of the user within the network, but also ensure that the data managed through the implementation of access control to them.

4.6 Access control

Access control is one of the most important mechanisms for managing and distributing data on the Internet. Access control provides data security by allowing only authorized access to data depending on the user's credentials. Credentials can be in the form of username and password, fingerprint scanning or part of code for the network to certify the user. Once the user confirms his identity, access to all or limited data and services of the network is provided depending on the level of access control. Access is usually divided into simple access that is limited to data reading and complex that offers the ability to manage and exploit data and services. In addition to accessing data and services, the ability to perform functions such as storing or creating a new transaction may also be checked. The purpose of the access control mechanism is to minimize unauthorized access to data and services within a network. If complete elimination of unauthorized access is not possible recording and controlling access is also a satisfactory scenario. Access control is a fundamental security mechanism that allows the system to set access levels to ensure that user data does not fall into the wrong hands. Even in the case of introducing a new user to the network using the appropriate software, the user is provided with the appropriate level of access to the network data.

Depending on the features and level of access provided, there are different types of access control. Initially there is a mandatory access control to which a central authority determines access to network data and services. Only the central authority has the ability to offer and remove access to the user. Then there is the discreet access control to which the data owner is responsible for providing access to them. The owner is the only one who provides access to the data, he manages to the stored data depending on the identity of the user. There is also a control over access to rules and features in which access is determined by predefined rules such as time, date, network or user characteristics. Finally, there is the role-based access control which the access level is determined by the role that the user occupies in the network. It doesn't matter what each user is, but users are grouped according to their roles. Access to data is provided to all users in the respective role and not to each one individually. Each role includes licenses provided within the network as well as restrictions imposed on it.

No type of access control is a panacea but the appropriate one is determined by the level of access required and the system features. If the system requires a strict level of security set by a central authority then mandatory control is the right choice. On the other hand, a system that is useful to offer the owner the option to choose who has access to store data then discreet control is the right mechanism. It is understood that the first mechanism is not suitable for Blockchain technologies since they need a central authority for the security of the system. In general, the concept of access control contradicts the public and decentralized nature of technology. But the security of the data stored by the system is necessary for the trust that the platform offers. For example, a network that manages user medical data that is classified as sensitive cannot provide access to all users. A mechanism is required that secures patient data and anonymity of the owner is achieved by using access control. Access control provides the required security on the network with authorized access to data. Security is not only the prohibition of access to data to specific users but also the registration of users who have access in them and when.

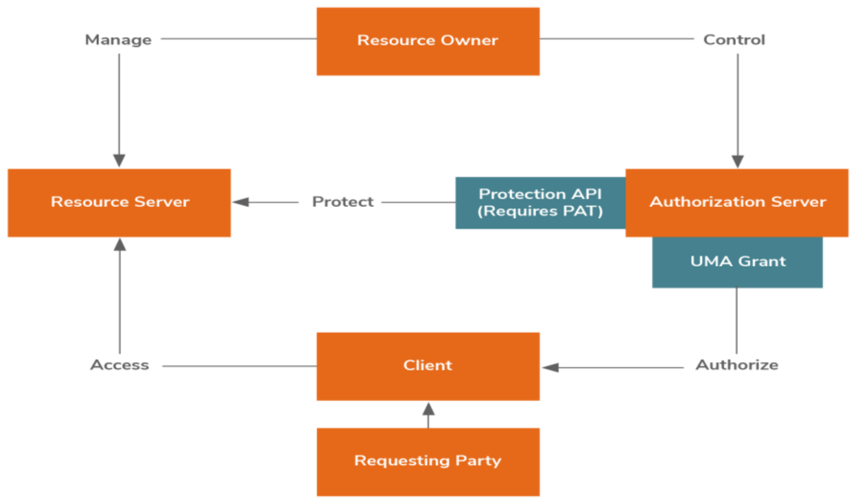
Ethereum as a platform uses not only methods such as hashing algorithms and encryption for personal data security. Users can save the keys to decode of the records off-chain. If they want another user to have access to their data then they give them access to the appropriate key. This is a type of access control used by the platform that is usually achieved by implementing smart contracts. Even while the development of the contract, the user can use private variables and functions that have limited access by nature.

4.6.1 User-Managed Access (UMA)

In addition to standard access control Blockchain technology can implement protocols such as UMA that focus on the owner of the respective data. UMA is a universal authorization protocol that allows data sharing between users. UMA relies on OAuth, which is a framework that allows applications and users to access their data over the Internet. UMA enables the data owner to control and select who can access them. In this way, the data of the users is ensured from unauthorized access by third parties. The data managed by the user can be of various types such as personal and medical records or audiovisual content. Data sharing between users is applied in three deferent types of data sharing. They support the sharing of user to self, from user to user and user to organization.

An example to understand both the concepts and how the protocol works is a scenario of granting access to a user's photos. Suppose the user wants his social media accounts to have access to his photos stored in a cloud. This is achieved by using the OAuth framework that allows the user to provide access to his data to other accounts he owns. However, if the owner wishes to provide access to his data to other users or an organization, there must be a security protocol. An access method can be achieved if the owner grants the corresponding codes to the other user but this is not appropriate for security reasons. On the other hand, using a protocol such as UMA is beneficial for the security of the owner and the data being protected. Using the protocol, the owner can select the shared data and the users who have access to it. The framework also guarantees the security of operations while preventing unauthorized access to data.

The UMA protocol usually consists of five roles that interact with each other and perform different functions in the system. First of all, the owner of the data is responsible for providing access to the protected data. Then there is the requested party, a person or group that seeks access to protected data with the support of the client. The client is an application that submits a request for access on behalf of requested party to the protected data according to the authorization policies established by the data owner. In addition, there is the data server that hosts and manages both the data and the requests for access to them. Finally, the authorizing server guarantees the security of the data stored in the data decorator and the access concession process. Multiple roles can be performed by the same entity within a developed environment.



Picture 4.3 The authorization process using the UMA framework.

During the operation of the protocol, as shown above, it consists of five procedures. First the owner **manages** his personal data on the resource server. Second, the authorization server and the resource server are connected to each other with the UMA framework to **protect** the security of the procedures. Thirdly, the resource owner **controls** who has access to the stored data by creating appropriate policies for the authorization server, this allows the owner to grant consent asynchronously and not at the time of the application. Fourth, the client acting on behalf of the requested group uses the UMA policies of the authorizing officer to obtain the unique **authorize** token by the owner to provide asynchronous access. Finally, the client presents to the resource server the token that if verified **access** is provided to the data in the requested group.

In general, UMA offers the ability of the data holder to control who can access to the digital form of the data through a server. This is achieved by using authorization policies regardless of the server in which the data is stored. For example, a patient has the ability to authorize their doctor or insurance company to access their medical records. The sharing of data is done selectively by the owner who is solely responsible for the process allowing the owner to have full control over the data he shares with the respective users. Finally, the framework guarantees the appropriate conditions for the granting of access as well as the independent decision by the user.

Regarding the mode of operation of the mechanism, we can observe it from a visual point of view of the human factor, ie the owner of the record and the requested group.

The owner informs the resource server that he wants to share his data with other users. The resource server requests the owner to create a token that will verify the authorized access to the protected resources by the authorization server. The resource server then sends a request to the authorization server to approve the created token. If the approval process by the authorization server is valid then the owner is responsible for handling the resources he owns. Finally, the owner determines the criteria for third party access to the sharable resources.

The requested team uses the client application to operate on its behalf to gain access to protected resources by the UMA mechanism. The client sends a request to the authorization server using a token, the server checks the validity of the token. If the token is not guaranteed then the server sends an appropriate message. In addition, the customer receives a "ticket" to use in order to obtain the appropriate token that meets the security policies of the protected resources. The customer sends the ticket and a modified form of the original token on the authorization server. If the process is successful then the server sends a token to the customer to access the protected resources, the customer uses it when he wants access to resources. When the client tries to access the protected resources, the authorization server checks the validity of the token, if the process is successful sends a message to the resource server to provide access to the customer.

Chapter 5: Smart Contract Lab Environment

5.1 Tools

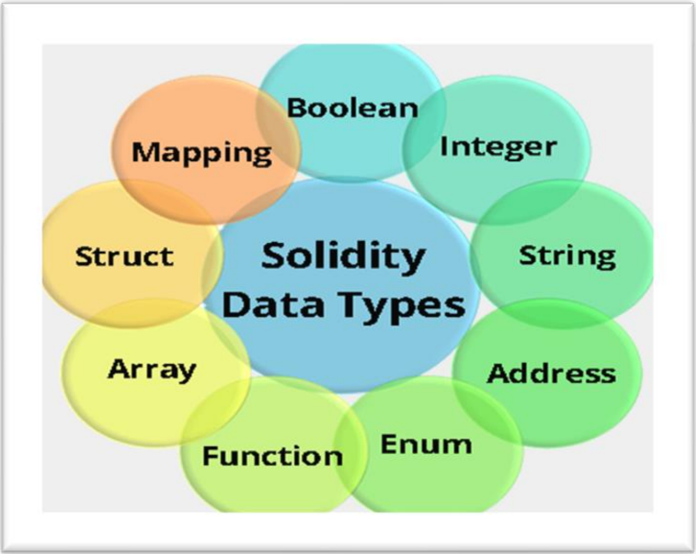
The creation of the contract analyzed in the previous section to resolve the problematic situation requires the use of various tools. The tools are aimed not only at creating the contract and its operation but also at its interaction with the environment. Initially, it was written using the Solidity programming language via the text editor visual studio code. In addition, tools were needed to create a virtual environment to simulate its operation in real conditions. This allows you to control the code and how to execute the contract without spending resources on an Ethereum network. The tools used will be analyzed below in order to understand how they work. At the same time, the role that affected the operation of the contract will be analyzed.

Solidity

Picture 5.1 The Solidity tool logo.

The programming language mainly used on the Ethereum platform is Solidity, which offers a familiar programming environment for creating contracts. The way it is written and executed are similar to the Java language from which it inherits the methodology and functionality of an object-oriented language. The way it is written is simplified as the EVM significantly reduces the complexity of the programs by performing actions in the background. The structure of the code begins with the declaration of the version used so that the Compiler knows the actions it needs to perform. It may then include importing functions from other network connections that work like libraries into other programming knowledge.

As we mentioned, it inherits the functions and logic of object-oriented languages such as declaring variables and using functions, but it differs in the way they work. Initially it contains types of variables such as (string, unit, Boolean) that are necessary for the operation of the network but introduces new types especially for the network. First, the address type is entered, which is a sequence of bits that refers to an Ethereum network account and offers the ability to communicate and execute actions with the corresponding account. Also, the type of mapping that works like an index or a fragmentation table, uses two values (key and value) that are associated with each other. The variable key can be a data type such as (uint, string, address) while the variable value can be a data type or an object such as a constructor or struct. The event type is used as the functions and marks an event depending on the parameters specified. Finally, there is the enum type used to create new types of objects that correspond to constant integers starting at 0.

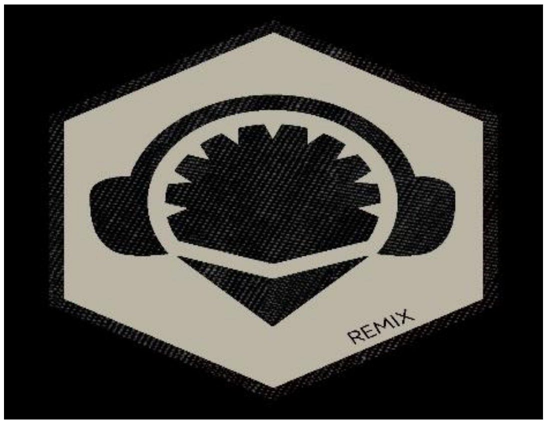


Picture 5.2 The different variables type that Solidity supports.

Apart from the differences in the data types that Solidity presents, their functions and mode of operation perform an important role. Functions are the way of communication between contracts. Most programming languages are limited to setting public and private functions and allow you to return prices depending on their type. In Solidity, in addition to the data it contains as parameters, the main types of functions are external or internal. Internal functions can be used within the contract or as a means of contract produced by it. External can be used outside of contracts by users. In addition, there are functions that allow payment within them and are called payable. These contracts are very important in Blockchain networks as they allow the exchange of money between the contracts and the users. Depending on the access that the function has to the data stored in the state trie, they are divided into pure and view. Pure functions do not have the ability to read or modify data. On the contrary, view functions can read data but cannot modify the status of the network. Finally, the term modifier is introduced, which is used to convert the functions of a function according to a condition of the require form. Although Solidity is Turing complete, it needs its combined use with tools to create a complete construction and execution of contracts.

Remix IDE

The Remix IDE is a simulation tool for executing smart contracts written in the Solidity language. The tool is used through a browser and creates a virtual environment for operating contracts off-chain. It enables the developer to check the mode of operation as well as the execution results of the contracts without the costs required within a Blockchain network.

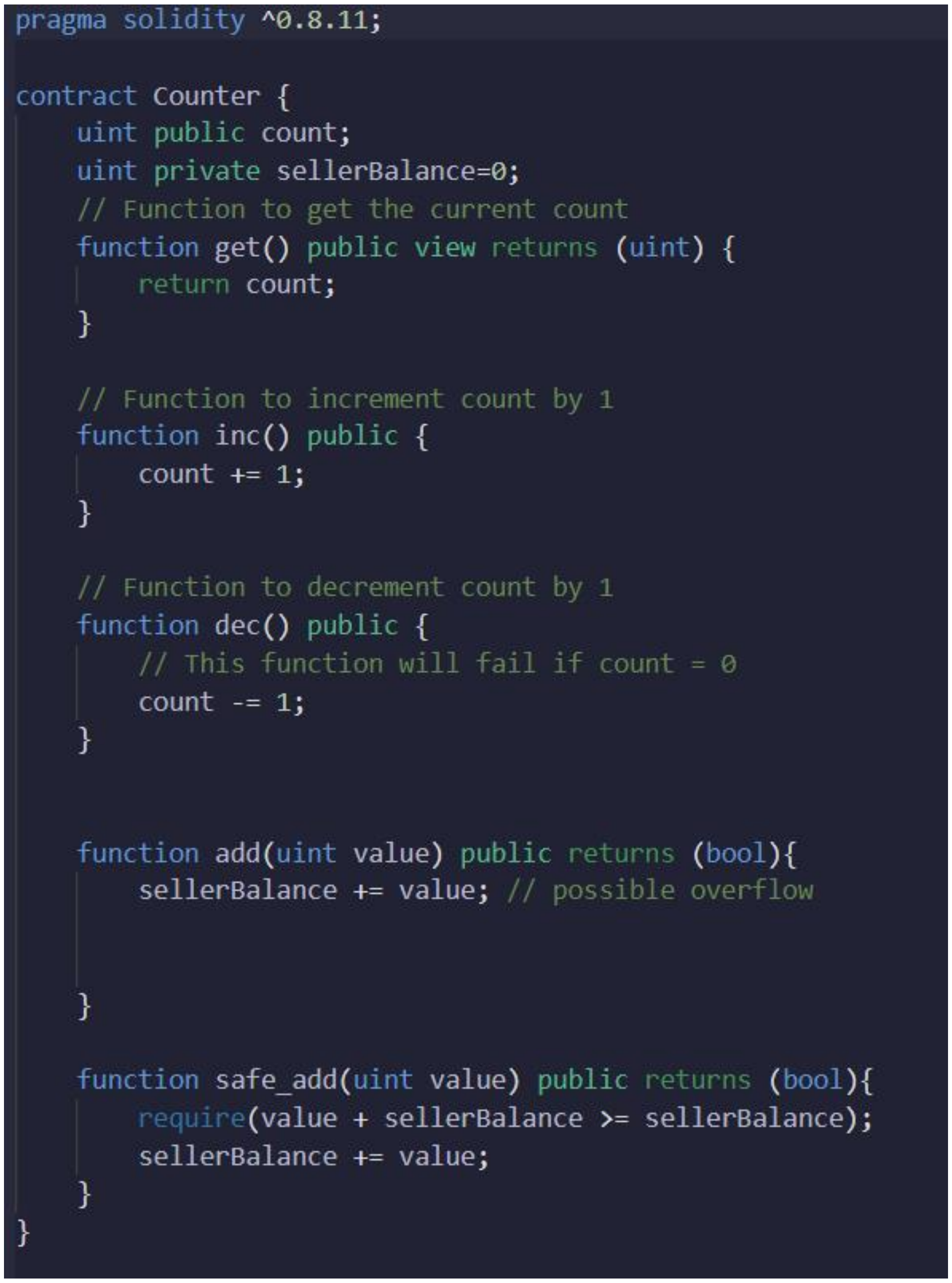


Picture 5.4 The REMIX IDE tool logo.

The tool essentially builds a virtual private Blockchain network with 5 addresses (the user can add more if desired) and a balance 100 ether that works temporarily on the Internet. The tool helps the developer in writing and correcting contracts as it recognizes syntactic errors when creating the contract. The tool uses the appropriate version of the compiler according to the user's choice. If there are no errors when writing the contract then the deploy process is done which allows the user access to the functions and operation of the contract as programmed. Finally, it presents information about the transactions if they were executed or there was a problem but also on the input and output of the transactions when they exist.

5.1 Smart contract Testing example

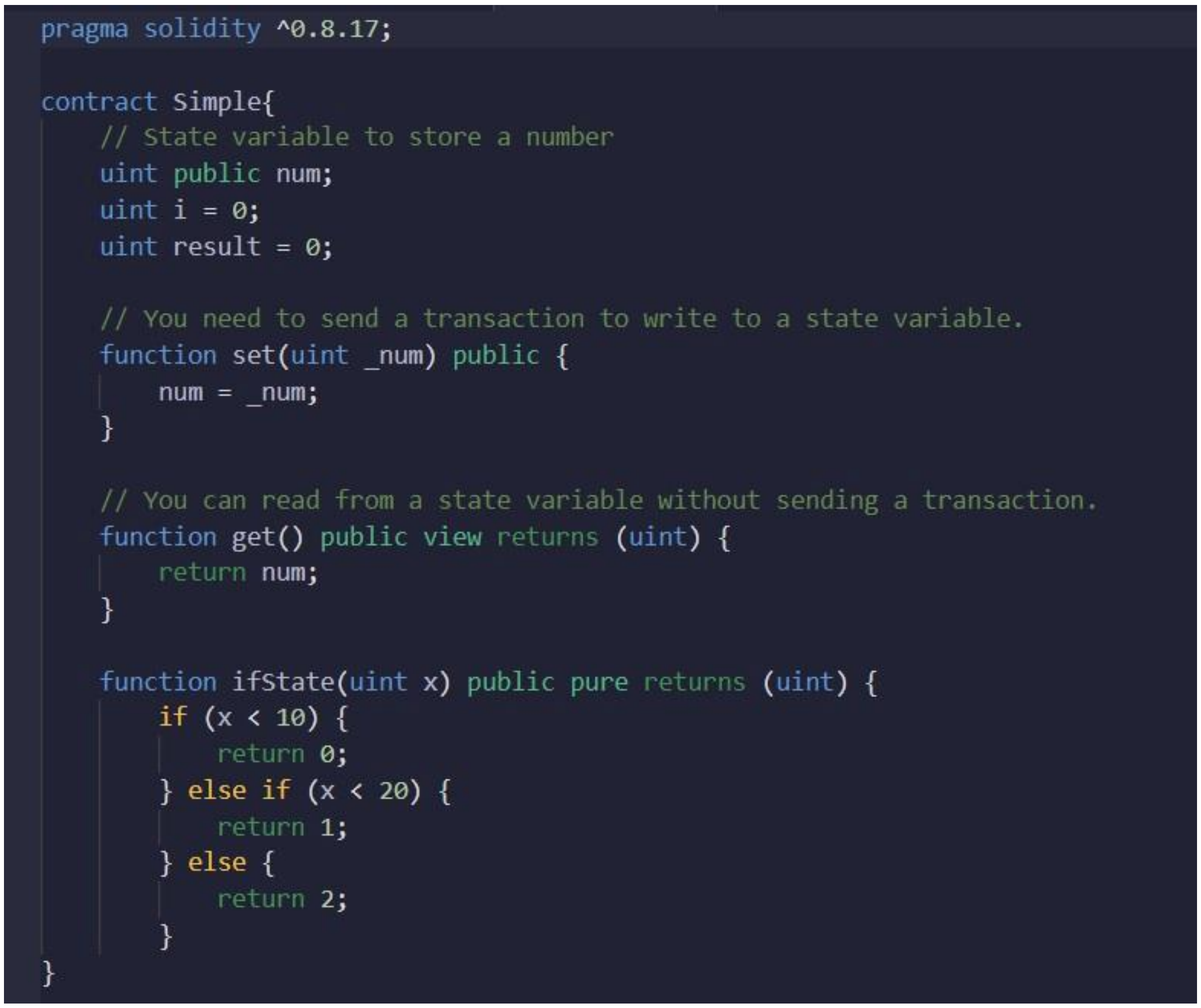
This chapter we would analyze basing functionalities of the Ethereum smart contract via examples written on solidity and deployed on Remix IDE. The smart contract was based on smart contracts on https://solidity-by-example.org/ where you could find more examples for solidity smart contract.



```
pragma solidity ^0.8.11;

contract Counter {
    uint public count;
    uint private sellerBalance=0;
    // Function to get the current count
    function get() public view returns (uint) {
        return count;
    }

    // Function to increment count by 1
    function inc() public {
        count += 1;
    }

    // Function to decrement count by 1
    function dec() public {
        // This function will fail if count = 0
        count -= 1;
    }


    function add(uint value) public returns (bool){
        sellerBalance += value; // possible overflow



    }

    function safe_add(uint value) public returns (bool){
        require(value + sellerBalance >= sellerBalance);
        sellerBalance += value;
    }
}
```

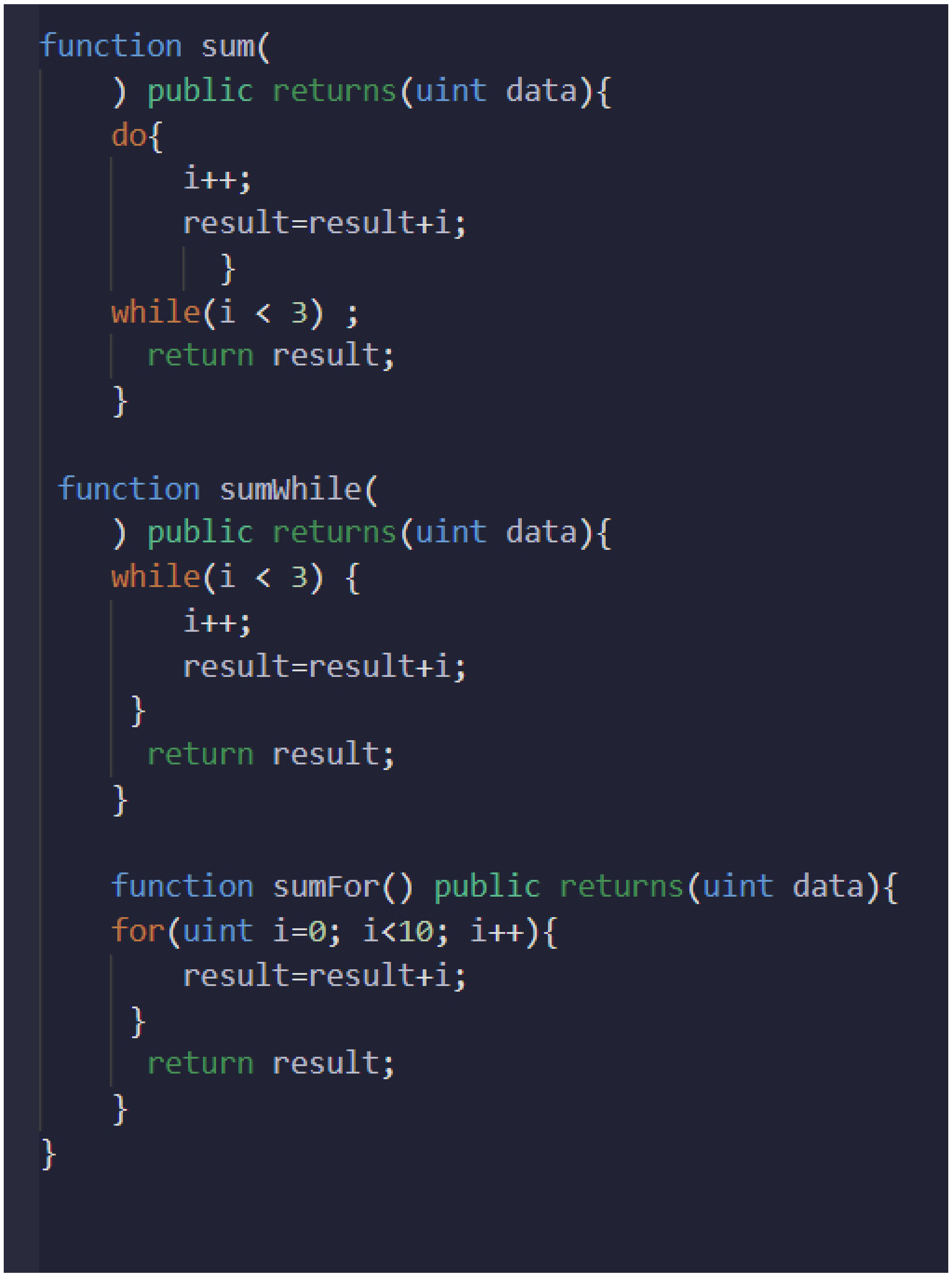
5.5 Counter smart contract

The above code consists of five functions (get, add, inc, dec, safe_add) and two variables count and sellerBalance. The functions add and inc show the basic methodology of adding and decreasing the value of int variables called count, also get function is a method that the user can extract the count current value. The last two functions of the smart contract are to add a value on the seller’s balance with the difference that safe_add function checks that the value that is added is not a negative number.



```
pragma solidity ^0.8.17;

contract Simple{
    // State variable to store a number
    uint public num;
    uint i = 0;
    uint result = 0;

    // You need to send a transaction to write to a state variable.
    function set(uint _num) public {
        num = _num;
    }

    // You can read from a state variable without sending a transaction.
    function get() public view returns (uint) {
        return num;
    }

    function ifState(uint x) public pure returns (uint) {
        if (x < 10) {
            return 0;
        } else if (x < 20) {
            return 1;
        } else {
            return 2;
        }
}
```

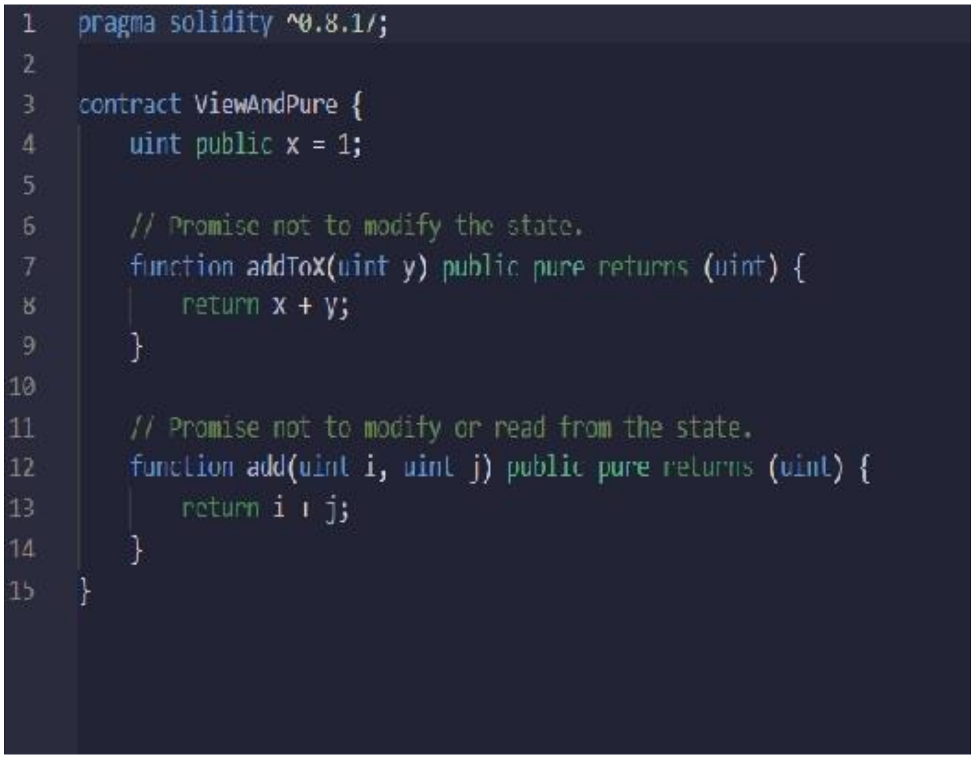
5.6 Simple smart contract



```
function sum(
    ) public returns(uint data){
    do{
        i++;
        result=result+i;
          }
    while(i < 3) ;
      return result;
    }

 function sumWhile(
    ) public returns(uint data){
    while(i < 3) {
        i++;
        result=result+i;
     }
      return result;
    }

    function sumFor() public returns(uint data){
    for(uint i=0; i<10; i++){
        result=result+i;
     }
      return result;
    }
}
```

5.7 Simple smart contract

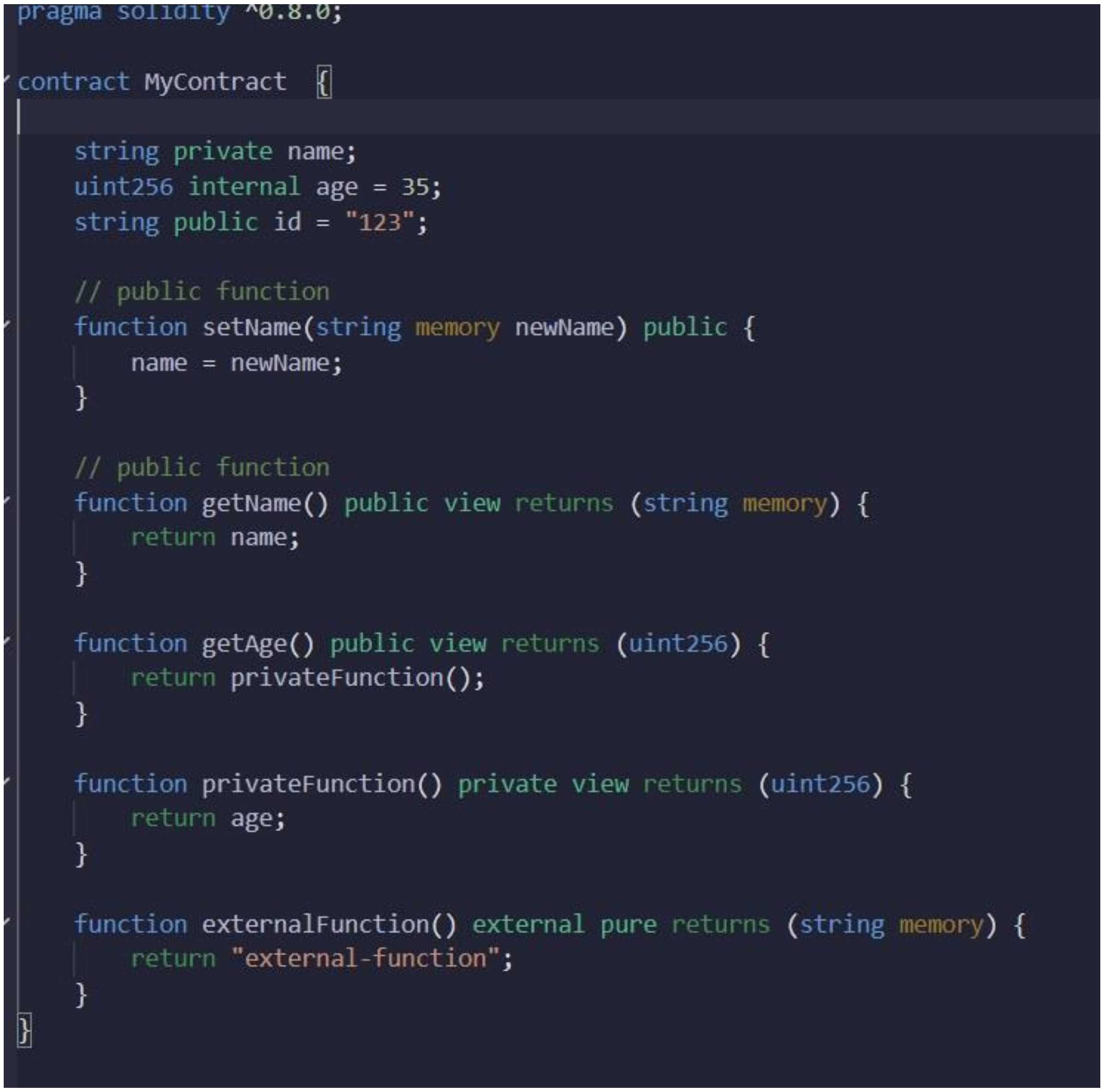
The goal of the smart contract called simple is to show the user the way smart contracts are implementing basic programming methods. The first two functions give an example of setting a variable's value through input of the user. Next it shows an if_else methodology on solidity that check the value of variable x and print a specific message depending specific requirements. The last functions show the basic loop methods on programming do_while, while_do and for by creating a sum variable and adding to it until 10 and then pint it.



```
pragma solidity ^0.8.17;

contract ViewAndPure {
    uint public x = 1;

    // Promise not to modify the state.
    function addToX(uint y) public pure returns (uint) {
        return x + y;
    }

    // Promise not to modify or read from the state.
    function add(uint i, uint j) public pure returns (uint) {
        return i + j;
    }
}
```

5.8 ViewPure smart contract

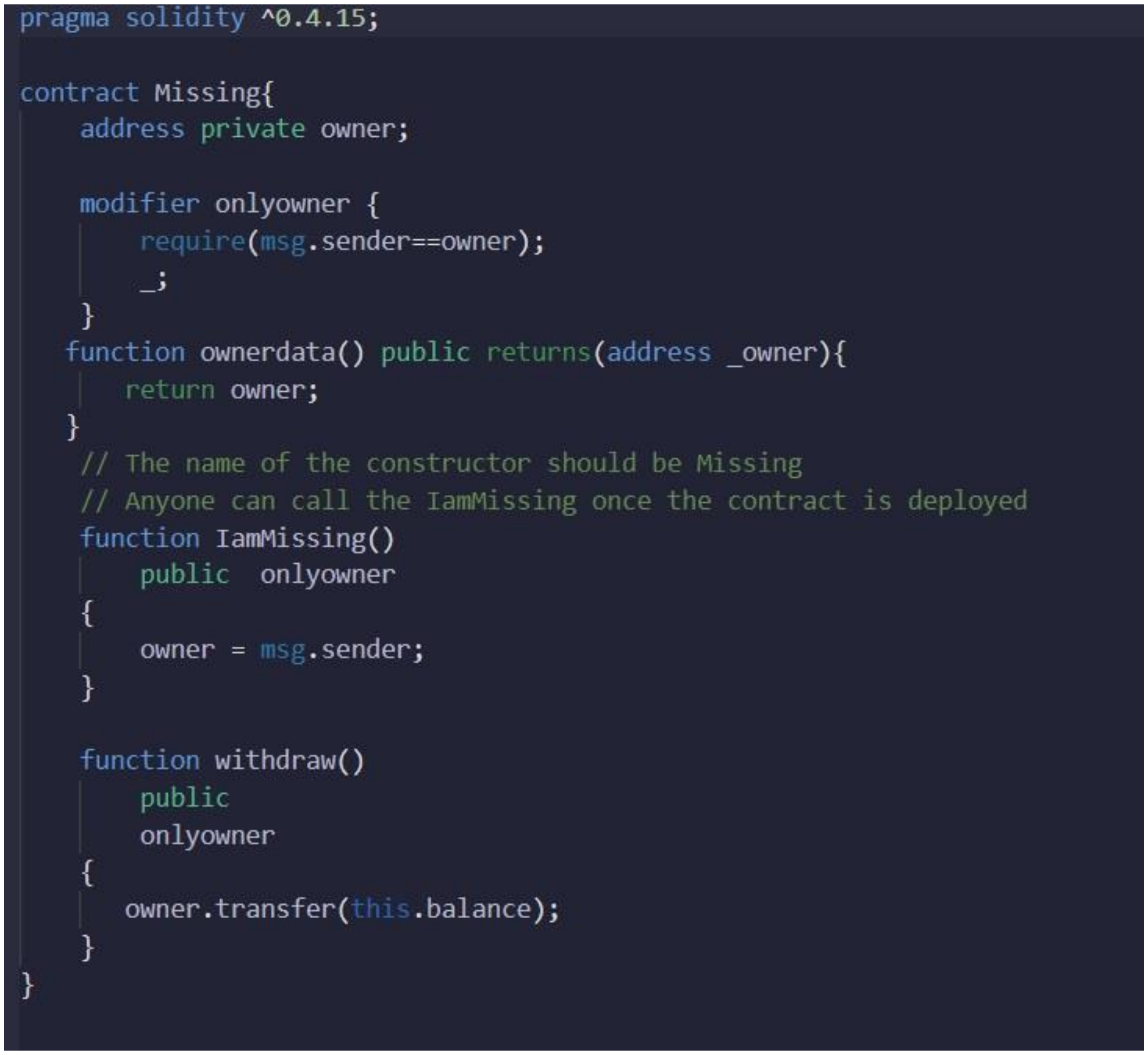
ViewAndPure smart contract tries to specify the deference between of pure and view modifier for Functions. There are two functions that add two values, the deference is that the two functions take deference amount of parameters from the user. A view function can read values from the Blockchain so it need only one input from the user to add to one stored already on the ledger. Contrary the pure functions can only view values from the ledger and not use them or modify them as a result I needs to uint input from the user to add them. It is important to mention that if we change the view function to pure (as it is shown above) the smart contract shows an error and cannot be compiled.



```solidity
pragma solidity ^0.8.0;

contract MyContract  {

    string private name;
    uint256 internal age = 35;
    string public id = "123";

    // public function
    function setName(string memory newName) public {
        name = newName;
    }

    // public function
    function getName() public view returns (string memory) {
        return name;
    }

    function getAge() public view returns (uint256) {
        return privateFunction();
    }

    function privateFunction() private view returns (uint256) {
        return age;
    }

    function externalFunction() external pure returns (string memory) {
        return "external-function";
    }
}
```

5.9 MyContract smart contract

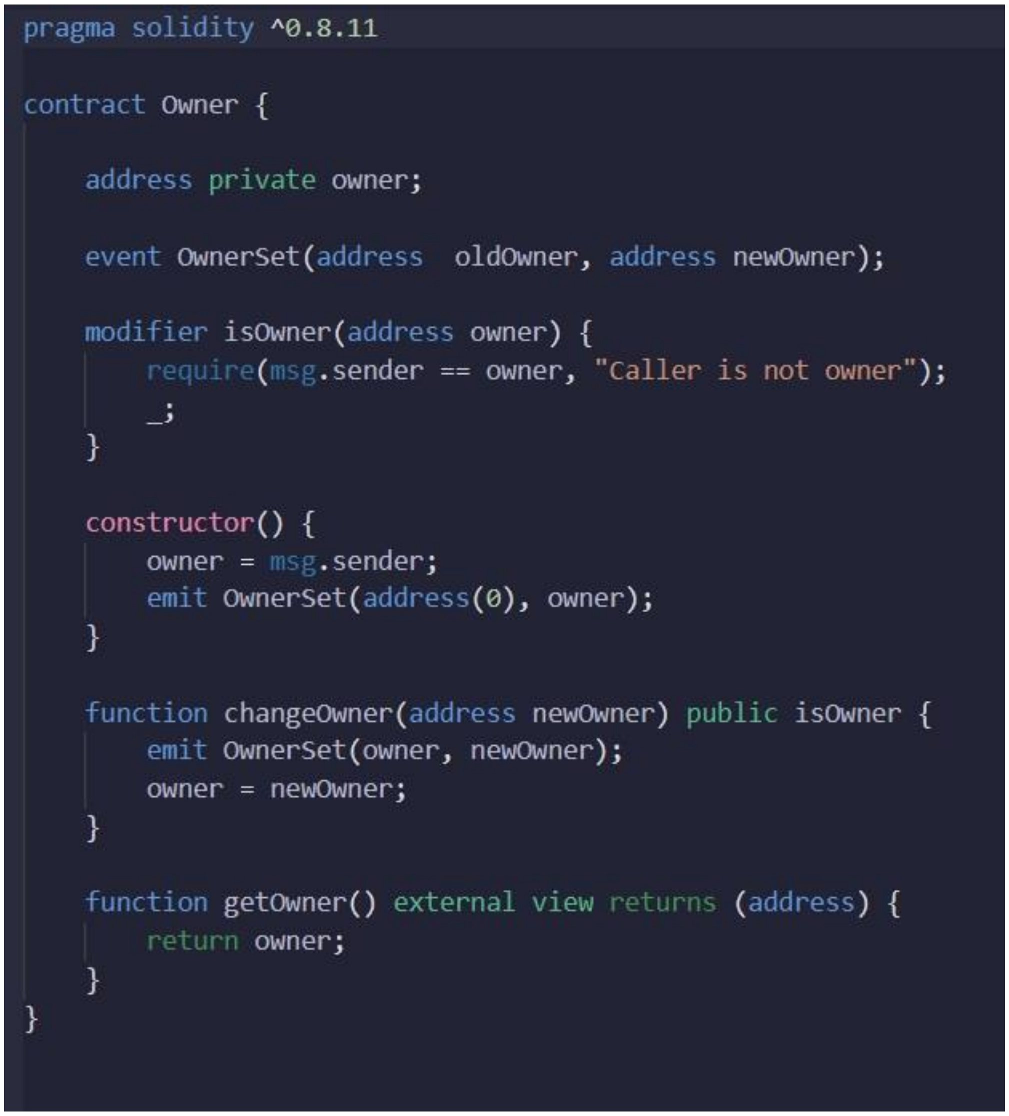
The myContract smart contract show the user the different visibility of functions depending on if it is public, private or external. **Private** – A private function/state variable is only available inside the contract that defines it. It is generally good practice to keep functions private. **Internal** – A internal function/state variable is only available inside the contract that defines it AND any contracts that inherit it. **Externa**l – An external function can only be called by external contacts. Not visible inside the contract that defines it. **Public** – A public function/state variable is available to any contract or third party that wants to call it. Public is the default if visibility is not specified.



```
pragma solidity ^0.4.15;

contract Missing{
    address private owner;

    modifier onlyowner {
        require(msg.sender==owner);
        _;
    }
   function ownerdata() public returns(address _owner){
       return owner;
   }
    // The name of the constructor should be Missing
    // Anyone can call the IamMissing once the contract is deployed
    function IamMissing()
        public  onlyowner
    {
        owner = msg.sender;
    }

    function withdraw()
        public
        onlyowner
    {
       owner.transfer(this.balance);
    }
}
```

5.10 Missing smart contract

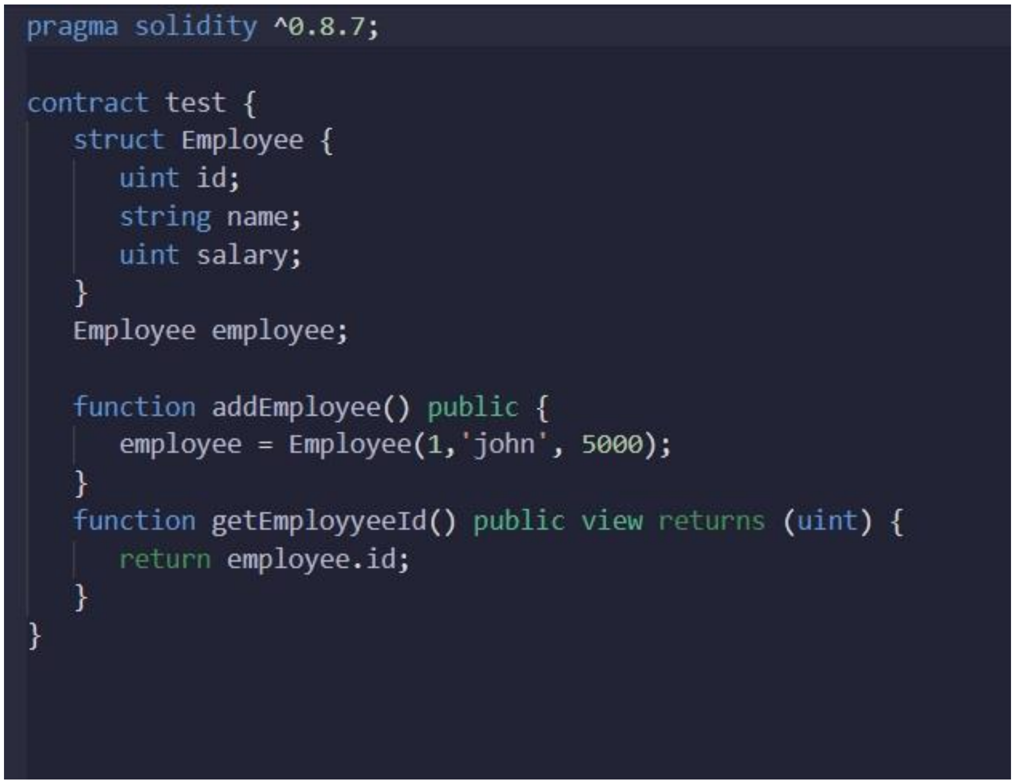
The missing smart contract shows a security issue with the way it is created by missing a constructor. Anyone can call the IamMissing function and become the owner of the smart contract, if someone become the owner can use the withdraw function and 'steal' the funds stored on the smart contract. Modifier is a special type of function on solidity that consist of a require statement that means when you use it in a function the require statement must be valid in order to run the function code. For example in the Missing smart contract the withdraw function have the onlyOwner modifier so it will check if the transaction sender is the owner, if it is true then the function will run as it is supposed otherwise it will not.



```
pragma solidity ^0.8.11

contract Owner {

    address private owner;

    event OwnerSet(address  oldOwner, address newOwner);

    modifier isOwner(address owner) {
        require(msg.sender == owner, "Caller is not owner");
        _;
    }

    constructor() {
        owner = msg.sender;
        emit OwnerSet(address(0), owner);
    }

    function changeOwner(address newOwner) public isOwner {
        emit OwnerSet(owner, newOwner);
        owner = newOwner;
    }

    function getOwner() external view returns (address) {
        return owner;
    }
}
```

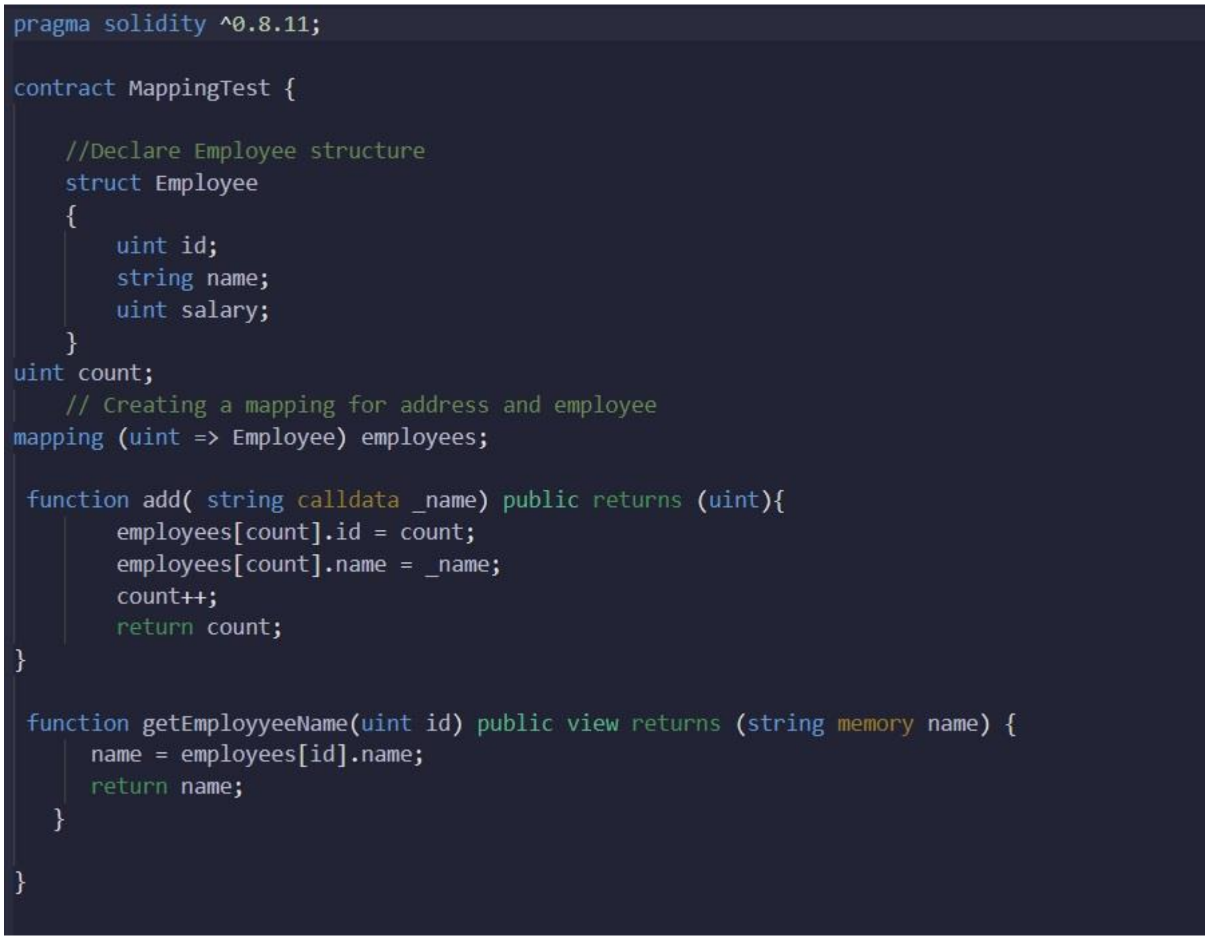
5.11 Owner smart contract

Owner smart contract solves the security issue that is mentioned above the solution is simple using a constructor. With the usage of constructor which by definition it is called only once when the smart contract is deployed, when the constructor is called sets the address that deployed the contracts as the owner. Later with usage of the modifier onlyOwner the owner of the smart contract can change the ownership of the contract.



```
pragma solidity ^0.8.7;

contract test {
   struct Employee {
      uint id;
      string name;
      uint salary;
   }
   Employee employee;

   function addEmployee() public {
      employee = Employee(1,'john', 5000);
   }
   function getEmployyeeId() public view returns (uint) {
      return employee.id;
   }
}
```

5.12 Test smart contract



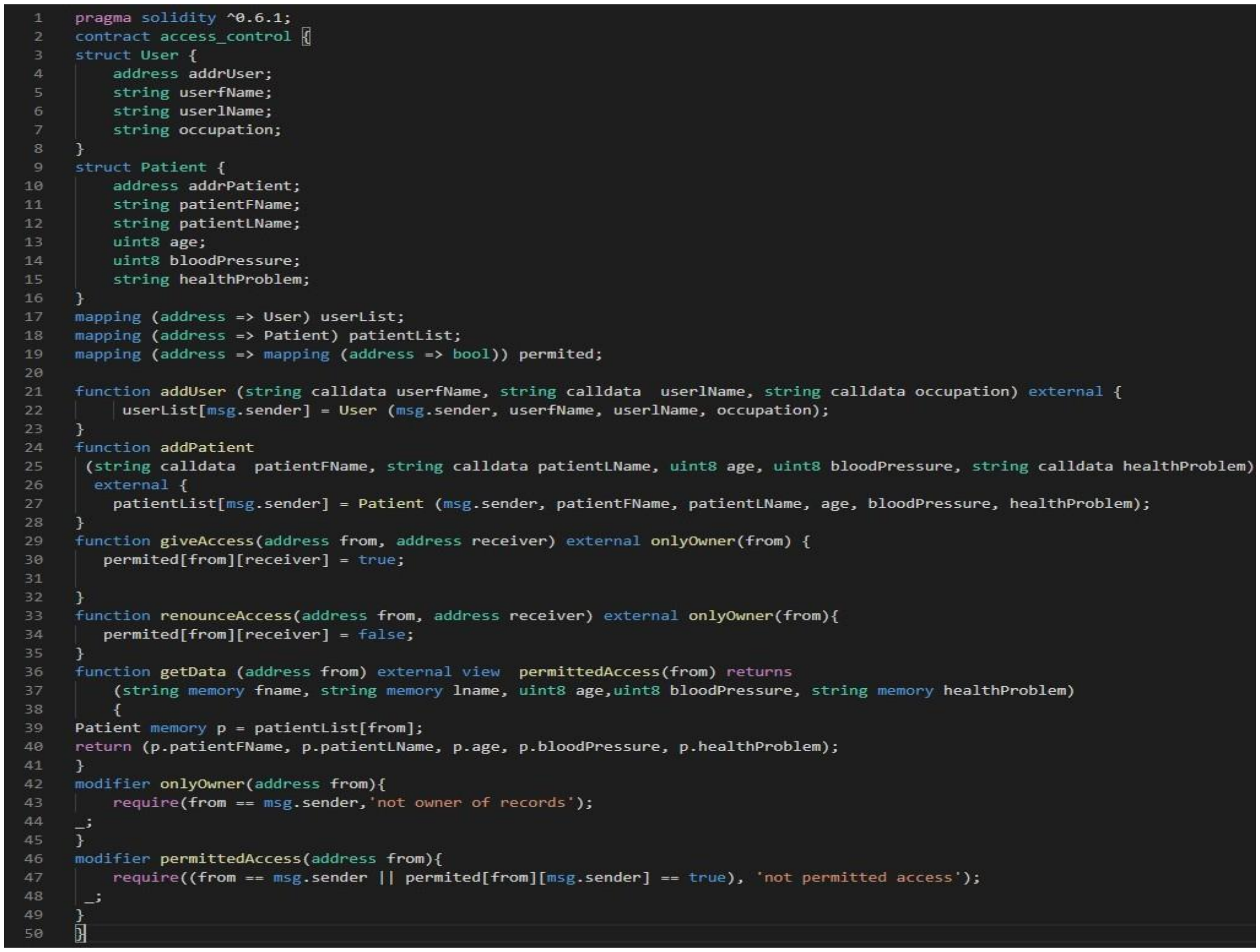
```
pragma solidity ^0.8.11;

contract MappingTest {

    //Declare Employee structure
    struct Employee
    {
        uint id;
        string name;
        uint salary;
    }
uint count;
    // Creating a mapping for address and employee
mapping (uint => Employee) employees;

 function add( string calldata _name) public returns (uint){
        employees[count].id = count;
        employees[count].name = _name;
        count++;
        return count;
}

 function getEmployyeeName(uint id) public view returns (string memory name) {
      name = employees[id].name;
      return name;
   }

}
```

5.13 MappingTest smart contract

The smart contracts above both create a struct of employees to store data of them such as id, name and salary. The first one stored them as a struct and the later use a mapping that use the id of employ to associate them to specific values.

5.2 Development AccessControl smart contract

For the implementation of the solution, a smart contract was developed that aims at the management of user data, but also the access control to them. The functions performed are the introduction of new user data as well as their management within the Blockchain. The contract allows the user to grant access to the data stored by third parties whenever they wish. It is also possible to revoke the granted access when seems necessary. Finally, safety policies are used to enforce the contract properly.



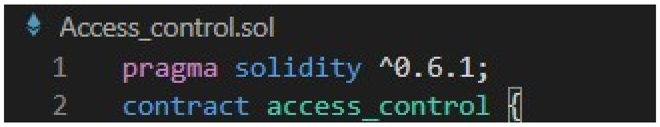
```
1   pragma solidity ^0.6.1;
2   contract access_control {
3   struct User {
4       address addrUser;
5       string userfName;
6       string userlName;
7       string occupation;
8   }
9   struct Patient {
10      address addrPatient;
11      string patientFName;
12      string patientLName;
13      uint8 age;
14      uint8 bloodPressure;
15      string healthProblem;
16  }
17  mapping (address => User) userList;
18  mapping (address => Patient) patientList;
19  mapping (address => mapping (address => bool)) permited;
20
21  function addUser (string calldata userfName, string calldata  userlName, string calldata occupation) external {
22       userList[msg.sender] = User (msg.sender, userfName, userlName, occupation);
23  }
24  function addPatient
25   (string calldata  patientFName, string calldata patientLName, uint8 age, uint8 bloodPressure, string calldata healthProblem)
26    external {
27      patientList[msg.sender] = Patient (msg.sender, patientFName, patientLName, age, bloodPressure, healthProblem);
28  }
29  function giveAccess(address from, address receiver) external onlyOwner(from) {
30     permited[from][receiver] = true;
31
32  }
33  function renounceAccess(address from, address receiver) external onlyOwner(from){
34     permited[from][receiver] = false;
35  }
36  function getData (address from) external view  permittedAccess(from) returns
37      (string memory fname, string memory lname, uint8 age,uint8 bloodPressure, string memory healthProblem)
38      {
39  Patient memory p = patientList[from];
40  return (p.patientFName, p.patientLName, p.age, p.bloodPressure, p.healthProblem);
41  }
42  modifier onlyOwner(address from){
43      require(from == msg.sender,'not owner of records');
44  _;
45  }
46  modifier permittedAccess(address from){
47      require((from == msg.sender || permited[from][msg.sender] == true), 'not permitted access');
48   _;
49  }
50  }
```

Picture 5.14 The form of the Access_control contract.

The form of the contract is shown in the picture above (Picture 5.5) while the next section refers to the step-by-step analysis of the contract procedures.

5.2.1 Partial analysis of the smart contract

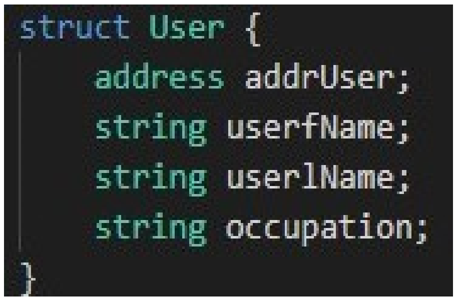
This section discusses how the smart contract was develop and how it functions to secure data. The methodology for the execution of the orders will also be thoroughly analyzed as well as parameters necessary for the purpose of its proper operation



```
Access_control.sol
1   pragma solidity ^0.6.1;
2   contract access_control {
```

Picture 5.15 Version statement and contract name.

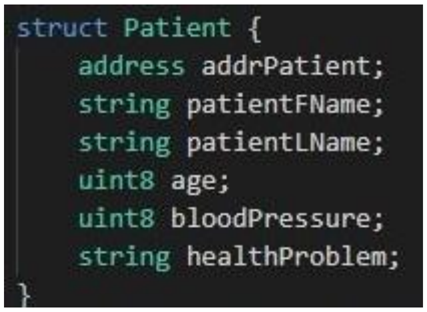
Every smart contract implemented through Solidity starts with the pragma statement followed by the language version. Line 1 shows that the version used is 6.0.1. This allows the compiler to know the commands and functions that can be performed depending on the version being used. The declaration is followed by a 'class' following the name of the contract that operates as the main function in Java. Line two shows that the name of the contract is access_control.



```
struct User {
    address addrUser;
    string userfName;
    string userlName;
    string occupation;
}
```

Picture 5.16 User Object Structure.

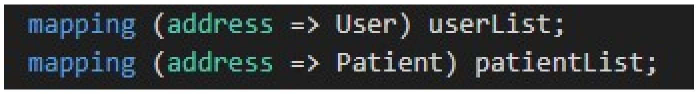
Picture 5.7 creates a structure of User-type objects. User objects have common features that are an address and three alphanumeric ones. AddrUSer is the address that the object stored within the network and is a unique key while the alphanumeric userfName and userlName indicate the username of the respective user while the occupation indicates the user status.



```
struct Patient {
    address addrPatient;
    string patientFName;
    string patientLName;
    uint8 age;
    uint8 bloodPressure;
    string healthProblem;
}
```

Picture 5.17 Patient object structure.

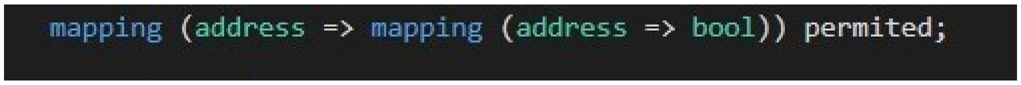
Picture 5.8, as above, creates a structure of patient-type objects that are characterized by common elements, one address and two alphanumeric ones. Correspondingly, with the above class, the addrPatient address determines the address within the network and the alphanumeric determine the name of the respective patient. In addition, it accepts two uint values, age and bloodPressure, which correspond to the age and blood pressure of the patient. Finally, an alphanumeric healthProblem is introduced that corresponds to the patient's health problem if existed.



```
mapping (address => User) userList;
mapping (address => Patient) patientList;
```

Picture 5.18 Mapping userList and patientList.

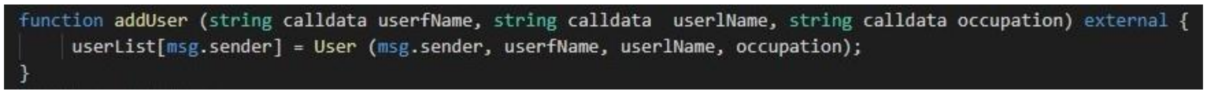
Picture 5.9 shows two mapping that serve as indexes for storing data. They get an address that is unique to the network and associates it with an object. The two-mapping named userList and patientList names that correspond to the User and Patient object structures mentioned above.



```
mapping (address => mapping (address => bool)) permited;
```

Picture 5.19 Mapping permitted.

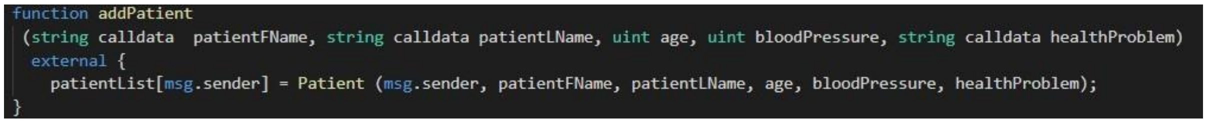
Picture 5.10 shows the nested mapping permitted which uses two addresses and relates them by storing a Boolean value. This practically means that if a condition applies between the addresses, the true value is stored while otherwise the false value is stored. In this particular contract, the condition under review is whether the first address has allowed the second to have access to its data.



```
function addUser (string calldata userfName, string calldata  userlName, string calldata occupation) external {
    userList[msg.sender] = User (msg.sender, userfName, userlName, occupation);
}
```

Picture 5.20 AddUser function.

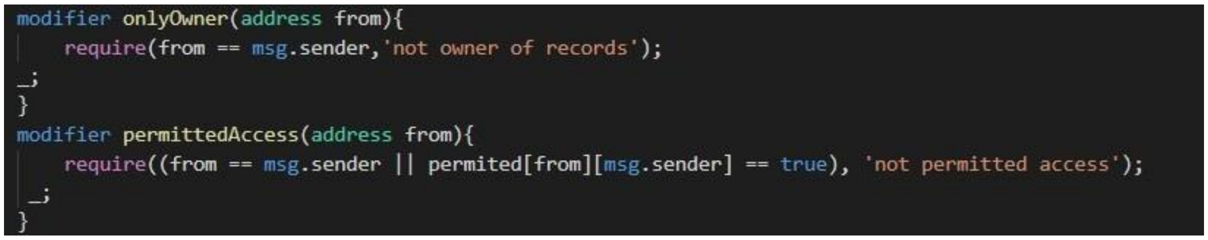
Picture 5.11 explains the addUser function, which allows new users to be introduced to the network. The function takes as parameters three alphanumeric which is the name of the new user and his occupation. The function creates a new object of the User structure. The corresponding item stores the alphanumeric as well as the address from which the function was called with the bound word msg. sender. Finally, the new user object is stored in the mapping userList and correlated with its corresponding address.



```
function addPatient
 (string calldata  patientFName, string calldata patientLName, uint age, uint bloodPressure, string calldata healthProblem)
  external {
    patientList[msg.sender] = Patient (msg.sender, patientFName, patientLName, age, bloodPressure, healthProblem);
}
```

Picture 5.21 AddPatient function.

Picture 5.12 shows the addPatient function which serves to introduce new patients into the network. The function works like the above function but creates a new Patient type object and saves it to the mapping patientList depending on its address it was called (msg. sender).



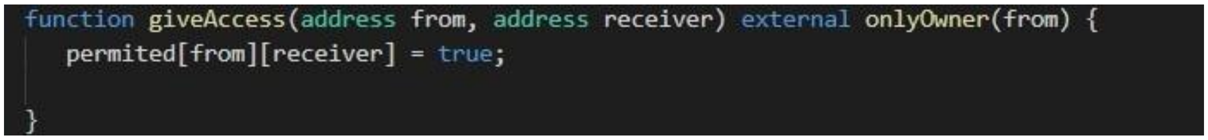
```
modifier onlyOwner(address from){
    require(from == msg.sender,'not owner of records');
_;
}
modifier permittedAccess(address from){
    require((from == msg.sender || permited[from][msg.sender] == true), 'not permitted access');
 _;
}
```

Picture 5.22 Modifier onlyOwner and permittedAccess.

Picture 5.13 lists the modifier onlyOwner, which ensures that only the data owner has the ability to manage third-party access to them. The modifier receives input an address from and checks if it is the owner of the data. The condition is checked if the address from is the same as msg. sender "from == msg. sender". This ensures that when an address calls a function that this modifier has, it can only modify data that belongs to that address. If the condition does not apply, it does not allow the function to be executed and displays the message 'not owner of data'.

Picture 5.13 also shows the modified permitted modifier which ensures patient data from unauthorized access. The modifier receives as parameter an address (from) and check whether the address from which the function was called the owner or has been granted access by the owner. This is achieved by checking if 'from == msg. sender' or if 'permitted [from] [msg. sender] == true' if one of the two conditions are true then access is authorized. If none of the conditions are true then the function is not executed and the message 'not permitted access' is displayed.

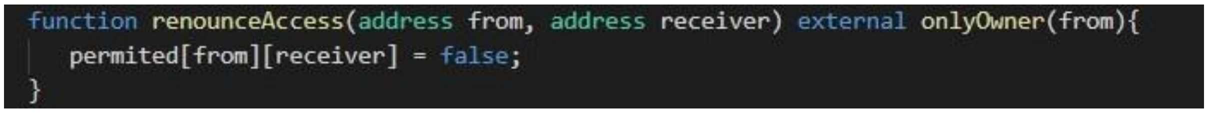
Both of the modifiers end with the command _; which states that if the condition under consideration is correct then the execution of the code continues. That is, the function that the modifier uses or in case a second modifier is used the second modifier executed.



```
function giveAccess(address from, address receiver) external onlyOwner(from) {
    permited[from][receiver] = true;

}
```

Picture 5.23 GiveAccess function.

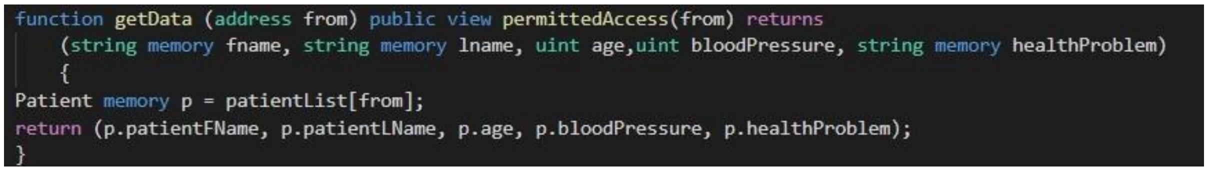
Picture 5.14 shows the giveAccess function which allows the user to provide access to his data when he seems necessary. The function accepts two addresses (from, receiver) as input and uses the modifier onlyOwner. The function allows the address from to provide access to the receiver address of personal data. Once the modifier onlyowner is used, only if the address from and the address calling the function are the same can the specified processes be executed. Access is granted by the command permitted [from] [receiver] == true.



```
function renounceAccess(address from, address receiver) external onlyOwner(from){
    permited[from][receiver] = false;
}
```

Picture 5.24 RenounceAcccess function.

Picture 5.15 shows the renounceAcccess function that allows the user to renounce the access has provided. The function works similar to giveAccess, receives two addresses (from, receiver) and contains the modifier onlyowner. Contrary to the above function, the mapping permitted [from] [receiver] sets the value to false. In this way the address from recalls the access it has offered to the receiver address.



```
function getData (address from) public view permittedAccess(from) returns
    (string memory fname, string memory lname, uint age,uint bloodPressure, string memory healthProblem)
    {
Patient memory p = patientList[from];
return (p.patientFName, p.patientLName, p.age, p.bloodPressure, p.healthProblem);
}
```

Picture 5.25 getData function.

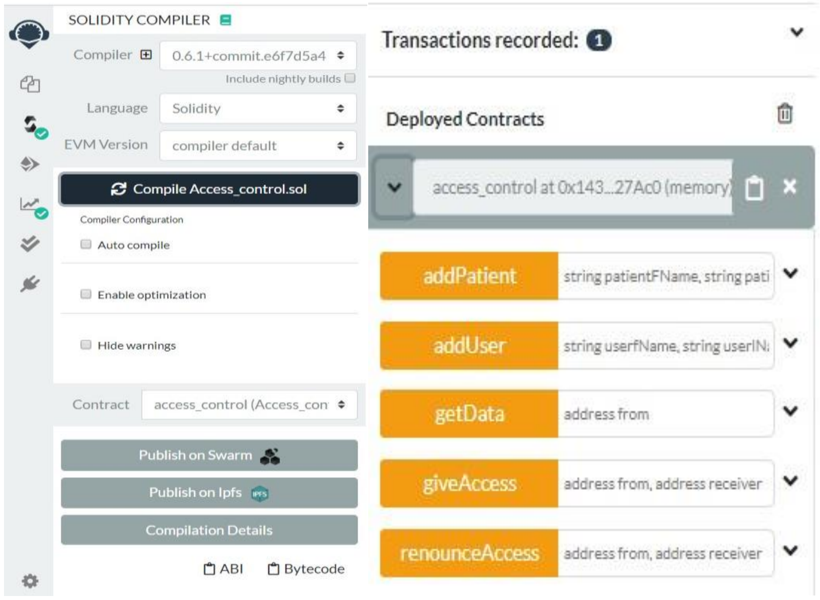
Picture 5.16 shows the getData function used to extract and display the patient's data. The function takes an address as input and uses the modified permittedAccess to ensure that data is exported by authorized users. If the modifier conditions are met, the function is executed normally and the patient data is exported and displayed. Initially the function creates a temporary Patient object called p and stores the address data from patientList. Through the created object it extracts and presents the data corresponding to the address it received as input.

The contract developed presents a proposed solution for cases of access control to patient’s medical data. It has been implemented as a useful tool in the "hands" of patients participating in a network for data management. However, it is in an experimental stage and this results in some functional assumptions being required for the system to perform properly. Initially, users and patients can only have one object input associated with their address. If they attempt a new data entry this means that the previous one is automatically deleted from the network and is no longer available. According to the method of execution of the contract, only the patients of the network have the option to offer and access the data they have in the network. This is because it is not necessary for other users participating in the network to have this option. Respectively, the appearance and extraction of data is done only by addresses of patients. User can extract patient records but not user record stored in the network.

5.2.2 AccessControl smart contract results

The Remix IDE tool will be used to check the functions and results of the access_control contract. The tool creates a virtual Ethereum network with five addresses with a virtual balance of 100 Ether. The user must first enter the code of the contract and then perform the compile and deploy processes. If the procedures do not present any error, mainly syntactic, then the user has access to all the functions of the contract and the results. The contract is executed as specified and operates similarly to an Ethereum network.

For the specific example, after the contract is put into operation, then a new patient and a new user will be introduced into the system with the corresponding data. The user will then, unsuccessfully, attempt to read the data that has just been entered without the patient's approval. Then the patient will give access to his data to the respective user, now the user can read the patient's data which he will attempt. Finally, the patient will revoke the access granted to the user then the user even if he tries to read the patient's data again will fail. In the present example we consider the address 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c as a patient function, while the address 0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C as the user address.


SOLIDITY COMPILER

Compiler: 0.6.1+commit.e6f7d5a4

Include nightly builds

Language: Solidity

EVM Version: compiler default

Compile Access_control.sol

Compiler Configuration

- Auto compile
- Enable optimization
- Hide warnings

Contract: access_control (Access_con

Publish on Swarm

Publish on Ipfs

Compilation Details

ABI Bytecode

Transactions recorded: 1

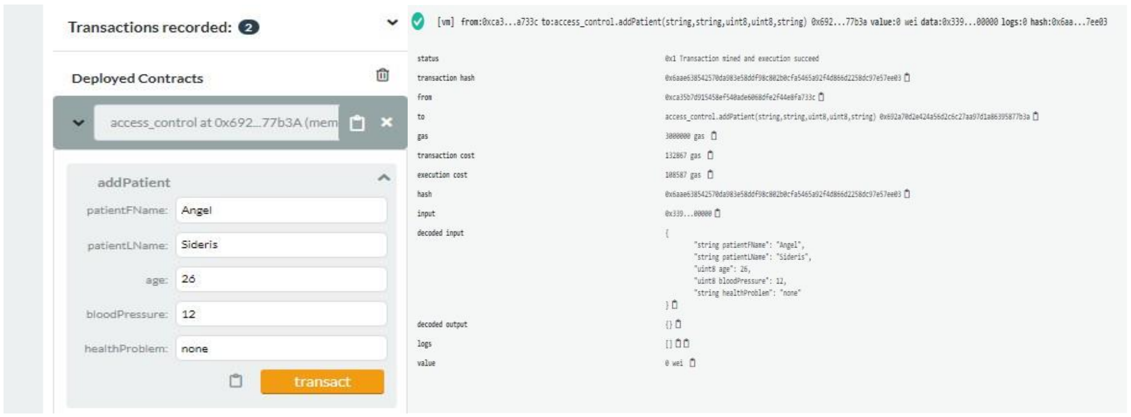
Deployed Contracts

access_control at 0x143...27Ac0 (memory)

addPatient	string patientFName, string pati
addUser	string userfName, string userlN
getData	address from
giveAccess	address from, address receiver
renounceAccess	address from, address receiver

Picture 5.26 Successful compile process and Access_control contract functions.

Picture 5.17 shows the successful compile process of the access_control contract, while the functions implemented by the contract are also presented. The addPatient and addUser functions are used to introduce new patients and new users into the system. While the giveAccess and renounceAcccess functions allow the patient to provide and renounce access to the data he holds. Finally, the getData function extracts patient data according to its called parameters. Users can use the functions at will.


Transactions recorded: 2

Deployed Contracts

access_control at 0x692...77b3A (mem

addPatient

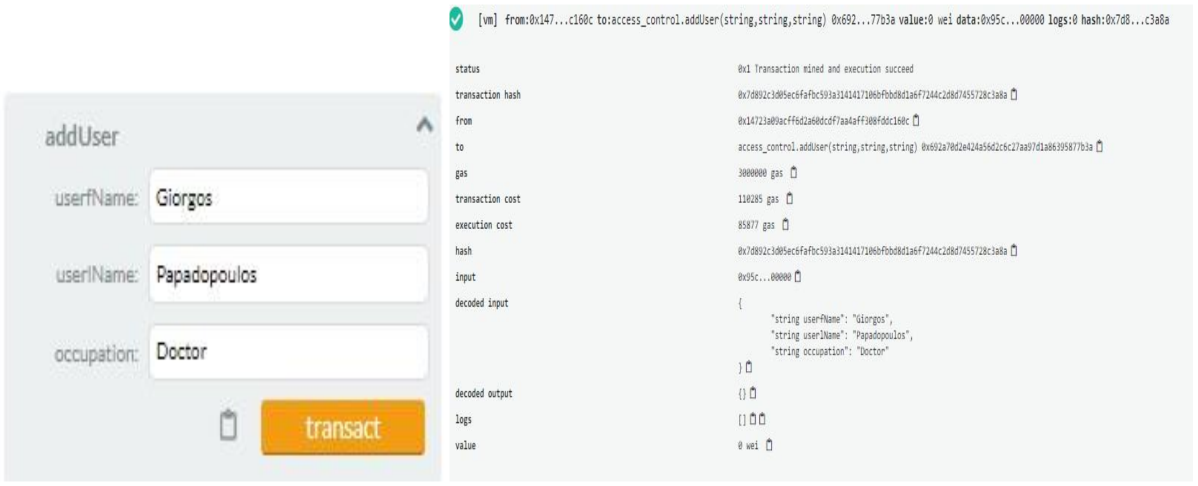
patientFName:	Angel
patientLName:	Sideris
age:	26
bloodPressure:	12
healthProblem:	none

transact

```
[vm] from:0xca3...a733c to:access_control.addPatient(string,string,uint8,uint8,string) 0x692...77b3a value:0 wei data:0x339...00000 logs:0 hash:0x6aa...7ee03

status              0x1 Transaction mined and execution succeed
transaction hash    0x6aae638542570da983e58ddf98c882b0cfa5465a92f4d866d2258dc97e57ee03
from                0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to                  access_control.addPatient(string,string,uint8,uint8,string) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
gas                 3000000 gas
transaction cost    132867 gas
execution cost      108587 gas
hash                0x6aae638542570da983e58ddf98c882b0cfa5465a92f4d866d2258dc97e57ee03
input               0x339...00000
decoded input       {
                        "string patientFName": "Angel",
                        "string patientLName": "Sideris",
                        "uint8 age": 26,
                        "uint8 bloodPressure": 12,
                        "string healthProblem": "none"
                    }
decoded output      {}
logs                []
value               0 wei
```

Picture 5.27 AddPatient function call and call results.

Picture 5.18 shows the call of the addPatient function from the patient address. The data (Angel, Sideris, 26, 12, none) is entered in the respective fields (patientFName, patientLName, age, bloodpressure, healthProblem). The network records the data and associates it with the address of the patient who made the call. At the same time, the results of the call are displayed, with the entry being presented in the form of a hash, while below is the decoded form.

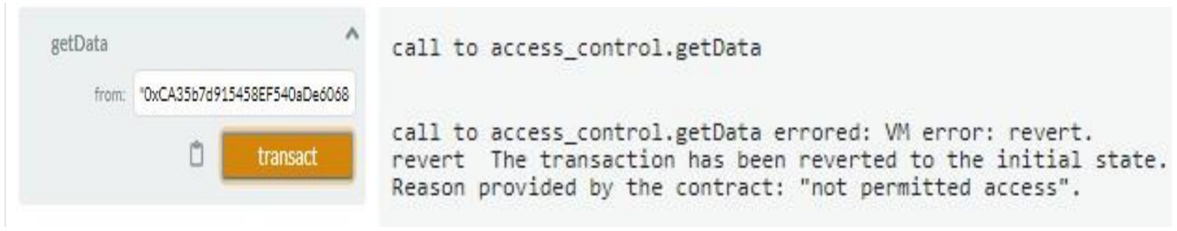


```
addUser
userfName: Giorgos
userlName: Papadopoulos
occupation: Doctor
transact
```

```
[vm] from:0x147...c160c to:access_control.addUser(string,string,string) 0x692...77b3a value:0 wei data:0x95c...00000 logs:0 hash:0x7d8...c3a8a

status              0x1 Transaction mined and execution succeed
transaction hash    0x7d892c3d05ec6fafbc593a3141417106bfbbd8d1a6f7244c2d8d7455728c3a8a
from                0x14723a09acff6d2a60dcdf7aa4aff308fddc160c
to                  access_control.addUser(string,string,string) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
gas                 3000000 gas
transaction cost    110285 gas
execution cost      85877 gas
hash                0x7d892c3d05ec6fafbc593a3141417106bfbbd8d1a6f7244c2d8d7455728c3a8a
input               0x95c...00000
decoded input       {
                            "string userfName": "Giorgos",
                            "string userlName": "Papadopoulos",
                            "string occupation": "Doctor"
                    }
decoded output      {}
logs                []
value               0 wei
```

Picture 5.28 Calling the addUser function and its results.

Picture 5.19 shows the addUser function call from the user address. The data (Giorgos, Papadopoulos, Doctor) is entered in the fields (userfName, userlName, occupation). The network records the data and associate the with the user address calling the function. At the same time, the results of the call are displayed, with the input being presented in the form of a hash, while below is the decoded form.

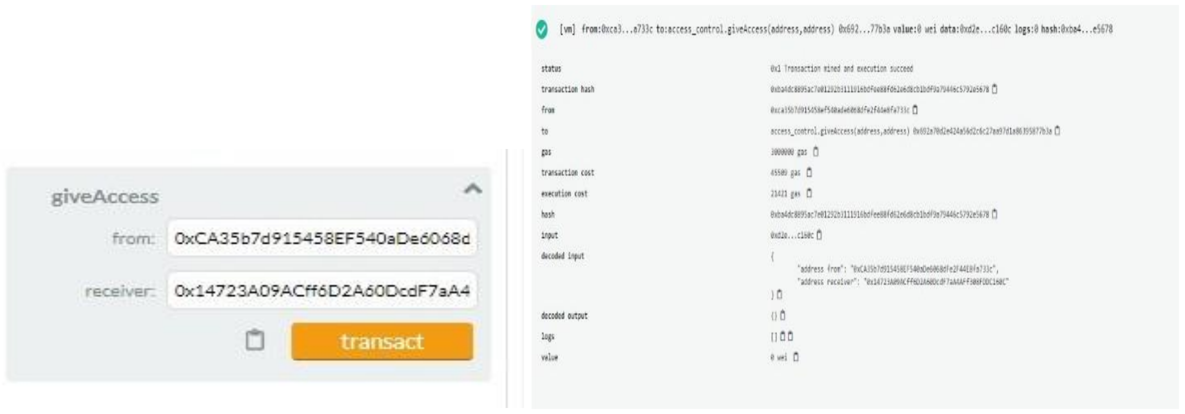


```
getData
from: "0xCA35b7d915458EF540aDe6068
transact
```

```
call to access_control.getData

call to access_control.getData errored: VM error: revert.
revert  The transaction has been reverted to the initial state.
Reason provided by the contract: "not permitted access".
```

Picture 5.29 Unsuccessful getData call and error message.

In Picture 5.20 the user tries to read the patient's data without the required authorization. The result is that the function does not display the data but displays the error message "not permitted access".

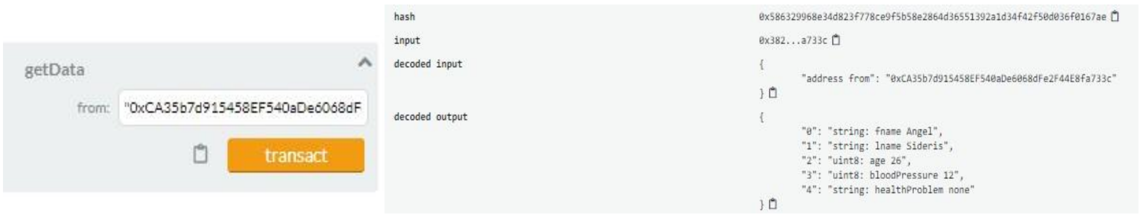


```
giveAccess
from: 0xCA35b7d915458EF540aDe6068d
receiver: 0x14723A09ACff6D2A60DcdF7aA4
transact
```

```
[vm] from:0xca3...a733c to:access_control.giveAccess(address,address) 0x692...77b3a value:0 wei data:0xd2e...c160c logs:0 hash:0xba4...e5678

status              0x1 Transaction mined and execution succeed
transaction hash    0xba4dc8895ac7e01292b3111916bdfee88fd62e6d8cb1bdf9a79446c5792e5678
from                0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to                  access_control.giveAccess(address,address) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
gas                 3000000 gas
transaction cost    45509 gas
execution cost      21421 gas
hash                0xba4dc8895ac7e01292b3111916bdfee88fd62e6d8cb1bdf9a79446c5792e5678
input               0xd2e...c160c
decoded input       {
                            "address from": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c",
                            "address receiver": "0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C"
                    }
decoded output      {}
logs                []
value               0 wei
```

Picture 5.30 GiveAccess function call and results.

Picture 5.21 calls the giveAccess function from the patient with parameters of both addresses. The result is that the user now has access to the patient's data.

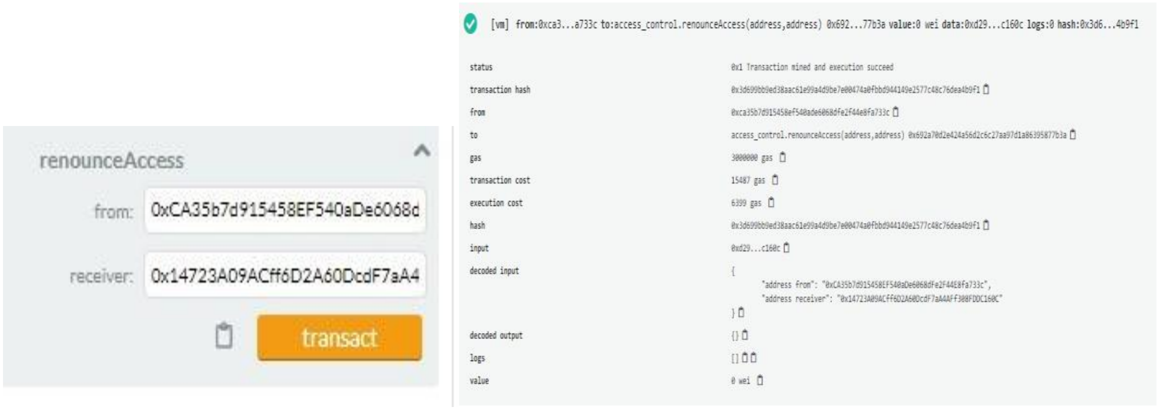


```
"0xCA35b7d915458EF540aDe6068dF
```

```
hash            0x586329968e34d823f778ce9f5b58e2864d36551392a1d34f42f50d036f0167ae
input           0x382...a733c
decoded input   {
                    "address from": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c"
                }
decoded output  {
                    "0": "string: fname Angel",
                    "1": "string: lname Sideris",
                    "2": "uint8: age 26",
                    "3": "uint8: bloodPressure 12",
                    "4": "string: healthProblem none"
                }
```

Picture 5.31 Successful getData call from the user and display of patient data.

Picture 5.22 shows the second call of the getData function with parameter the patient address by the user after performing the giveAccess function. After the user has provided access to the patient's data, the call is successful. The call results are displayed in hash format and then the output and input are decoded. Patient data is now displayed to the user.

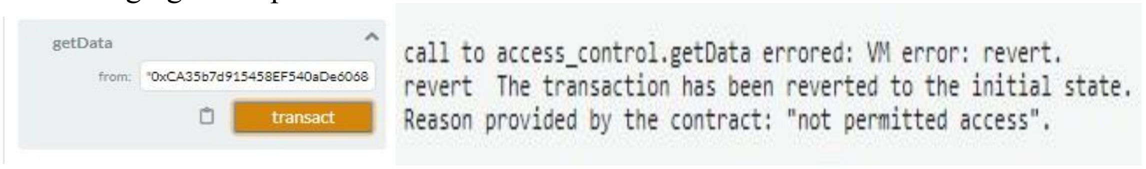


```
0xCA35b7d915458EF540aDe6068d
0x14723A09ACff6D2A60DcdF7aA4
```

```
[vm] from:0xca3...a733c to:access_control.renounceAccess(address,address) 0x692...77b3a value:0 wei data:0xd29...c160c logs:0 hash:0x3d6...4b9f1

status              0x1 Transaction mined and execution succeed
transaction hash    0x3d699bb9ed38aac61e99a4d9be7e00474a0fbbd944149e2577c48c76dea4b9f1
from                0xca35b7d915458ef540ade6068dfe2f44e8fa733c
to                  access_control.renounceAccess(address,address) 0x692a70d2e424a56d2c6c27aa97d1a86395877b3a
gas                 3000000 gas
transaction cost    15487 gas
execution cost      6399 gas
hash                0x3d699bb9ed38aac61e99a4d9be7e00474a0fbbd944149e2577c48c76dea4b9f1
input               0xd29...c160c
decoded input       {
                        "address from": "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c",
                        "address receiver": "0x14723A09ACff6D2A60DcdF7aA4AFf308FDDC160C"
                    }
decoded output      {}
logs                []
value               0 wei
```

Picture 5.32 call the renounceAcccess function and the results.

Picture 5.23 calls the renounceAcccess function from the patient with the parameters of the two addresses. The result of the function call is that the user no longer has access to the data belonging to the patient.



```
"0xCA35b7d915458EF540aDe6068
```

```
call to access_control.getData errored: VM error: revert.
revert  The transaction has been reverted to the initial state.
Reason provided by the contract: "not permitted access".
```

Picture 5.33 call the getData function for the 3rd time and the results.

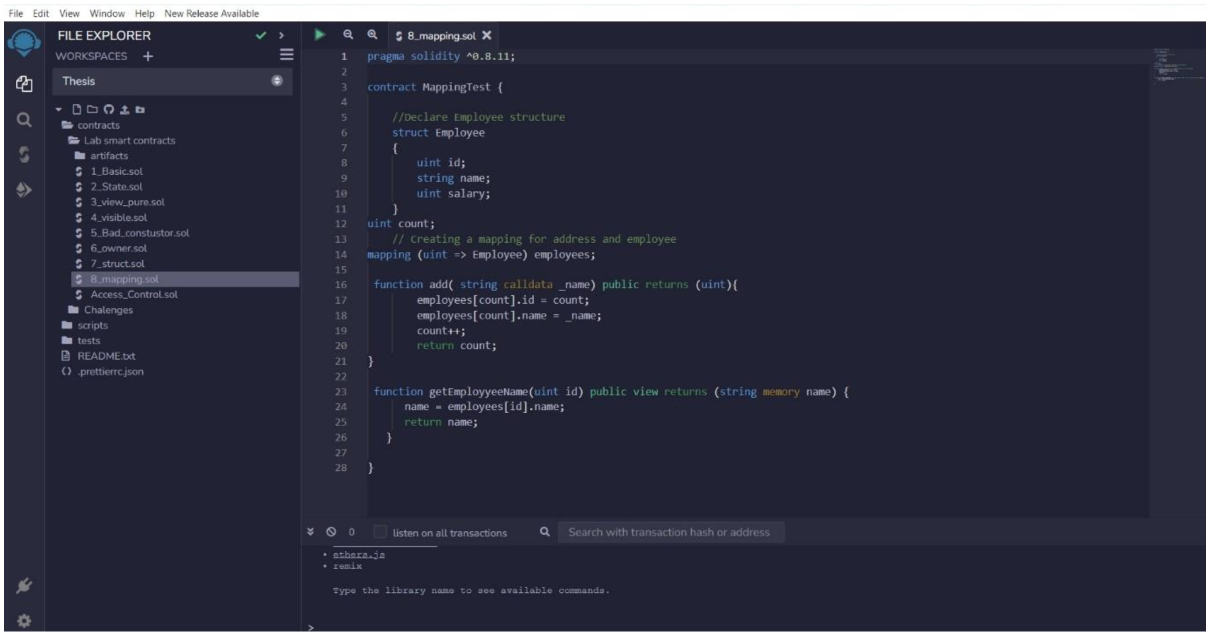
Picture 5.24 shows the user receiving the getData function with the patient address as a parameter after performing the renounceAcccess function. The user no longer has access to the patient's data, so calling the function is unsuccessful and the message "not permitted access" appears.

5.3 Remix Tool toturial

Below will be analysed the basic steps for using the Remix tool which will be used to prepare the work. The tool is used to write and run smart contract on a virtual network locally and at no cost.

5.3.1 Smart contract panel

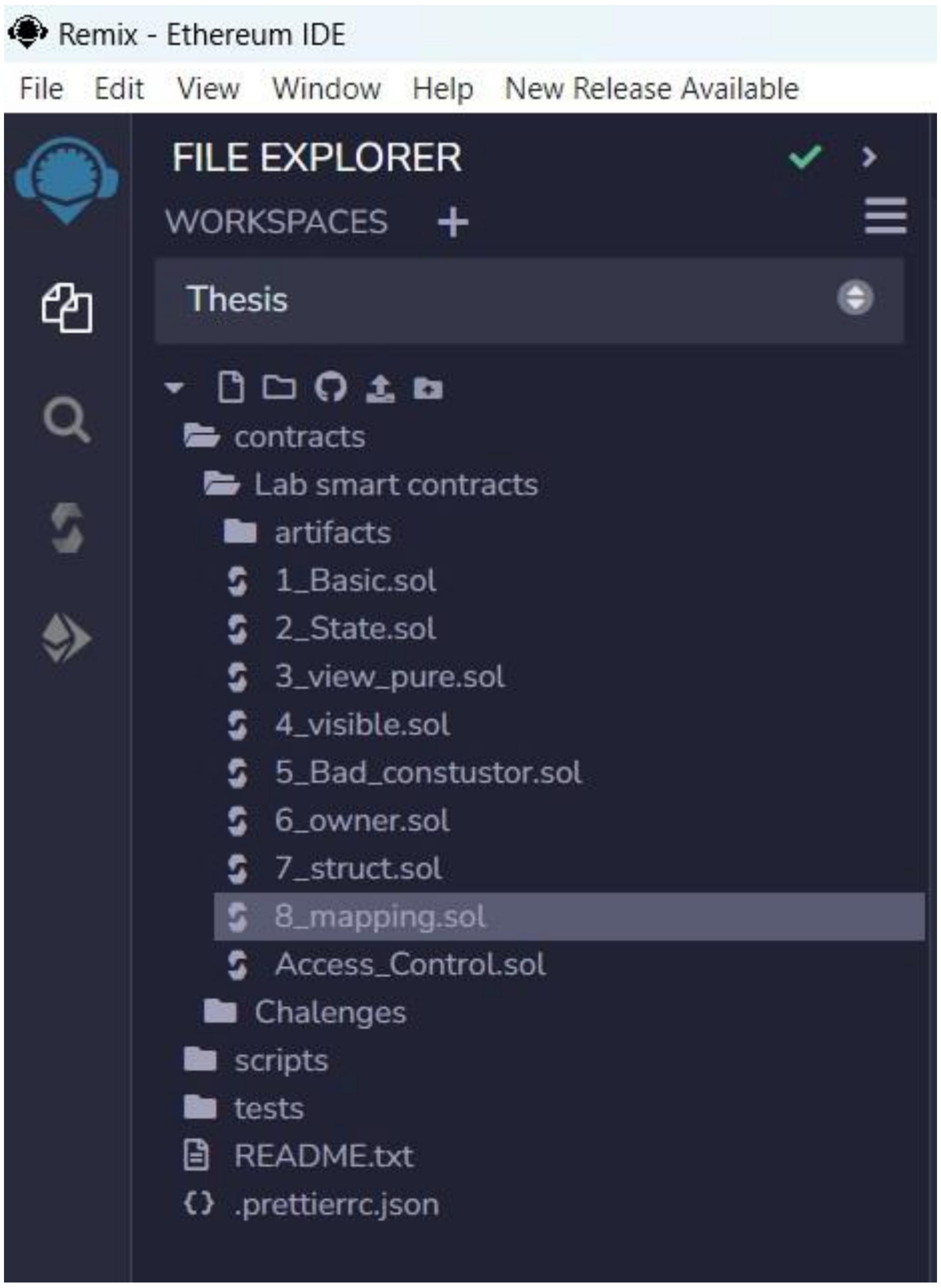
smart code contracts in the right field while on the left you will be able to see the smart ones contracts that exist in the specific workspace . The photographs below show an example similar to what you will have in the workspace provided to you with the smart contract for your work.


File Edit View Window Help New Release Available
FILE EXPLORER
WORKSPACES
Thesis
8_mapping.sol
listen on all transactions
Search with transaction hash or address

```
contracts
  Lab smart contracts
    artifacts
    1_Basic.sol
    2_State.sol
    3_view_pure.sol
    4_visible.sol
    5_Bad_constustor.sol
    6_owner.sol
    7_struct.sol
    8_mapping.sol
    Access_Control.sol
  Chalenges
scripts
tests
README.txt
.prettierrc.json
```

```solidity
pragma solidity ^0.8.11;

contract MappingTest {

    //Declare Employee structure
    struct Employee
    {
        uint id;
        string name;
        uint salary;
    }
uint count;
    // Creating a mapping for address and employee
mapping (uint => Employee) employees;

 function add( string calldata _name) public returns (uint){
        employees[count].id = count;
        employees[count].name = _name;
        count++;
        return count;
}

 function getEmployyeeName(uint id) public view returns (string memory name) {
      name = employees[id].name;
      return name;
   }

}
```

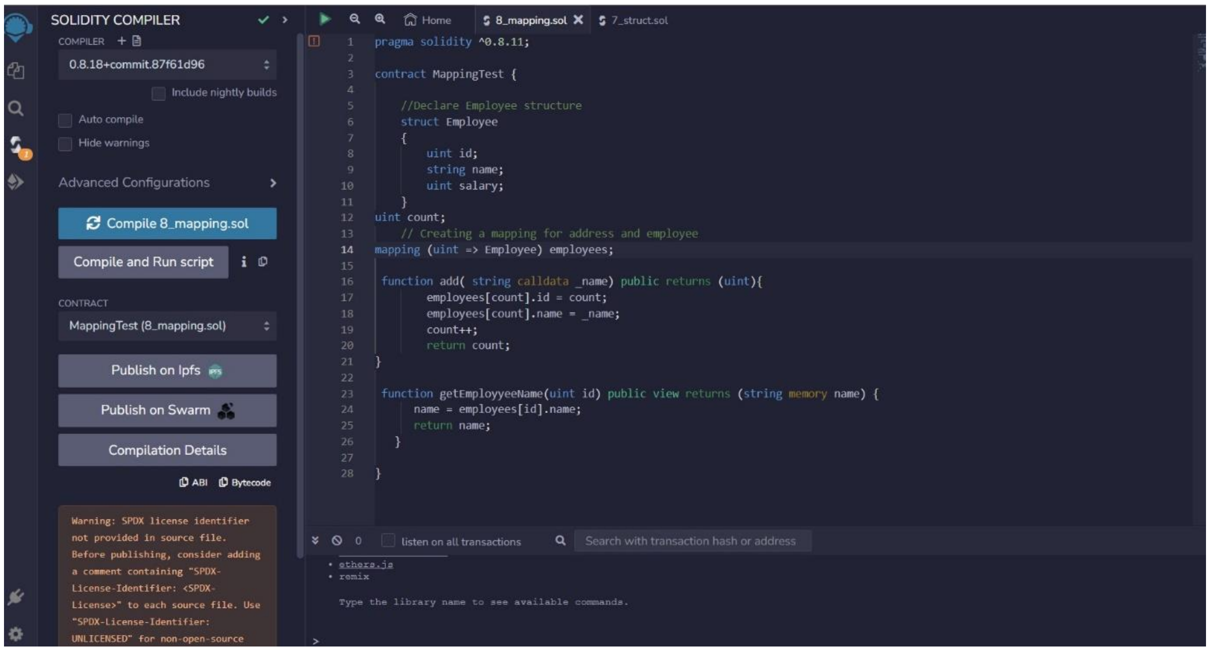
```
0
• ethers.js
• remix

  Type the library name to see available commands.

>
```



```
contracts
  Lab smart contracts
    artifacts
    1_Basic.sol
    2_State.sol
    3_view_pure.sol
    4_visible.sol
    5_Bad_constustor.sol
    6_owner.sol
    7_struct.sol
    8_mapping.sol
    Access_Control.sol
  Chalenges
scripts
tests
README.txt
.prettierrc.json
```

5.3.2 Compiler Smart contract

In this window you can compile the smarts contract and check if there is any error in the code you have written, any errors that may exist with an explanation of the type of error (only errors in red do not allow the compile process to be completed).



COMPILER: 0.8.18+commit.87f61d96

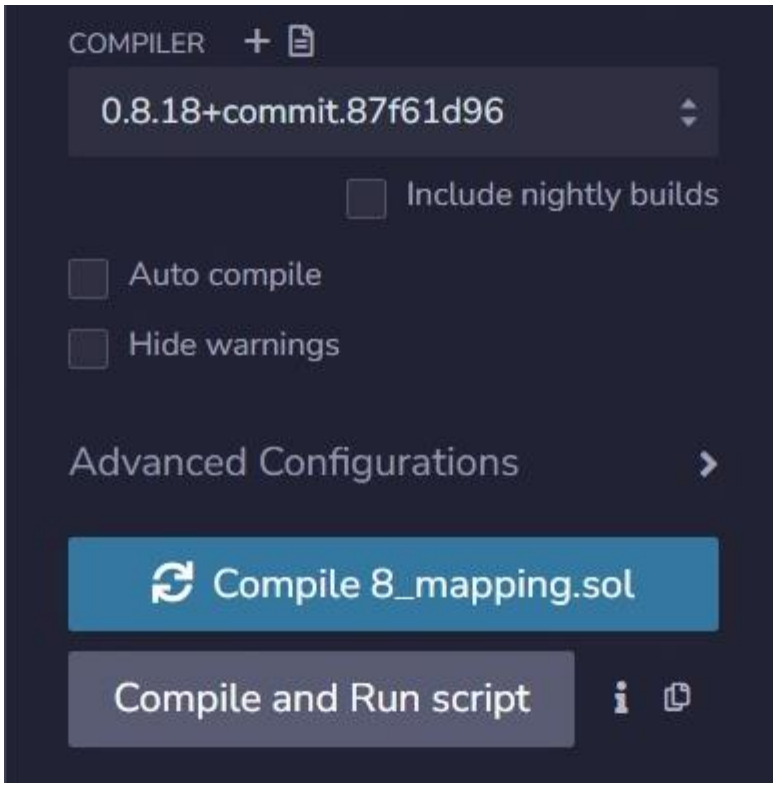
CONTRACT: MappingTest (8_mapping.sol)

```
Warning: SPDX license identifier
not provided in source file.
Before publishing, consider adding
a comment containing "SPDX-
License-Identifier: <SPDX-
License>" to each source file. Use
"SPDX-License-Identifier:
UNLICENSED" for non-open-source
```

```solidity
pragma solidity ^0.8.11;

contract MappingTest {

    //Declare Employee structure
    struct Employee
    {
        uint id;
        string name;
        uint salary;
    }
uint count;
    // Creating a mapping for address and employee
mapping (uint => Employee) employees;

 function add( string calldata _name) public returns (uint){
        employees[count].id = count;
        employees[count].name = _name;
        count++;
        return count;
}

 function getEmployyeeName(uint id) public view returns (string memory name) {
      name = employees[id].name;
      return name;
   }

}
```

```
• ethers.js
• remix

  Type the library name to see available commands.

>
```

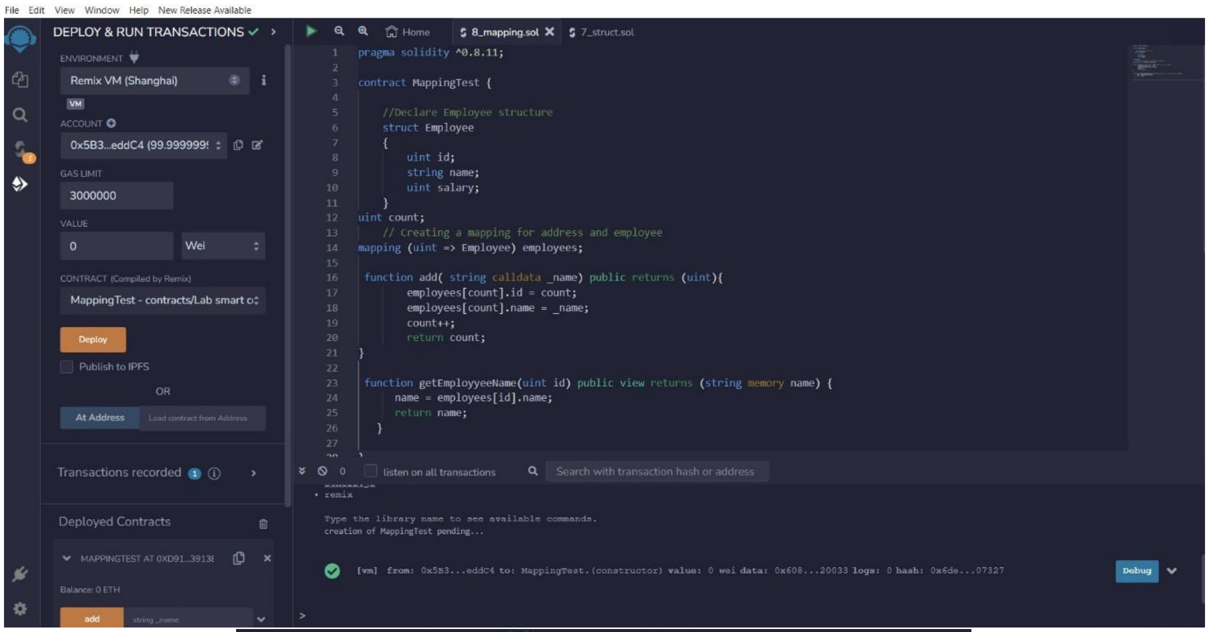
In the photo below if the blue button is pressed the process will be done is the first step you will need for the test process. At the top you can change the compiler version according to what is written in the first line of the code given to you.


COMPILER

0.8.18+commit.87f61d96

5.3.3 Deploy Smart contract

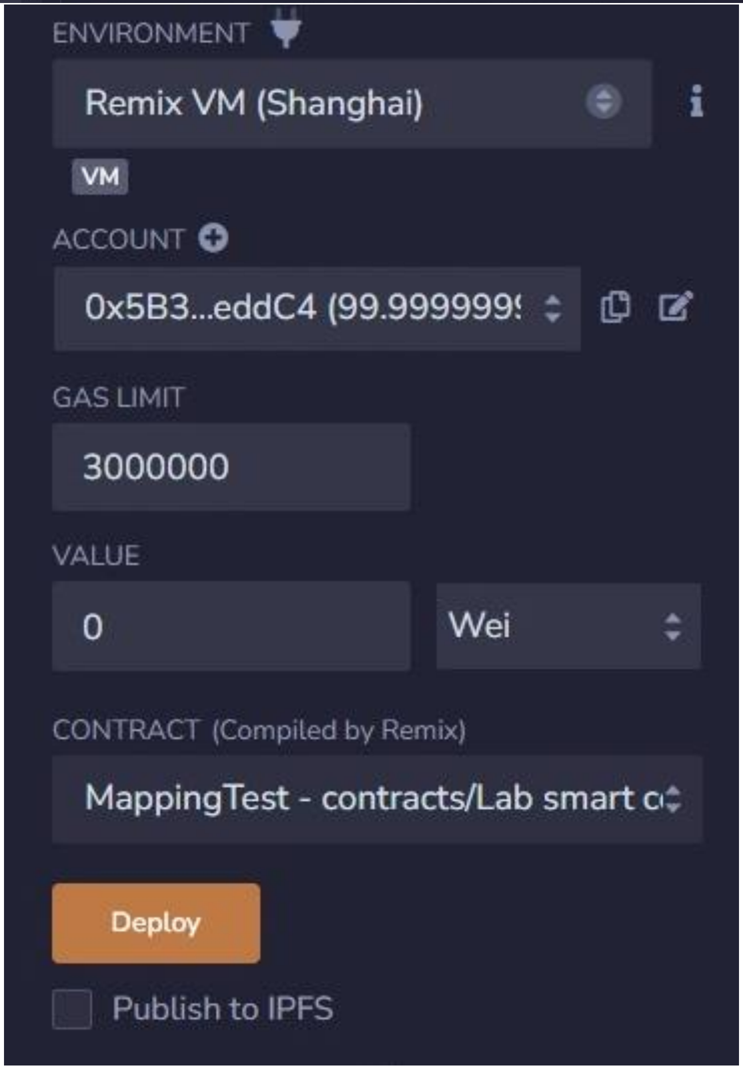
In this window you can deploy the smarts contract and execute the transactions you wish.



ENVIRONMENT: Remix VM (Shanghai)
ACCOUNT: 0x5B3...eddC4 (99.999999
GAS LIMIT: 3000000
VALUE: 0 Wei
CONTRACT (Compiled by Remix): MappingTest - contracts/Lab smart c

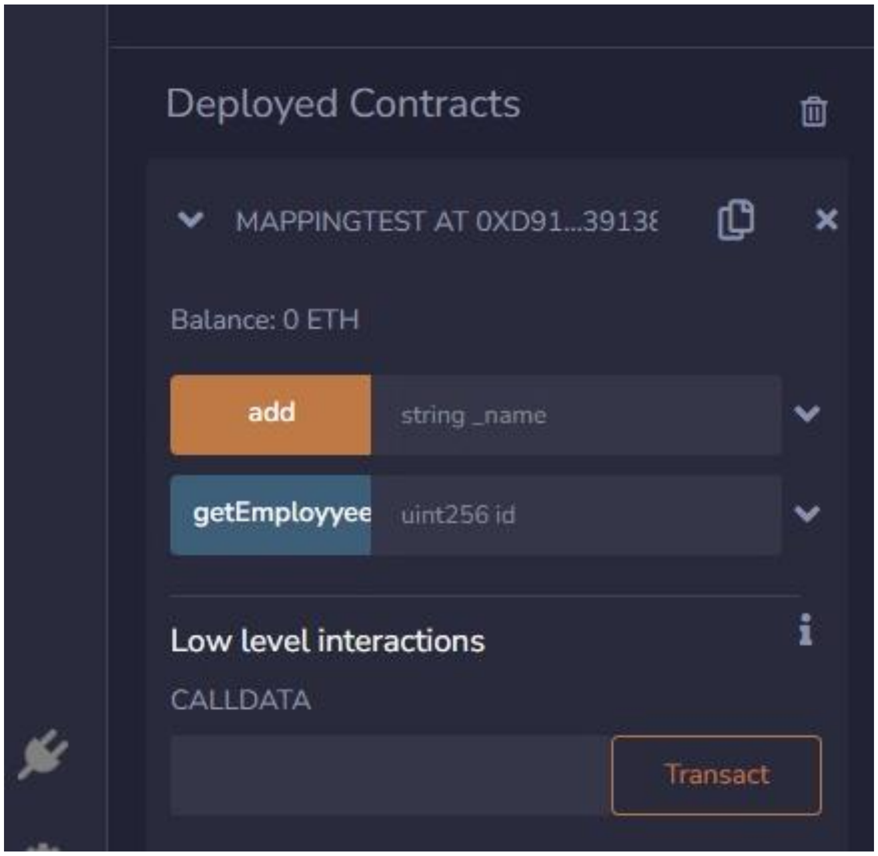
```
pragma solidity ^0.8.11;

contract MappingTest {

    //Declare Employee structure
    struct Employee
    {
        uint id;
        string name;
        uint salary;
    }
uint count;
    // Creating a mapping for address and employee
mapping (uint => Employee) employees;

 function add( string calldata _name) public returns (uint){
        employees[count].id = count;
        employees[count].name = _name;
        count++;
        return count;
}

 function getEmployyeeName(uint id) public view returns (string memory name) {
      name = employees[id].name;
      return name;
   }

```

```
• remix

Type the library name to see available commands.
creation of MappingTest pending...

[vm] from: 0x5B3...eddC4 to: MappingTest.(constructor) value: 0 wei data: 0x608...20033 logs: 0 hash: 0x6de...07327
>
```



ENVIRONMENT: Remix VM (Shanghai)
VM
ACCOUNT: 0x5B3...eddC4 (99.999999
GAS LIMIT: 3000000
VALUE: 0 Wei
CONTRACT (Compiled by Remix): MappingTest - contracts/Lab smart c

The above photo shows the fields related to the costs of the transactions to be executed (gas limit and the value contained in each wei unit). In addition, Remix enables the operation of a virtual Blockchain network and provides you with ten accounts with 100 ether each to perform transactions simulating a normal Blockchain network. You can select and toggle accounts presented using a dropdown field.



Once the process is complete, you will see deployed in the field contracts to create a copy of smart contract you deployed. In this way you have access to all its functions, you will be able to call the functions that are implemented by pressing the corresponding button. In the image above there is the add function which accepts a name parameter which when called creates a new user with the corresponding name. For this example smart has been used contracts that have been presented in class and you have access to their code from the lab material. As I mentioned above, it is possible to switch accounts, so depending on which account you choose, it will be the one that performs the corresponding transaction, when you do the deploy process, the account you choose is the one you consider the owner. The last image shows the results of the above successful transaction.

Conclusion

This thesis attempts to familiarize the reader with how the Ethereum platform (based on Blockchain technology) works and how to develop smart contracts. In particular, the security of the platform and the measures applied against attacks are mentioned. In addition, a detailed analysis is performed on the access control to patient's medical data within a Blockchain network. The elaboration of the thesis is achieved by the development of a smart contract environment that allows the user to familiarize with basic operation of solidity smart contract. The created smart contracts aim to create basic knowledge about security and operation of Ethereum network.

The contracts implemented has a beneficial effect on the educational purposes of the analyzed methods. At the same time, there are improvements needed to make it more efficient and complete. Initially, the creation of an interface that allows direct communication with Web3.0 will be beneficial on a more hands on educational methods. In addition, the most important improvement proposed is introduction on frameworks like truffle and ganache, with the use of this framework creates a more complete environment between user and the blockchain network. After this inclusion on the environment there will be a space to explore direct attacks on the network nodes and the blockchain network

Bibliographical references

1. https://koumentakislaw.gr/arthra/Blockchain-epanastash-stis-synallages/
2. https://www.dummies.com/personal-finance/the-structure-of-Blockchains/
3. https://ccoingossip.com/advantages-and-disadvantages-of-Blockchain/
4. https://medium.com/@MLSDevCom/Blockchain-architecture-basics-components-structure-benefits-creation-beace17c8e77
5. https://mlsdev.com/blog/156-how-to-build-your-own-Blockchain-architecture
6. https://www.coinspeaker.com/everyday-uses-Blockchain/
7. https://ambisafe.com/blog/smart-contracts-10-use-cases-business/
8. https://www.apriorit.com/dev-blog/578-Blockchain-attack-vectors
9. http://sebfor.com/what-are-smart-contracts-and-how-do-they-work-examples/
10. https://searchcompliance.techtarget.com/definition/smart-contract
11. https://blockgeeks.com/guides/smart-contracts/
12. https://medium.com/nakamo-to/breaking-down-the-smart-contract-45b249b8bc71
13. https://medium.com/coinbundle/smart-contracts-whats-the-big-idea-149048e41d9f#880b

14. https://atozmarkets.com/news/pros-and-cons-of-smart-contracts/
15. https://101Blockchains.com/smart-contracts/
16. https://legalvision.com.au/smart-contracts-legally-binding/
17. https://medium.com/@preethikasiredired/how-does-ethereum-work-anyway-22d1df506369
18. https://medium.com/bethereum/ethereum-consensus-and-scalability-Blockchain-series-part-iii-4acd78d0eb41
19. http://www.gjermundbjaanes.com/understanding-ethereum-smart-contracts/
20. https://jaxenter.com/cryptographic-hashing-secure-Blockchain-149464.html
21. https://jaxenter.com/cryptographic-hashing-secure-Blockchain-149464.html
22. https://medium.com/coinmonks/Blockchain-cryptography-and-hashing-in-a-nutshell-23f4fd7c77b6
23. https://www.dataversity.net/Blockchain-can-used-secure-sensitive-data-storage/#
24. https://blockgeeks.com/smart-contract-security-audits/
25. https://medium.com/talo-protocol/how-to-secure-sensitive-data-on-an-ethereum-smart-contract-77f21c2b49f5
26. https://www.Blockchain-council.org/Blockchain/how-to-store-data-on-ethereum-Blockchain/
27. https://www.entrepreneur.com/article/318477
28. https://hackernoon.com/how-you-can-secure-your-data-with-Blockchain-539655295b70
29. https://hackernoon.com/smart-contracts-part-1-the-state-of-security-4c37988c770
30. https://hackernoon.com/databases-and-Blockchains-the-difference-is-in-their-purpose-and-design-56ba6335778b
31. http://www.prasannapattam.com/2017/08/data-types-in-ethereum-solidity-smart.html
32. https://www.c-sharpcorner.com/article/variables-and-types-in-solidity/
33. https://backstage.forgerock.com/docs/am/6/uma-guide/
34. https://medium.com/@dewni.matheesha/user-managed-access-uma-2-0-0-bcecb1d535b3
35. https://wso2.com/library/article/2018/12/a-quick-guide-to-user-managed-access-2-0/
36. https://medium.com/coinmonks/solidity-tutorial-returning-structs-from-public-functions-e78e48efb378
37. https://www.beckershospitalreview.com/healthcare-information-technology/5-ways-Blockchain-could-improve-healthcare.html