



FINAL PROYECT
DEGREE IN COMPUTER ENGINEERING
MENTION ON INFORMATION TECHNOLOGIES



Certification of IoT elements using the blockchain

Student: Sergio Valle Trillo

Director: Antonino Santos del Riego

A Coruña, September 2022.

Dedicatoria a todas las personas que me han apoyado durante este proceso

Acknowledgements

A Hugo, a mi familia y a Silvia que me han ayudado en el transcurso de este trabajo y sobre todo el grado. A mi hermano, por haberme hecho mas ameno los días de trabajo duro del trabajo. A mi familia por haberme brindado las oportunidades de convertirme en la persona que soy ahora mismo. A Silvia por la ayuda y apoyo que me ha dado en los momentos duros.

Abstract

The non-fungible tokens have been widely used to prove ownership of art and gaming collectibles and used as utility tokens. The use of this tokens in this work is to represent the ownership of the internet of things devices from the manufacturing phase, in the distributed and decentralized public ledger. This physical devices will have attached a token that represent them in the blockchain and the possession of an owner by an unique identifier. Hence, the devices are identified by their public blockchain address and their token that associates them to their owner. Besides, this address allow the Internet of Things devices to participate in the network and establish a shared secret between owner and device. This work, proposes to use the physical unclonable functions to establish a noose between the physical world and the blockchain by deriving the private key of the blockchain address from the physical unclonable functions response. This link is difficult to tamper and can be traced during the lifetime of the token. Moreover, there is no need of using a security module or similar to store the key since the physical unclonable functions response is generated each the private key is needed so that it not stored in a non volatile memory. Once we have the shared secret this are used to cipher the certificates that will be deployed by the owner of the devices on a decentralized storage blockchain like FileCoin or the InterPlanetary File System. This certificates are used to communicate with other devices using standard protocols like Transport Layer Security or Datagram Transport Layer Security. An API called Powergate, is part of the infrastructure of certification of the Internet of Things elements, providing communication with the decentralized storage blockchains.

Resumo

Os tokens non funxibles utilízanse amplamente para demostrar a propiedade de obxectos de colección de arte e xogos e utilízanse como "utility tokens". O uso destes tokens neste traballo é para representar na rede distribuído e descentralizado que é a blockchain, a propiedade dos dispositivos Internet of Things desde o mesmo momento da súa creación, é dicir, durante o proceso de manufactura. A estes dispositivos físicos achégaselles un token que os identifica na blockchain e permite representar a posesión dun propietario mediante un identificador único. Polo tanto, os dispositivos identifícanse pola súa dirección pública na cadea de bloques e o seu token é o que os asocia ao seu propietario. Ademais, esta dirección permite aos dispositivos da Internet of Things participar na rede e establecer un secreto compartido entre propietario e dispositivo. Este traballo, propón utilizar as funcións físicas non clonables para establecer un lazo entre o mundo físico e a blockchain derivando a clave privada da dirección do blockchain a partir da resposta das funcións físicas non clonables. Este vínculo é difícil de manipular e pode ser rastrexado durante a vida do token. Ademais, non é necesario utilizar un módulo de seguridade ou similar para almacenar a clave, xa que a resposta da función física non clonable é xerada durante o proceso de arranque e é guardada nunha memoria non volátil. Unha vez que teñamos o secreto compartido, este utilizarase para cifrar os certificados que serán despregados polo propietario dos dispositivos nunha blockchain de almacenamento descentralizado como FileCoin ou InterPlanetary File System. Estes certificados utilizaranse para comunicarse con outros dispositivos utilizando protocolos estándar como son Datagram Transport Layer Security y Transport Layer Security. Unha API compondrá a infraestrutura de certificación dos elementos do Internet of Things proporcionando comunicación coas blockchains de almacenamento descentralizadas.

Keywords:

Blockchain
Root of Trust
IoT Security
Non-fungible Tokens (NFTs)
Physical Unclonable Functions (PUFs)
IPFS
File Coin

Palabras chave:

Blockchain
Raíz de confianza
Seguridade do IoT
Tokens non funxibles (NTFs)
Funciones físicas non clonables (PUFs)
IPFS
File Coin

Contents

1	Introduction	1
2	The breakout of the Blockchain	5
2.1	Bitcoin Wallets	5
2.1.1	Types of wallets	6
2.1.2	Transactions	7
2.2	Creation of Bitcoin	7
2.3	How it works?	8
2.3.1	Structure of the block	9
2.3.2	The Merkle tree	10
2.4	Mining	11
2.4.1	How a candidate block is found	12
2.5	Limitations of Bitcoin	13
3	Ethereum	14
3.1	Genesis of Ethereum	15
3.1.1	The four stages of Ethereum	15
3.2	Test Networks	16
3.3	Account	18
3.4	Transaction	19
3.4.1	The Nonce	20
3.5	The Ethereum Virtual Machine	21
3.6	Gas	22
3.6.1	How the gas is consumed during execution	22
3.7	Smart Contracts	23
3.7.1	Contract deletion	25
3.7.2	EVM programming languages	25
3.7.3	Security of smart contracts	27

CONTENTS

3.7.4	Modifiers	27
3.7.5	Decorators	28
3.7.6	Most common vulnerabilities	28
3.8	Tokens	33
3.8.1	Intrinsicity of tokens	33
3.8.2	The ERC-20 standard	34
3.9	Non Fungible Tokens	36
3.9.1	The ERC-721	36
4	Decentralized storage	38
4.1	IPFS	38
4.2	FileCoin	41
4.2.1	Users	42
4.2.2	Storage miners	42
4.3	How it works	43
4.3.1	Storage providers	43
4.3.2	Deals	43
4.3.3	Proofs	43
4.3.4	Gas	44
4.4	PowerGate	45
5	Physical Unclonable Functions	46
5.1	Arbiter-based PUF	47
5.1.1	Discussing Arbiter PUFs and its compositions	48
5.2	Lightweight Secure PUF	48
5.3	Ring Oscillator PUF	48
6	Infrastructure for certification of IoT devices	50
6.1	Actors	52
6.2	How it works	53
6.2.1	Smart Contract	54
6.2.2	The code of the contract	56
6.3	The workflow of the actors	58
6.3.1	Manufacturer	59
6.3.2	Owner	59
6.4	Web Interfaces	60
6.4.1	Powergate integration	60
6.5	Security aspects	61

CONTENTS

6.6	Root of Trust	61
6.7	Description of the environment	62
6.8	Development of the project	62
6.9	Pricing	64
6.10	A possible continuation	64
7	Conclusion	65
A	User Guide	68
A.1	Manufacturer	68
A.2	Owner	71
	List of Acronyms	77
	Glossary	79
	Bibliography	80

List of Figures

2.1	Structure of the blockchain	9
2.2	Merkle Tree	10
3.1	Structure of the Ethereum blockchain	19
3.2	Ethereum storage scheme	22
3.3	Deployment of a contract	24
3.4	Reentrancy attack	30
4.1	FileCoin workflow	42
5.1	Structure of an PUF arbiter	47
5.2	Ring oscillator structure	49
6.1	Infrastructure proposed	54
6.2	States of the contract	56
6.3	Integration with Powergate	61
6.4	Spiral model Boehm	63
A.1	Log in page	68
A.2	Contract deployment	69
A.3	Contract deployment	69
A.4	Contract deployment	70
A.5	Register device	70
A.6	List devices	70
A.7	Registration of manufacturer	71
A.8	Account created	71
A.9	Change of address	71
A.10	Registration of device	72
A.11	Device's registration form	72

LIST OF FIGURES

A.12 Registration success	72
A.13 Request engagement	73
A.14 Engagement finished	73
A.15 Full engagement	74
A.16 Certificate form	75
A.17 Certificate created	75
A.18 Certificate retrieved	76

List of Tables

2.1	Specification of the block	10
6.1	Specification of the block	55
6.2	Specification of the block	58

Introduction

OVER the past decade, the [Internet of Things \(IoT\)](#) has grown from an emerging technology to an absolute technological phenomenon that is transforming businesses across continents and industries. As more enterprises integrate IoT devices into their infrastructure, concerns about IoT network security are becoming more prevalent. From January to June 2021, there have been more than 1.5 trillion IoT security breaches, and this number will only increase as it is expected to reach more than 40 billion connected devices. For this reason, it is important to provide these devices a high degree of security. An important part of this process is to provide certificates to the IoT devices. The certificates provide to them identification, confidentiality and integrity of the data.

The key part of identification in certificates is linking the certificate to the real identity that is representing. These capabilities have been provided in part by certificates managed by [Certification Authorities \(CA\)](#). However, these are not free from failures, they represent a single point of failure as they are centralized entities, all the trust placed in them. In addition, problems arise when these root certificates expire, leaving devices unsecured since their certificates are linked to a no longer valid root certificate. Besides, most of the certificates have not free and having many of them to secure each individual device can be a considerable overprice.

This work proposes a new [Root of Trust \(RoT\)](#) scheme that does not rely of the traditional Certification Authorities. This entities are shifted and the trust is attach to the device during the manufacturing process. The way it is linked is by using the hardware particularities of each device. Besides, this link is validated and traced on the blockchain.

Blockchain technologies are becoming increasingly popular for use with [Internet of Things \(IoT\)](#) , as they provide a secure and tamper-proof data trail that can guarantee ownership of data and user privacy[1]. However, this solutions imply that our device is constantly active in the blockchain which may be not desirable or possible for some devices. Every block in the global immutable data structure that is the blockchain is connected to their siblings and

parents using a cryptography hash function. Miners, the key actors in the protocol, add new blocks to the chain by a process called mining. The lucky miner that finds a candidate block, broadcast it through the network and if most of the miners agree on the veracity of it, a new block is created. [Proof of Work \(PoW\)](#) consensus algorithm is [2] currently used in [Bitcoin \(BTC\)](#) [3] and [Ethereum \(ETH\)](#) [4] though [Proof of Stake \(PoS\)](#) is going to be the standard for the latter [5].

This technologies are tamper-proof so that a malicious user wanted to change the block, the [hash](#) representing it will change and therefore the block will not be accepted. In other to change an already mined block, the attacker would have to change the current and all the previous blocks until reaching to the targeted one, a quite computationally expensive task [6]

This work uses this property to store the data used to establish a secure channel and the certificate's digest. You may be wondering why we are storing the hash and not some other data like an HMAC or a signature that is more secure. Well, the truth is that the data is added to the contract through a transaction that is signed and later an access control mechanisms verifies that the sender of the data is the owner of the device, so that authentication is provided and there is no need for a signature or similar. Moreover, the hash perfectly fits of a uint256 variable, thus reducing the gas cost of the operations.

Afterwards, there the data is interpreted by the [Ethereum Virtual Machine \(EVM\)](#) [7] and formalized and manipulated using the scripts in the network. This scripts are called smart contracts and once they are validated on-chain, they cannot be changed, It is as if they were set into a [One-Time-Programmable Memory \(OTP\)](#). The same thing happens to the data registered in the blockchain, it cannot be changed as well, whenever something is deployed in the network is there for the lifetime of the blockchain.

This property is really desirable for our infrastructure so that all the data that is shared between the participants cannot be tampered by an attacker. Besides, the network provides transparency because anyone in the blockchain can consult the data of a transaction, including the smart contract code. Therefore, encryption must be used to protect sensitive data pushed to the network. The properties that the blockchain offers guarantee that both the data and the originator of it can be trusted without the necessity of any third party.

The Ethereum network along with [Ethereum Improvement Proposals \(EIPs\)](#), EIP-721[8], introduces the concept of ownership of non fungible tokens, that is non interchangeable tokens. This ownership can represent the provenance of a physical or digital asset. This can allow us to represent the ownership of our devices in the blockchain making it impossible to tamper and be traced during the lifetime of the device. Operations like changing of owner or assigning it to a new user are registered in the ledger.

The EIP-721 is really useful for representing the ownership of physical devices like IoT although it does not come with no drawbacks. Problems arise when trying to associate a

token to physical device, like is the person in charge of assigning that ownership reliable? Can we trust that the device behind that **Blockchain Address (BCA)** is the device that we want to communicate with? That will get us on relying on **Certification Authorities (CA)**, back to a central authority scenario. However, the use of **PUF** [9] [10] can physically identify the device and NFT tokens can be used to link the device to an owner. The use of this **Physical Unclonable Functions (PUF)** to physically identify the devices and to link it to an NFT as already being proposed in [11]. However, we want to go further in this work.

Physical Unclonable Functions (PUF) are like the genome sequence of the silicon chip of a device. This kind of functions use the variations on the integrated circuit during the manufacturing process of the chip to uniquely identify it. Even if, the whole process was consistently motorized, random differences are introduced somehow so that each chip is unique. This way the device is identified by its physical characteristics and not by a third entity. It is like identifying a person by its face.

Once we have identified the device and associated to a blockchain address. An NFT is used to represent the ownership of the device that it is bound to a token. The device is also linked to a x509 certificate, that is currently used in most of the secure communication protocols and in access control mechanisms in layer two like IEEE 802.1X. [12]

The novelty introduced in this work, is the design of web interfaces to control the workflow of operations introduced in [11], integrate it with the existing secure communication protocols as **Transport Layer Security (TLS)** and **Datagram Transport Layer Security (DTLS)** to provide support to legacy devices. Besides, a infrastructure of managing the certificates proposed, some of the roles defined are also removed and the necessity of the device to participate actively in the blockchain is also suppressed.

In an effort to increase the decentralized properties of the infrastructure, decentralized based storage system is use to store the certificates of the devices of the infrastructure. Networks like IPFS [13] and FileCoin [14] [15] are used in this work.

This document starts making an overview of the concepts of the resources used, the first chapters of this document, introduces the basis concepts for the proposal introduced. The first chapter will touch the Bitcoin blockchain, its components and how this components contribute to the functioning of the network. The second chapter introduces the Ethereum network which can be considered an expansion of the Bitcoin network in the sense that the functioning of the network is similar and the possibility of writing code in the protocol is added through smart contracts. Concepts of ownership and tokens are also part of the network and they are used intensively in this work.

Chapter number four, introduces the concept of decentralized storage and explains the mechanisms used in the proposal to store the certificates. The fifth chapter, talk about the **PUF**, the cryptography anchors that tie the physical world and the blockchain addresses of the

devices. Last chapter will be devoted to describe in detail the infrastructure and mechanism used to certify our IoT devices.

The breakout of the Blockchain

THE first implementation of a public blockchain was released in January of 2009 by an anonymous author called Satoshi Nakamoto. The identity of this person or group of persons remains secret although it is thought that the real identity of this nickname could be around four guys: Dorian Nakamoto, Craig Wright, Nick Szabo and Hal Finney. Basic knowledge about hashing algorithms, public cryptography and encryption algorithms is assumed.

2.1 Bitcoin Wallets

The Bitcoin wallet is secured through digital keys, bitcoin addresses, and digital signatures. Bitcoin uses public key infrastructure to authorize the transactions and each wallet has a pair composed by a private and a public key. The private key's used to sign the transactions and the public key is like the unique identifier of the bank account, it is used to receive BTC. The private key is generated by a user of the chain and the public key is derived from it. The private can also be referred to as the *signing key*, it creates message signatures that can be verified using the public the *verifying key*, thus linking both keys. The great thing about this is that we do not need to disclose the private key to prove the validity of the signature. So to spend bitcoins, the current bitcoin owner must include both their public key and a unique signature—produced using their private key each time—in the transaction. Everyone on the bitcoin network may validate the transaction's authenticity and that the individual sending the bitcoins actually possessed them at the moment of the transfer by displaying the public key and signature. Therefore, if a user exposes his key pair, he will lose control of his key pair and he will probably end up losing all the funds of the wallet.

Bitcoin uses the [Elliptic Curve Digital Signature Algorithm \(ECDSA\)](#) algorithm [16] to generate the private key. Then, the bitcoin address is made through double hashing. A hashing algorithm may be a non-invertible function that creates a hash or summary of data of

any size. This kind of algorithms are deterministic, meaning that they always create the same output given the same input. In bitcoin, cryptography hash functions are widely utilized in bitcoin addresses, script addresses, and is it applied multiple times in the "Proof-of-Work" method for mining.

SHA256 and RIPEMD160, are the algorithms wont to create a bitcoin address from a public key. Within the process of generating the wallet address, we compute the RIPEMD160 hash of the output, which yields a 160 bit (20 byte) number, starting with the general public key K, then compute the SHA-256 hash of the result.

We take the public key P and compute the SHA-256 hash and the RIPEMD160 hash of the 256-hash, obtaining a 160 bit number:

$$W = RIPEMD160(SHA256(P))^1.$$

After this process, most all the wallets are codified using *BASE58Check* to provide a more human readable way.

2.1.1 Types of wallets

We can make two clusters of types of wallets depending on the generation of the key and the storage of the key. Depending on the *generation of the key* we have two types of wallets:

- **Non-Deterministic Wallets:** This wallet is made up of randomly generated numbers that compose one private key. This type of wallet is called a Type-0 Non-Deterministic Wallets. Because they are difficult to manage, backup, and import, this sort of wallet is also known as "Just a Bunch of Keys," or JBOK. Deterministic wallets are replacing such wallets.

Random keys have the drawback that if you produce a lot of them, you have to retain copies of every single one, necessitating regular wallet backups. If the wallet cannot be accessed, it happens that the irrevocable loss of the cash it controls. However, by utilizing each bitcoin address for a single transaction, we are avoiding address reuse. Because several transactions and addresses are linked together, address reuse lowers privacy. If you wish to minimize address re-use as much as possible, the use of Type-0 non-deterministic wallet can help.

- **Deterministic Wallets:** They are wallets that employ a one-way hash algorithm to generate private keys for each wallet that are all obtained from a single seed. The private keys are derived from the seed, which is a randomly generated integer. The seed can

¹ where P is the public key and W the wallet address

be composed with other information like the code of the chain ² (number one is for the main network of ETH) and other like index number. This wallets only require a single backup at creation time since the seed is enough to retrieve all of the generated keys reducing all the overhead of type-0 wallets. The user's keys may be easily transferred across other wallet implementations thanks to the seed's sufficiency for exporting or importing wallets.

The other type of wallets that are classified by the storage of the private key, are the following:

- Cold Wallets: The private keys are generated and kept offline. They are usually composed of a hardware security module that stores and constructs the transactions offline so that the private secrets are never exposed to the Internet.
- Cloud Wallets: This types of wallets are the ones used by the exchanges such of Binance, KuCoin, etc, who are the true owner of the private keys. This kind of wallets are not recommend, you are not the true owner of the funds, "not your keys not your money", it basically breaks the decentralized principle of the blockchain technology.
- Desktop/Smartphone wallets: The private keys are kept in a device that is connected to the Internet such as a laptop or a mobile phone or any other device. The compromise of the device can led to lose of ownership of the private key.

2.1.2 Transactions

A transaction starts with the creation also known as origination, where the parameters of the transactions are filled with the correct data. It is followed by a process of signature by the owner or owners (multi signature wallet) that originated the transaction. The signed transaction is then broadcast across the bitcoin network, where each node (participant) verifies and spreads the transaction until it reaches (nearly) every node in the network. The transaction is then added to a block of transactions that are stored on the blockchain after being validated by a mining node. The data contained in a transaction is the one specified in the Figure 2.1, the block head, the transaction count and the transaction pool.

2.2 Creation of Bitcoin

We have had a brief introduction about the basis on the protocol, so let us make a rest from the theoretical concepts and introduce a little bit of history about this technology.

² The chain ID is a property of the blockchain that is managed by the nodes, it identifies the different networks. This mechanism is used to prevent double-spending of digital assets by ensuring that each transaction can only be spent once.

On August 18, 2008, The domain name bitcoin.org was registered on the 18th of August of 2008. Two months and ten days later the October 28, the anonymous author released the official white-paper and by the beginning of the year Nakamoto mined the genesis block—the first block in the chain—on January 3, 2009, the bitcoin network was born. The sentence "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks" was inscribed on the coinbase of this block. This has been seen as both a timestamp and a critique of the instability brought on by fractional-reserve banking.

Hal Finney, the person who in 2004 developed the first [reusable proof-of-work system \(RPOW\)](#) and one of the guys that is thought to be behind [BTC](#), was the recipient of the first bitcoin transaction. The day it was released, Finney downloaded it, and on January 12, 2009, Nakamoto sent him 10 bitcoins. Wei Dai, the originator of b-money, and Nick Szabo, the creator of bit gold, were both early proponents of cypherpunk. The first commercial buy was made in 2010 when programmer Laszlo Hanyecz paid Jeremy Sturdivant 10k [BTC](#) for two Papa John's pizzas.

Before vanishing in 2010 and giving Gavin Andresen control of the code repository and the network alert key, according to blockchain researchers, Nakamoto had mined around one million bitcoins. Andresen eventually rose to the position of lead developer at the Bitcoin

2.3 How it works?

The Bitcoin protocol is peer-to-peer fully-distributed system in which there is no central entity or server that owns or controls the protocol. By an economic perspective, it is decentralized digital currency that you can trade directly, without an third party like a bank. The words "decentralized" and "fully-distributed" here mean that no intermediary is needed to make the transactions so there is no single point of failure on the network. New Bitcoin is constantly added to the network each time new transactions are confirmed through a process called *mining*. Thus making the Bitocoin an inflationary currency, the more the more inflation you have.

An ordered collection of blocks of transactions with back-links makes up the blockchain data structure. This data structure can be stored in a straightforward database or as a flat file. Each block in this distributed database is tied back to its parent by a cryptographic link.

The header of each block in the blockchain contains a hash that was created using the SHA-256 cryptographic hash technique. The "previous block hash" element in the block header of each block also serves as a reference to a preceding block, sometimes referred to as the parent block. In other words, each block has a header that includes the hash of its parent. A chain leading back to the genesis block, the first block in the blockchain is formed by the series of hashes connecting each block to its parent. All of this can seen a bit confusing at a

first glance and sometimes, it is more illustrative to have an image to see the structure so let us take a look at the following image:

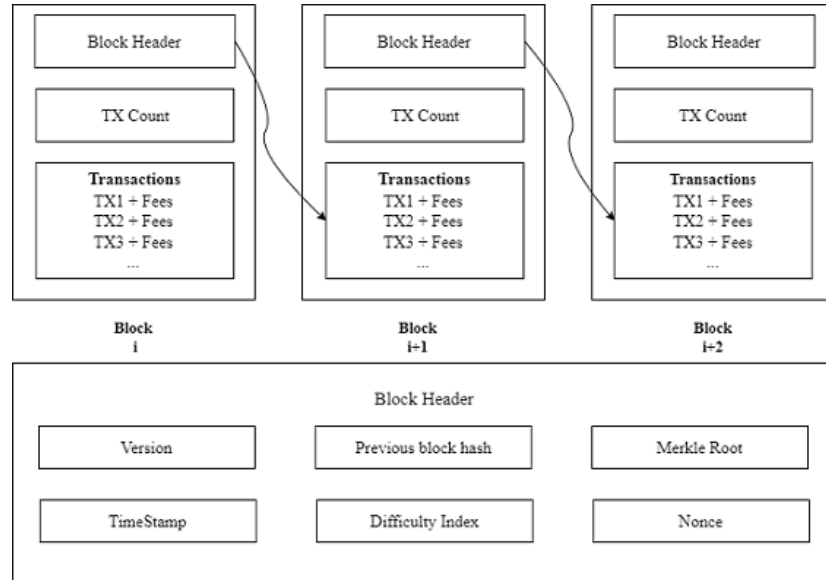


Figure 2.1: Structure of the blockchain

In figure 2.1, we can observe that the block header of the previous block link to the next one and each of it has the previous block hash field that we talked about, keeping the 256-bit hash of the previous block header. Although, each block can only have one parent, a brand new fresh block can have multiple siblings, but how is that possible? That can happens because of the way the consensus algorithm works. When distinct blocks are found, almost simultaneously by various miners or if a malicious miner tries to include a faulty block, a transitory condition known as a "fork" in the blockchain occurs, resulting in several offspring, this known as a fork. The "fork" is eventually resolved when only one kid block remains and joins the blockchain. Despite the fact that a block may have several children, each block can only have one parent. This is due to a block only having one "previous block hash" field that points to its only parent.

2.3.1 Structure of the block

The block header is composed of four sets of data. The first set contains the information related to the mining process, namely the difficulty, timestamp and *nonce*, this operation is further explained in the *mining section*. The second set, reference to the previous block hash, the "previous block hash" field, which connects the current block to the previous in the chain of block. The third group of information contains the Merkle Root field which holds the summary of all of the transactions of the current block. The last set is used by miners

for readiness signaling and protocol upgrade support. The following *Table 2.1* exemplifies the structure of the block.

Field name	Size	Description
<i>Version</i>	4 bytes	Version of the protocol to track updates
<i>Previous Hash Block</i>	32 bytes	Hash of the previous block in blockchain
<i>Merkle Tree</i>	32 bytes	Hash of the Merkle tree of the current block
<i>Timestamp</i>	4 bytes	The approximate time of creation of the block
<i>Difficulty Index</i>	4 bytes	The difficulty of current block
<i>Nonce</i>	4 bytes	The number used to solve the hashing challenge

Table 2.1: Specification of the block

2.3.2 The Merkle tree

The merkle tree used in the bitcoin blockchain is used to summarize each block’s transactions. A merkle tree is a data structure that helps keep track of the location of data items in a database. Its structure is used for quickly condensing and confirming the accuracy of enormous data sets. Binary trees called Merkle trees are used to store cryptographic hashes. In computer science, branching data structures are called ”trees”, but these graphs are usually upside-down, with the ”root” at the top and the ”leaf” at the bottom, as you’ll see in the examples next.

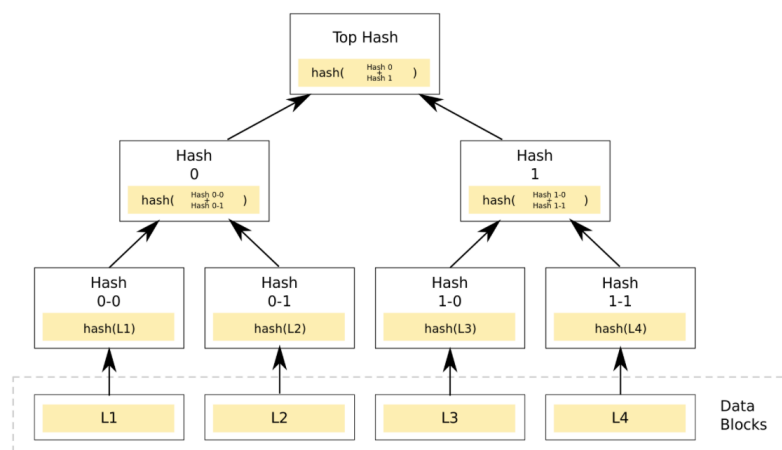


Figure 2.2: Merkle Tree

The summarization of all the transactions in a block using merge trees in bitcoin results in a comprehensive digital fingerprint of the entire group of transactions, making it very quick and easy to check whether a transaction is part of a block. Recursively hashing node pairs creates a Merkle tree, which is then built from the root (also known as the merkle root) of the tree. Double-SHA256, often known as double-SHA256, is the cryptographic hashing method used in the merkle trees of bitcoin.

This is an extremely effective data structure since it only requires at most $2 \cdot \log_2(N)$ computations to determine if any one data element is present in the tree once N data items have been hashed and compiled into it.

Besides another type of tree, Fast Merkle Trees, is devoted to reduce the overuse of the compression step in Satoshi Nakamoto's original Merkle tree implementation for the Bitcoin network.

2.4 Mining

Approximately, every 10 minutes a new block is mined and a new bitcoin reward is issued to a lucky miner that found the answer to the cryptographic puzzle. The transactions that are contained in that mined block are considered have one confirmation and the more mined blocks on top of it the more confirmations the transactions have. The protocol works more or less as a Sudoku contest where you have to pay a entry fee to solve a Sudoku and you get rewarded whenever you are the first participant to solve it, afterwards the winner broadcast the result to all the participants who check the result. In the Bitcoin protocol the entry fee is the electricity cost spent trying to solve the cryptography puzzle, finding a hash that is less than the target. This puzzle is very difficult to solve, but once found, it is easy to verify by others. This property is key so that participants can verify that the output hash comply with the established conditions in the chain. PoW [5] is name of the consensus algorithm of the network, that makes sure that blocks are only considered valid if they require a certain amount of processing power to be created. [2]

The miners who are in charge of creating new blocks, use a nuncio to repeatedly conduct the mathematical operations until they discover a valid output hash. This, depending on the circumstance, is a random integer, *nonce field*, that is repeatedly utilized and altered until a suitable exit signature or hash is discovered. Therefore, the mining operation can be reduced to repeatedly changing the nonce field and computing the hash of the block header until a hash less than the target is found.

This operation is computationally expensive so that the more blocks mined on top of a block the more expensive becomes to revert or modify the previous transactions. If a malicious miner wanted to corrupt a newly mined block, he/she would be required to have at least 51%

of all the computational power of the network in order to set its "own consensus", also known as the *51% Attack on the Blockchain*. The hash rate needed to perform the attack at the time of writing, 5th of august, is around:

$$210M * 0.51 = 107,1Terahashes/second,$$

which is an huge computational power. However, let us imagine that we are able to get 51% of the total amount of hashing rate, even if we successfully tamper the first new mined block, the propability for us mining the next and the following ones drops exponentially by time as stated in the calculations section of [3]

2.4.1 How a candidate block is found

The target is derived from the difficulty field found in every block header of Bitcoin. This value is calculated using the following formula:

$$target = targetmax/difficulty$$

where $targetmax = 00000000ffff000$ in hexadecimal.

Let us say that our current difficulty is:

$$\begin{aligned} TMax_{10} &= 26959535291011309493156476344723991336010898738574164086137773096960; \\ Target_{10} &= TMax_{10}/Difficulty_{10}^{750,605} = \\ &= 950899733299872436965338707642045189129538301706690407.48; \\ Target_{16} &= 00000000000000000009ed87fffffe9b188b008b03acea781a2fc2ddafcdf67; \end{aligned} \tag{2.1}$$

Being, the superscript of the difficulty variable, the block number and the subscript, the base of the number; we have that the target of the block 750,605 would be the variable Target16. Therefore, we have to find an output hash less than $target16$ by computing over and over the different hashes that result from changing the nonce each iteration. All the information about the process of calculating the target is further explained in [17] & [18]

This chapter was a little scoop of the Bitcoin blockchain, how it works and its properties that arise from the blockchain. The next chapter will be devoted to an special block which is, in my opinion, the most important blockchain at the time of writing.

2.5 Limitations of Bitcoin

The Bitcoin blockchain supposed a revolution in the IT and banking industry introducing a new type of decentralized database that store transactions that hold money and that are sent through a network of unknown actors. That isn't crazy? However, this network comes with limitations as the Bitcoin Scripting language which reduces the options and increases the complexity for developers that want to deploy scripts or programs that run in this network. From this idea, a new blockchain network was born by a young programmer in 2015 which we introduce in the next section.

Ethereum

ETHEREUM shares numerous common factors with other open blockchains like Bitcoin a peer- to- peer network connecting actors, a intricate fault-tolerant agreement algorithm for coinciding state updates(a evidence- of- work blockchain), though PoS [5] is currently released, the use of cryptographic primitives like digital signatures and hashes, and a digital currency.

Although, Ethereum is different than other open blockchains, in that it has a specific purpose and construction different from blockchain like Bitcoin. Ethereum’s primary purpose is not to be a payment network, but to serve as a foundation for decentralized applications. It is a more ambitious project that has the potential to revolutionize the way we interact with the internet. Ether is both essential to the operation of Ethereum and a means of payment for using the Ethereum platform as a world computer, the latter is the most important role of this currency.

Ethereum was designed to be a decentralized machine that runs a virtual machine with the ability to run programs of any complexity and to be programmed in a general-purpose programming language. Bitcoin’s simple scripting language is perfect for accurately monitoring spending conditions, while Ethereum’s more versatile Turing-complete language is perfect for performing general tasks. Due to its, simplicity in outputs when it comes to validating a transaction, the Bitcoin programming language is restricted to true/false evaluations.

Unlike Bitcoin, the Ethereum Client network does not have a reference implementation, it has a reference specification where a variety of clients are built from. This reference is composed by a Yellow Paper [4]

In this chapter, we would not get into the details of the blockchain, we will just focus on the high level functionalities that Ethereum offer to us. This work uses one of the test networks called Goerli as part of its infrastructure, so a small section will be dedicated to this topic.

3.1 Genesis of Ethereum

The motivation for this new blockchain came from the developers' dilemma of having to write code in existing blockchains. This would have forced them to work on an environment with various limitations like scarce storage sizes and data types with limit the types of application that could run on Bitcoin. This meant that either they write their code in a restricted, non-Turing complete Bitcoin Script language, or they create a new blockchain, the latter of which was chosen and Ethereum was created.

The Ethereum network began with the release of its whitepaper [19] in 2013 by a programmer Vitalik Buterin where he proposed a Turing-complete, general-purpose blockchain. The authors of one of the reference [20], were most of the concepts were taken from, were curious about the propose of Vitalik and intrigued by the idea and asked Vitalik about design a completely independent blockchain network to create their own consensus rules that will be used by smart contracts upon execution. Dr.Gavin Wood was one of the first people to offer Vitalik an opportunity as an C++ programmer. Gavin is now one of the founders of Ethereum, Code Signer and CTO of Ethereum.

The day the first Ethereum block was mined, 30 of July of 2015, a new world level computer started functioning non-stop and on a global scale. This set the beginning of the Ethereum network.

3.1.1 The four stages of Ethereum

Ethereum had four main stages during its development and by the time of release they are ordered in the following way: Frontier, Homestead, Metropolis, and Serenity. The Frontier was the *initial state* of Ethereum, the state from block 0 till block 200,000 in March 2016. On that block, the difficulty of mining increased exponentially which the Ice Age hard-fork to promote the change to a PoS network. At block 1,150,000, the Homestead stage, *the second stage* that introduced changes for the protocol and network detailed in three different EIP's (Ethereum Improvement Proposal), the EIP-2, EIP-7 and EIP-8. It was launched in March 2016.

The DAO was one of first decentralized autonomous organization and it was deployed on the Ethereum main network. This makes the code of the contract public and verifiable to everyone that can understand Solidity programs and this making it more trustfully. However, it comes with a drawback, anyone could find a bug an exploit a vulnerability on your code and that is what happen to DAO. The DAO held \$150 million on its hands worth of ether after a crowdsale in April 2016. This was one the leading projects at that time and it was a milestone as one of the most financed at the time. However, not even three months went by and the project was hacked. A bug that was found in the project's code caused \$60 million to be stolen, an exploit that costed \$60M. Although, the funds were taken back to investors

after fork that was proposed in the main network .After this event, the Ethereum network was forked in two different blockchains, Ethereum Classic and Ethereum, the latter was the one tat restore the funds.

A hard fork was introduced in block 2,463,000, the Tangerine Whistle its purpose was to modify the calculation of the gas price for certain I/OHheavy operations and clean up accumulated wealth from a Denial of Service (DoS) attack which used the low cost of the operations to perform the attacks.

Although, the Tangerine Whistle tried to solve DoS attacks, another hard-fork was needed to be introduced. The Spurious Dragon hard fork (EIP-607)[21], came up in block 2,675,000 in response to a second round of attacks in September and October of the previous year of publication of the proposal. The hard fork proposed, addresses important but less important problems that will help improve the security of the network.

The third stage implements a number of updates within the Ethereum blockchain, including the inclusion of the "REVERT" opcode, which enables error checking without consuming all gas, added support for modular exponentiation of large integers, support for variable-length return values, and changes to Difficulty Adjustment Formula. Some of the most controversial inclusions include the addition of completely anonymous zero-zero knowledge tests (ZK-Snarks) and the delay of the difficulty bomb (nicknamed the Ice Age) by one year, as well as the reduction of the *block creation reward from five to three ethers*. This changes were applied in block 4,370,000.

The fork that was considered in [20] as the last stage of Ethereum in this book written by creators but the truth is that is one many improvements introduced in the chain. This fork paved the way to the future consensus algorithm PoS so that the blockchain do not freeze before implementing it, some certain opcodes suffer a price adjustment and the ability to interact with addresses that don't exist yet was introduced.

The rest of the forks are included in the appendix.

3.2 Test Networks

Besides the core network, the main network, there are public test networks. These networks are used by developers to test protocol updates and potential smart contracts in a simulated environment before deploying them on the core network. Production servers are typically used to generate and process data, whereas storage servers are used to store data. In most cases, it is important to deploy and try your code and search for different errors on a test network before allowing other people allow people other than you to discover the bugs on the main network. If you are building a decentralized app that talk with already deployed smart contracts, you may be able to access copies of the project on test networks.

Most test networks use a proof-of-stake consensus mechanism. That is, a small number of nodes are chosen to validate transactions and create new blocks by staking their identities in the process.

Incentivizing mining in a proof-of-stake network with proof-of-work is difficult because it can lead to vulnerabilities. ETH has no real value in a PoS network. Therefore, there is no market for ETH in proof-of-stake networks. Most web applications that allow users to send Ethereum to other users use addresses as input.

- Main Ethereum Network (Chain ID = 1): This is the real network, where the real consensus is achieved and the real decentralized applications are built.
- Ropsten Test Network (Chain ID = 3): It was the main test network by default but now is deprecated and it is increasingly out of use. It was launched in November 2016, with the same consensus algorithm as the main network *proof of work* consensus algorithm. The one used in the most used blockchain Bitcoin.
- Rinkeby Test Network (Chain ID = 4): This is one of test networks available on Ethereum. It uses a modified version of PoS algorithm, the Proof of Authority consensus algorithm is used. This network was created in 2017 but is deprecated and it will soon be replaced by the Goerli network.
- Goerli Test Network (Chain ID = 5): It is the first cross-client Proof of Authority (PoA) testnet is syncing Parity Ethereum, Geth, Nethermind, Hyperledger Besu (formerly Pantheon), and EthereumJS. This testnet is a community-based project that is open-source and does not require a natural environment. The Ethereum community launched an open-source project in September 2018 called ETHBerlin that has been gaining popularity and attracting more contributors ever since.
- Sepolia Test Network (Chain ID = 11155111) is *proof-of-work* a test network created by the Ethereum Core developers and it was released in October 2021 that will be maintained until it transitions to proof-of-stake in June 2022. After that, Sepolia and the Goerli testnet will move from proof-of-work to proof-of-stake to mimic the Ethereum mainnet. According to Ethereum core developer Tim Beiko, the development team will be able to finalize the date to complete the Ethereum Merge once Ropsten, Goerli and Sepolia have forked and stabilized.
- LocalHost 8545: This is reserved for Ethereum blockchains implemented on a local network like Ganache.
- Custom RPC: The providers like Metamask get in this category.

Note that the most important property of the block chain networks, the decentralized consensus comes with the scale of the network, the more people they use it the more reliable it is. That is why the more miners join the more difficult it is to mine a block and the more robust the network become. The 51% attack is less probable with more users on the network.

This work will use the Goerli test network which one of the easiest test network from which a decent amount of Ether can be obtained through faucets. This networks do not have the same computational power as the main network so it is not difficult to get some ether by just mining. For this work, this current Goerli faucet was used to obtain GoerliEther.[22]

3.3 Account

As we saw in last chapter, the wallet address is generally composed by two variables, a public key which derives the public blockchain address and a private key, used to prove ownership of a wallet address without disclosing it.

The public key used in Ethereum is composed by two points x and y in the curve that satisfy its equation. The curve used in Ethereum is the named curve *secp256k1*, the same one as Bitcoin. [23]

In simpler terms, the public key is the combination of the two coordinates. These points are produced from the private key using a one way function which makes easy to verify the public key but almost impossible to revert it. This is due to the discrete logarithm problem (DLP) whose solution is to try all the possible combinations until the correct one is found.

Using the elliptic curve multiplication, the public key is calculated : $K = k * G$

The private key k is used in conjunction with the generator point G to calculate the public key K . The public key K can then be used with the special elliptic curve multiplication operator $*$ to perform various operations. The inverse of this formula is what is know the discrete logarithm problem and its difficulty to be inverted supports the cryptography of the network.

The most common type of addresses are the [Externally Owned Account \(EOA\)](#) that are used to send and receive Ethereum, they are of the same types of the ones in Bitcoin. The smart contracts addresses are the created when the smart contract is deployed, on creation of a smart contract, a unique Ethereum address is returned. This type of wallets, as opposed to the externally owner, do contain smart contract code that the others simply cannot.

The smart contract address can also send and receive Ether, however they should be programmed to allow that kind of operations. In the case, of sending Ethereum to a contract that cannot be triggered by us or that do not have any logic or statement that allow to make relief funds from it, all the funds sent to that contract address will be lost forever. Therefore, we should be really careful when sending funds to a smart contract address.

Note that all the logic of the address is controlled by the smart contracts and that they

cannot instantiate by themselves new transactions but react to them. The transaction along with the transaction data can trigger a specific function in the smart contract to run.

The changes introduced in Ethereum not only affect the types of account in the network, also the transaction structure and data types contained in it that we will see in next section.

3.4 Transaction

In the last chapter, an overview of the structure and the operation of the transactions in the Bitcoin network. The transactions in Ethereum are quite similar to Bitcoin's so this section is not devoted to explain all the details of the structure and some of the components are referred to the previous chapter. This figure illustrates how the Ethereum network is designed:

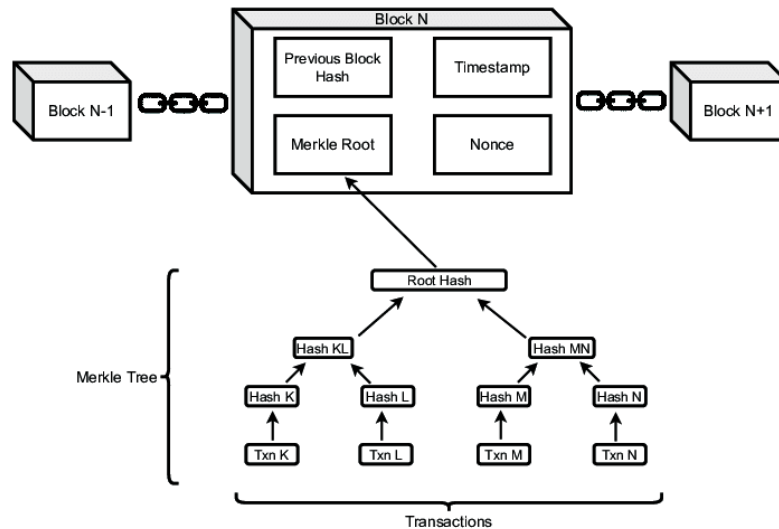


Figure 3.1: Structure of the Ethereum blockchain

Each block in this network, is composed by the following elements:

- Nonce: This field prevents message replay, it works as a sequence number.
- Gas price: The price in wei that the issuer of the transaction is willing to pay.
- Gas limit: The maximum amount of gas that the sender is willing to use for this transaction¹.

¹ Executing code requires making a transaction, therefore sender, issuers, the one in charge of deployment can be referred as similar concepts

- **Recipient:** The destination address of the transaction, they can be either a contract or another user. The representation of both it is the same, a public Ethereum wallet address.
- **Value:** The amount in ETH that is included in the transaction.
- **Data:** The data payload. This field is not fixed although is limited.
- **v,r,s:** This three variables compose the ECDSA digital signature of the sender EOA.

The Structure of the Transaction Message is Serialized using the RLP Encoding Scheme, Designed Specifically for Byte-Perfect Data Serialization in Ethereum. Ethereum numbers always occupy multiples of 8 bits and use the big-endian order of bytes to store the integers.

There is an important change in the Ethereum transaction scheme and it is that the field "from" that identifies the originator EOA, is present in BTC but not here. That is because the public key associated to the account can be built from the v,r,s field of the transaction thus recovering the ECDSA signature. The public key can be used to derive the address. When you see a transaction displaying the From field, it is just created by the program for increasing clarity in the visualization . The transaction frequently includes other metadata included by the client software: the transaction ID (computed hash) and the block number (once it is mined and embedded in the blockchain). This data is based on the transaction itself, and is not included in the transaction message itself.

3.4.1 The Nonce

The nonce is used to avoid message replay and counts the number of confirmed transactions done by a current originating account. This integer is not stored explicitly and it is not part of the account state. Due to that, this number is calculated dynamically each time a user wants to make a transaction. This attribute belongs to the sender and only has a meaning in this context. In the definition of the yellow paper [4], we can read:

nonce: A scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.

This number is crucial and avoids the following scenarios to happen:

- Let us imagine that we are out for dinner with friends and one of our friends payed for the whole dinner in cash and the payment is split between all of the presents and reimbursed in Ethereum to his account. We make the calculation and each of us have to make a payment to our friend of 0.1 ETH. Each one take, their phone wallet and create a payment of 0.1 ETH to our friend, great. However, we forgot to add drinks that we took before the dinner and that were in another bill so no we should reimburse him 0.15 ETH. So, we quickly take our account and make him another transaction while the

other one was not confirmed, believing that other one will be reverted and the later will be accepted. However, that is not the case, since the nonce is present, the first transaction of 0.1 ETH has a nonce of n and the second of 0.15 ETH has a nonce of $n+1$. Since the transaction with nonce n is not processed and confirmed yet, the next transactions including the one with nonce $n+1$ will be discarded.

- Let us suppose that we have 50 ETH in our account. There is some crypto art that we want to buy that cost 5 ETH. So, we make a transaction that is worth 5 ETH, get our current nonce for our account, sign the transaction and broadcast through the network. One random node, receives the transaction and it just so happens that this one is the seller so it tries to trick the blockchain. The seller, clones the transaction and sets the nonce to null and broadcast the transaction. However, every transaction is unique, ensuring that each one is secure and reliable. With the increasing nonce, the duplicate payment problem, also known as double spending has gone.

3.5 The Ethereum Virtual Machine

The Ethereum Virtual Machine is part of Ethereum and it is in charge of execution and deployment of smart contracts in the network. It executes as a stack based machine with 1024 elements in depth and a word size of this machine is 256-bit which the same size as the output of the Keccak-256 hashes or secp256k1 signatures so that it is easier to use [7]. It has the following data components:

- An immutable program code **ROM**, where the **bytecode** to be executed is loaded
- A volatile memory, initialized to zero at the beginning of execution
- The Ethereum state includes a permanent storage area that is zero-initialized.

A transient memory is kept during execution, when it is finished the memory is flushed (memory used on execution does not persist between transactions) and the output is kept in the global state of the **Ethereum Virtual Machine (EVM)**. (Figure 3.2)

This global state is a huge data structure called a modified *Merkle Patricia Trie*. It keeps all accounts linked by hashes and can be reduced to a single root hash stored on the blockchain.

The output of the compilation of smart contracts result in bytecode, more or less like Java. This bytecode executes as a number of **EVM** opcodes which perform standard stack operations and also serve as a measure for the gas. Each opcode has a certain gas cost and it allow us to calculate how much gas we need to execute certain stream of code prior execution. The gas is a new concept introduced in Ethereum so that the **EVM** is a quasi-Turing complete state machine and determines the halting by the gas that the user has provided prior execution.

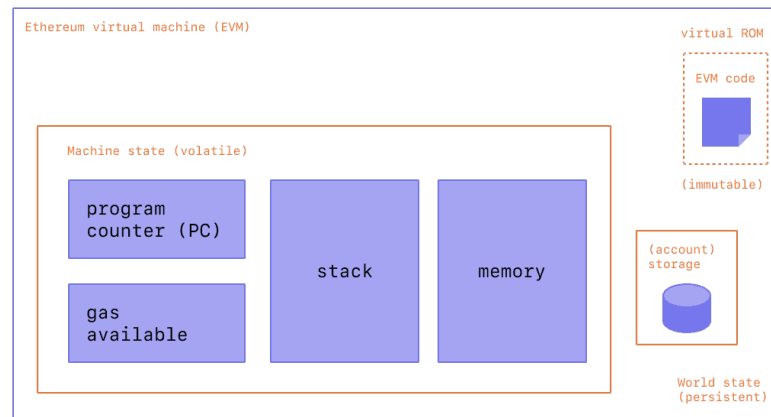


Figure 3.2: Ethereum storage scheme

3.6 Gas

Ethereum uses gas to measure how much computational and storage resources are needed to carry out actions on the Ethereum blockchain. In contrast to Bitcoin, where the transaction fee only takes into account the size of the transaction in kilobytes, Ethereum has to take into account every computational step performed by the transaction and execution of the smart contract code. Each operation performed by the tract requires a certain amount of gas. Some of the examples of operations that require gas are multiplication (MUL) consumes 5 gas, a division (DIV) consumes 5, (ADD) consumes 3 and the SHA3BASE operation which is the calculation of the Keccak-256 hash is 30 of gas and 6 of gas for any additional 256 bits being hashed.

Gas is important for both ensuring that Ethereum's price remains stable and protected from attackers. It costs a certain amount of gas to do various things, and it's important to use enough gas to do the task at hand without running out. The gas is also used to prevent situations where the execution of the program gets stuck and so the Ethereum virtual machine, measures have been taken in the design to prevent infinite loops. Each transaction requires a set limit on how much computation the initiator is willing to pay for. By using a gas system, attacks like malicious or spam transactions are not even worth a try to attackers, as they must pay for the resources they consume in a proportional manner.

3.6.1 How the gas is consumed during execution

For each opcode present in the [EVM](#), there is a gas cost associated to it and during the execution of the program, the gas is decreasing depending of its cost. Before each operation, the [EVM](#) checks to see if there is enough gas to pay for the operation. If there is not enough gas,

execution is suspended and the transaction is aborted. If the [EVM](#) finishes executing successfully, without running out of gas, the gas cost used is paid to the miner as a transaction fee, the gas price specified in the transaction determines the amount of ether that is converted:

$$fee = gascost * gasprice$$

The gas that is not used in the execution is refunded to the originating address in the form of ether, which is calculated based on the gas price specified in the transaction:

$$unspentgas = gaslimit - gascost \text{ refund}(ETH) = remaininggas * gasprice$$

It can happen that the transaction originating an execution did not have enough gas to run through the whole operations and so it happens that an exception "out of gas" is raised, we have had "run out of gas". In this case, the transaction is reverted, no change is made to chain and no refund is made to the sender. That is why we should carefully estimate the gas before execution and include an extra Ether that will be reimbursed at the end of the execution.

3.7 Smart Contracts

The first time the concept of smart contract was mentioned was in 1990 by a cryptographer called Nick Szabo[24]. He defined the smart contracts as the following: *"Smart Contracts combine protocols with user interfaces to formalize and secure relationships over computer networks."* From that, the smart contracts have evolved with the introduction of decentralized consensus networks like Bitcoin.

The Ethereum network posed a revolution on this matter, it allowed to run immutable computer code that run deterministically in a decentralized computer, the [EVM](#). However, the term smart contract can be a bit misleading since the code is not smart, it should be coded as if we were using other general purpose programming language, nor a legal contract.

Most of smart contracts are written in Solidity, a general purpose programming language. After the code is written, the code is compiled to a low-level bytecode that runs in the Ethereum Virtual machine. Once we have the opcodes, we include it in a transaction, sign it with our private key and send it to a special address with is the NULL address, the 0x0 address. After deployment, a contract address is returned, which is generated from the originating account and nonce associated with the contract's creation transaction. An Ethereum contract's address can be used to send it funds or as the recipient of a function call. It is important to note that, unlike with regular accounts, no keys are associated with a smart contract address. It is worth noticing that as the content creator you are not granted with special permissions, as the creator you should specify the permissions on the smart contract

logic. There is no restriction on the permissions that can be set in the contract, they should only be coded into it. The contract account does not have an owner, as it owns itself. As opposite to externally owner accounts, there are no private keys for this accounts.

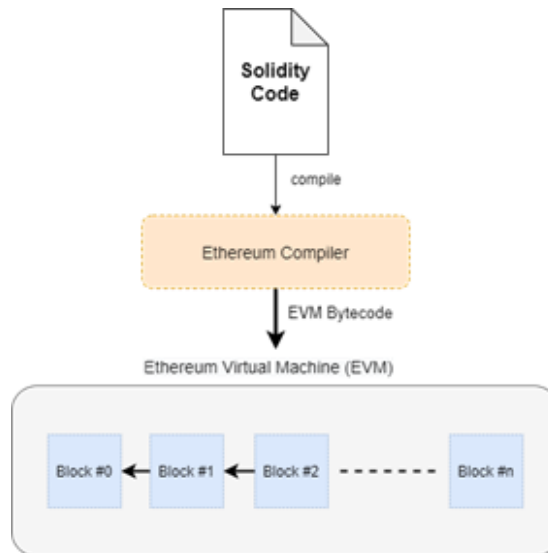


Figure 3.3: Deployment of a contract

Contracts can only be executed if they are triggered by a transaction. Since, they cannot initiate a transaction by themselves, an [EOA](#) should make the transaction that triggers the execution.

There exists the possibility of a contract triggering the execution of another contract, a contract can call another contract, and so on. Although, the root transaction the transactions path was an externally owned account. Contracts do not automatically execute; rather, they must be manually triggered by a transaction. Transactions can trigger contract execution directly, or indirectly as part of a series of contract calls. It is important to note that smart contracts are not executed in parallel, but rather one after the other in a linear fashion. Instead, they are executed sequentially on the Ethereum virtual machine, which can be considered a single-threaded machine.

Transactions are atomic, meaning they cannot be divided into smaller parts. This is true regardless of how many contracts are called during the transaction, or what those contracts do when they are called. The transaction is fully executed, any changes to the global state (contract, account, etc.) Only if all execution terminates successfully will it be recorded. Successful completion means that the program ran without errors and reached the end of execution. If the execution fails due to an error, all its effects and changes to state of the network are reverted. The failed transaction is still registered as an attempt. The gas is never refunded since the computations have been made and the cost associated them, the ether spent on this

transaction is removed from the account. However, the failed transaction does not affect the state of the contract or account in any other way. It is important to remember that the code for a contract cannot be changed once it has been created. The only changes permitted to the structure of the contract is the deletion.

3.7.1 Contract deletion

A contract can be deleted leaving the associated address as a blank account. Any funds sent to that address will be lost and no code execution will occur, it will be the same as sending funds to a randomly-chosen address. To delete the contract, the opcode SELFDESTRUCT is issued. The operation of deleting stored state results in a gas refund, negative gas cost, which incentives clients to release resources from the network. Deleting a contract will not erase its transaction history, as the blockchain is immutable. This capability of contract deletion is only available if the designer of the contract decided to implement this option. The smart contract can only be deleted if the code includes a SELFDESTRUCT opcode, and if the code is accessible.

3.7.2 EVM programming languages

The Ethereum Virtual Machine runs a code that is similar to assembly named [EVM](#) bytecode. It can be compared to the machine like operations run by a CPU. It is possible to program smart contracts in bytecode, but it is much easier to read and understand if it is written in a higher-level language. Most blockchain developers use high-level programming languages, the same as for normal programs, almost no one writes a complex application in assembly code.

It would be very difficult to adapt an existing programming language to work with the code of Ethereum's virtual machine, and this would probably cause a lot of confusion. The environment where the smart contracts execute is restricted and strained so that security and reliability are easier to preserve. It is easier to create a smart contract language specifically designed for that purpose, rather than taking another existing programming language and adapt it to the new requirements and characteristics of the new infrastructure. Ethereum has several programming languages that can be used to compile bytecode for the [EVM](#). The programming languages can be divided into two different clusters: declarative and imperative. Declarative programming focuses on what the program should do, while imperative programming focuses on how the program should do it.

In declarative programming, we write functions that express the logic of a program without dictating the program's flow. This type of programming is useful for creating programs without side effects, meaning that there are no changes to state outside of a function. Declarative programming languages are those that allow programmers to specify what they want

the program to do, without having to specify how the program should do it, specify what to do not how to do it. Examples of declarative languages include Haskell and Prolog.

In imperative programming, a programmer writes a set of procedures that combine the logic and flow of a program. Examples of these type of programming languages are Java, C and C++. There exists also a "hybrid" version which sets at a middle point between these two types, it is also referred as multi-paradigm programming languages. Some examples of hybrid programming languages are OCaml, JavaScript, and Python. It is generally possible to write code in a declarative style using any imperative language, though the resulting code is often less elegant than code written in a pure declarative language. In purely declarative languages, there are no "variables".

The most common type of programming language is the imperative programming. When different parts of a program can change the state of other parts, it becomes difficult to understand how the program will execute. This also creates more opportunities for errors. The declarative programming as opposite is a style of coding that makes programs easier to understand by removing unexpected effects. This means that each part of the code can be understood independently, without affecting other parts of the program.

The bugs are know to be a problem for most project written in imperative like languages and it aggravates in Solidity. The smart contracts handle real money so any bug as the one stated at the beginning of this section, can end up in loosing all the funds.

The most used languages order by importance right now:

- Solidity: A imperative programming language with similar syntax to C++, Java, JavaScript. Solidity's market adoption is due to its status as the first smart contract programming language, and it is being used to build many decentralized applications.
- Rust is memory efficient and fast programming language that is statically-typed. It enforces memory safety and assumes that you want to follow best design and development practices, but you can change them if you want. Rust is a language that doesn't rely on a garbage collector, meaning that there would be no unexpected incidents during runtime.
- JavaScript: It is a programming language that was designed by its creator in a week. It is a multi-paradigm and dynamically typed programming language. It has been used for web development and it has made a niche in the blockchain world since websites are evolving to Web3.0 and with the design of [DApps](#).
- Vyper: It includes features that are specific to the contract, such as event notifiers that allow listeners to be notified of events, custom global variables, and global constants. Vyper was designed to improve security in Solidity by addressing common security

issues. It was developed to supplement Solidity, not to replace it. Vyper was designed to have fewer features than Solidity in order to make contracts more secure and easier to audit. As a result, Vyper does not support modifiers, inheritance, inline assembly, function and operator overloading, recursive calling, infinite-length loops, and binary fixed points.

- Yul: Yul is an intermediate programming language designed to be compiled to bytecode for use with different backends. Yul can be used either as a stand-alone language or for inline assembly inside Solidity. Yul is designed to be a common denominator that is usable on both [EVM](#) and [ewasm](#) platforms. Yul is an ideal target for high-level optimization stages that can improve the performance of both [EVM](#) and [ewasm](#) platforms.

As we can see we have a wide variety of choices of programming languages to develop our smart contracts and DApps and to even integrate our centralized solutions with the decentralized world.

3.7.3 Security of smart contracts

Most of the problems and vulnerabilities with smart contracts[25], can be simplified to, three situations:

- Greedy contracts: Contracts that get into a state where they cannot release Ether.
- Suicidal contracts: Contracts that have the `SELDESTRUCT` option enable and have no access control to that operation. Therefore, anyone can call the deletion of the contract.
- Prodigal contracts: This kind of contracts do not have correctly implemented access control to their functionality of releasing Ether and any address can make the contract to send Ether to some arbitrary address.

As we can see most of the problems are related to a lack and/or misconstrued access control mechanism. However, this is not just the absence of certain condition that check some address, it can come from other type of errors.

3.7.4 Modifiers

Modifiers in Solidity are a key functionality. They enforce rules on a function before executing the function code. They work as a pre-check on the functions where it is included. For example, we can declare a modifier like this:

```
1 modifier requireOwner(address _blockchainAddress)
2 {
3     require(msg.sender == _account);
```

```
4     _;  
5 }
```

This modifier requires the function in which it is included to comply with the condition stated above. For example, we can include it in the following function:

```
1 function sendFunds(address _receiver)  
2     payable  
3     requireOwner(owner)  
4 {  
5     require(balances[msg.sender] >= amount, "Insufficient funds");  
6     emit Transfer(msg.sender, _receiver, amount);  
7     balances[msg.sender] -= amount;  
8     balances[_receiver] += amount;  
9 }
```

Although, the modifiers are not single use, they can be used for more functions than enforcing a condition. They can for example, add to an integer variable a certain amount each time the function is executed or whatever other code comes to our mind.

3.7.5 Decorators

Another way to control the permissions of our functions in our code is through the use of decorators. They may be used at the start of the function. Solidity provides the following:

- `@private`: If this decorator is used, the function can only be called inside of the contract.
- `@public`: If this decorator is used, the function is public and can be called from outside of the contract. This is the decorator by default when no decorator is specified. The functions that have this decorator in the contract are exposed to the outside.
- `@constant`: If this decorator is used, the function will not be able to change the state. It will fail to compile if there is some statement inside of the function tries to change the state.
- `@payable`: If this decorator is included in a function's contract, the function can transfer value.

3.7.6 Most common vulnerabilities

The focus on security sometimes have been relegated to a second perspective but in the blockchain development it must be taken into account. The smart contracts are visible to anyone and they handle real money, meaning that the consequences can cost a decent amount of money. This section will expose the most common vulnerabilities during blockchain development:

- *Integer underflow and overflow*: This vulnerability is probably the easiest to exploit and the easiest to forget.[26] Solidity has various integer sizes, uint8, uint16, uint32, etc. Let us take for example uint16, the maximum value for this variable is:

$$\text{uint16} = 16\text{bits} = 2^{16} - 1 = 65.536$$

If we reach this maximum value and add one to this variable, an overflow will be produced and the value of the variable will be zero. If we exceed the maximum value the variable will reset as if we were adding modulo *maximum value*. This happens because of the way we have to represent integers in binary.

The opposite can happen also if we subtract one unit to an unsigned integer whose value is zero, we will reach the lower bound of our variable and the value that is next is the maximum value. It is the same as we were subtracting modulo maximum value. Solidity after a couple of releases fixed this bug in version 0.8. If a lower version has to be used, it is recommended to use the SafeMath library from OpenZeppelin or make the correspondent checks by ourselves.

- *Unchecked call return variables*: Contracts can invoke other contracts, and some of these contracts can come from a third party source that we think that can be trusted. But the execution of this third party contract, can be deleted or might fail or return an unexpected result, thus making our contract logic inconsistent. Therefore, we should always deploy the contracts that our contracts call or use trusted party smart contracts. It is a good practice to check the returned values after continuing with the logic of our program.
- *Re-entrancy attacks*: An attacker contract call a victim contract function so that during execution the victim calls the contract again in a recursive way. This attack may be skipping some of the checks present in the programming logic and/or avoiding the part where the balance update is made. This can be fixed using the "Non reentrancy" pattern and applying measures like `mutex` and making the critical operations first, like updating the balance. [27] (Figure 3.4)
- *Denial of Services attacks*: Let us say that you deployed a contract that makes some logic at the beginning of the execution and calls an external contract, returns the information from it and continues the execution based on the returned value. If the external code runs with no errors, the program will always run until the end of the contract however if an attacker intentionally makes the external contract that he/she owns, to fail in a certain point, this external call acts as a breaking point in our contract. Thus causing a denial of service.

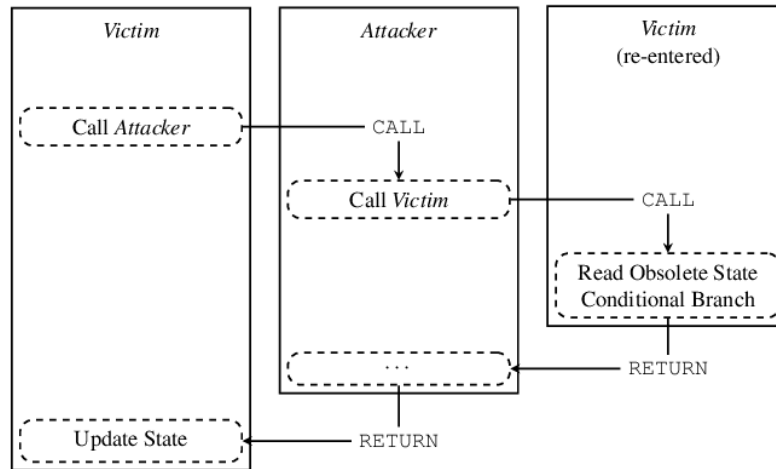


Figure 3.4: Reentrancy attack

It is always best to assume that calls made to external sources can fail, in order to avoid potential disappointment or frustration. If it is possible, please implement a "pull" pattern instead of a "push" one. When using a "push" pattern, your contract will invoke a function that will in turn invoke a third-party contract. This approach is elegant and persuasive, as it allows you to take advantage of the functionality of the third-party contract while still maintaining control over your own contract. If a malicious contract could block other users, that would be a serious problem. The "pull" pattern of contract invocation ensures that your function can only be called by another contract, and not directly by users. This protects against [Denial of Service \(DoS\)](#) attacks, as a malicious developer would only be able to block their own contract, and not yours.

- *Front Running attacks:* Transactions are not immediately added to the blockchain's distributed ledger. The ledger only contains entries that are part of a block. Blocks are collections of entries that are added to the ledger together. In order for a transaction to be added to a block, the nodes in the blockchain network need to be aware of it. Since blockchain transaction creation is decentralized, the entire blockchain network needs to be informed of these transactions. A blockchain user creates a transaction and then broadcasts it to all the other users on the network. Nodes receive copies of transactions and add them to a pool of unconfirmed transactions. When a new block is created, the block creator draws from the current pool of unused transactions. The order of the transactions being added into the blocks is usually based on the transaction fees. While some blockchains may have a minimum fee, users are generally able to set their own fees. This means that a user can increase the priority of a particular transaction by paying a higher fee. The block creator will add transactions to blocks based on fees in order

to increase their profits.

Front-running attacks exploit the process of adding transactions to blocks based on transaction fees. This results in unfairness and undermines the integrity of the system. An attacker can prioritize their transaction by including a higher transaction fee, ensuring that it is processed before any other transaction.

- *Unexpected ether balance*: A contract can have no functions with the payable decorator and still receive Ether to its account. The following situations may arise:
 - The result of "self-destruct" operation of other smart contract. When a contract is deleted, it can decide to which account reimburse the Ether from the deletion operation.
 - The contract address can receive Ether from a newly mined block. Every time a miner finds a block candidate and adds it to the blockchain, it is rewarded with Ether and it decides to whom it can send the reward.

It is a good practice that you never assume an static amount of Ether in your account.

- *Replay signatures attacks*: Signatures are used to prove ownership of the blockchain accounts. The allows to sign our transactions and make them valid to the network. The owner of the privates keys, the "original" account will sign a message. The sender account will then send the message to a smart contract. In this way, it is the delivery account that pays for the transaction fees, rather than the original account. That is why smart contracts should have the ability to to verify signatures and perform required tasks. When considering the use of signed messages in smart contracts, it is important to be aware that if the message is valid, anyone with access to it could send it multiple times. If your smart contract code is executed multiple times, it may not be what the account holder originally intended.

The solution for this attack is already solve in the Ethereum network by including a nonce on every transaction. Our smart contract, should be able to increment a variable each time an accounts signs a message. Messages with the same nonce and same originating address should then be discarded.

- *Function default visibility*: By default, if no decorator is stated in the function declaration, the public decorator is set. If you set to public a internal function that should no be, you might break the access control of your smart contract. The solution to his issue is to define the visibility for each function, never use the default.
- *Floating pragma*: Unexpected behaviours can happen if a floating pragma Solidity version is left. (*pragma solidity >= 0.6.5 < 0.8.15*). The version should be specified before

deployment.

- *Loop through long arrays*: In other programming languages we are accustomed to using without regard arrays and loop through them. However, in Ethereum, each iteration over the array costs gas so if we are iterating over a long array we may end up running out of gas. The solution is to use mappings whenever we can instead of using arrays.
- *Wrong inheritance*: Solidity supports multiple inheritances, which can introduce ambiguity (known as the "Diamond Problem"). This may be the case where multiple parent contracts implement the same function, in that case, which function will be inherited by our contract?

Solidity uses C3 Linearization to establish priorities among parent contracts. We must keep an eye on this, to avoid unexpected behaviours. As a rule of thumb, the contracts should go from the most generic to the most specific to avoid problems.

- *Saving data using weak or no encryption*: The data in the blockchain is PUBLIC meaning that it can be read by anyone and therefore all the secrets or compromised data should be cipher with a level of security comparable to AES-128 bits. Use always stream ciphers that pseudo-randomize the output, not block ciphers.
- *Access outside array limits*: Dynamic array lengths could be changed in early versions of Solidity. The problem is that anyone can edit the size of arrays and potentially change any storage space in the contract. Early versions should not be used to avoid this issue.
- *Delegate calls to untrusted sources*: The "delegate calls" should be used carefully since they use the context of your smart contract on the call of another contract. That is, the execution context is your contract's context when using delegate call. Therefore, the called contract has full access to your contract variables.
- *Calls to untrusted sources*: The external contracts are out of the scope of our contracts and therefore we should take care when using them. They can return unexpected results and block the logic of our contracts.
- *Insecure randomness*: Smart contracts and the EVM is deterministic by design. There is no true source of randomness on the network and therefore the sources of randomness cannot be used to support our operations; for example, cryptography. An oracle should be used as a source of randomness.
- *Block Timestamp manipulation*: Miners decide the timestamp of the block and they can use that ability to trick smart contracts. Contracts in Solidity should be tolerable at least a 15 seconds variation; the time that takes to mine a block.

- *Uninitialized storage pointers*: The default empty values could lead to unexpected behaviours, so it is a good practice to initialize variables. From Solidity version 0.5.0 on, storage pointer should be initialized.
- *Non upgradable smart contracts*: Smart contracts are by definition immutable, meaning that they cannot be modified once they are deployed. This also means that if a smart contract is deployed with a bug, it cannot be fixed. Solution: Always use upgradeable patterns for your smart contracts (transparent agents, responsible agents, beacon agents, etc...).

You don't need to implement it, you can reuse the function provided by openzeppelin for example. You might also want to add the "Pausable" pattern to some of the contract's functions. The "pausable" function can be paused, which will give you more time to fix a bug and deploy a new version.

3.8 Tokens

Tokens by its common and traditional meaning, were used to refer to privately issued coin-like items that represent a special ownership or service, like the laundry tokens, arcade game tokens, collectible tokens for a raffle. With the blockchain, the purposes of tokens have exploded and that tokens can be traded for each other or for other currencies in exchanges and liquid markets.

In the first place, it is necessary to introduce some concepts like fungibility to analyze the properties of tokens in Ethereum. Fungibility can be defined as the following: "*In economics, fungibility is the property of a good or a commodity whose individual units are essentially interchangeable.*" [28]

Tokens can be substituted for one another without any loss of value, when they are fungible. If the history or origins of a token can be determined, then it is not entirely fungible. The ability to track the origins of digital assets can lead to the blacklisting and whitelisting of those assets, reducing or eliminating their fungibility.

In opposite, the non-fungible tokens unique and they are devoted to represent a tangible or intangible item thus two non-fungible tokens are not the same and they are no interchangeable. As an example, we can have a token that represent the ownership of a land and a token that represents the ownership of a crypto-art, they are not easy interchangeable and they do not have the same value.

3.8.1 Intrinsicity of tokens

By definition, intrinsic means "belonging naturally; essential."

Some tokens are created on the blockchain and they are intrinsic to the blockchain. This property is really desirable to avoid counter-party risk, since the consensus rules apply to the network as a whole, this same rules apply to the assets belonging to the network and thus making no need for third party holders. This rules apply to the ownership of the tokens and is equivalent as the ownership of a BCA.

There is a distinction between tokens and ether because the Ethereum protocol does not recognize tokens. Sending ether is an action that is built into the Ethereum platform, but sending or owning tokens is not. The balance of Ethereum accounts is handled at the protocol level for ether, and at the smart contract level for tokens. In order to generate a new token on Ethereum, you have to deploy a contract associated to that token. The smart contract is responsible for managing all aspects of the agreement, including ownership, transfers, and access rights. When creating a smart contract, it is advisable to adhere to an existing standard in order to ensure optimal results. We will examine these standards in more detail later on. We will weigh the advantages and disadvantages of the following standards at the end of the chapter.

3.8.2 The ERC-20 standard

The developer Fabian Vogelsteller proposed in 2015 the ERC-20 as how to standardize the tokens within smart contracts on the Ethereum blockchain. An Ethereum Request for comments, because it was the twentieth comment, it had been assigned the designation ERC-20.

He followed the standard procedure for proposals in Ethereum. The proposal was approved and implemented in 2017 and it is known as an Ethereum Improvement Proposal 20 (EIP-20). However, the old name it is still kept, ERC-20 because that's how it was named until it was approved.

ERC20 is a standard for fungible tokens, which refers to tokens that are interchangeable and have no unique properties. The ERC20 standard provides a consistent interface for contracts that implement a token, allowing any compatible token to be accessed and used in a uniform manner. The interface is made up of a number of functions that every implementation of the standard must have, as well as some optional functions and attributes that developers can choose to add.

The ERC20 implementation is composed by two data structures, the first data mapping holds an internal table that tracks the token balances for each owner. This tracks who owns the tokens. Transfer are just subtraction from someone's balance and addition to someone's balance.

```
mapping(address => uint256) balances;
```

The second data structure is "permission" data mapping. ERC20 tokens allow the holder of the token to delegate powers to the user to spend a certain amount of money (benefits)

from the holder's balance. Become. The ERC20 contract uses a two-dimensional mapping to keep track of allowances, with the primary key being the address of the token owner. This maps to a spender address and an allowance amount.

mapping (address => mapping (address => uint256)) public allowed;

The ERC20 interface specification is the following in Solidity:

```
1 contract ERC20 {
2     function totalSupply() constant returns (uint theTotalSupply);
3     function balanceOf(address _owner) constant returns (uint
4     balance);
5     function transfer(address _to, uint _value) returns (bool
6     success);
7     function transferFrom(address _from, address _to, uint _value)
8     returns (bool success);
9     function approve(address _spender, uint _value) returns (bool
10    success);
11    function allowance(address _owner, address _spender) constant
12    returns (uint remaining);
13    event Transfer(address indexed _from, address indexed _to, uint
14    _value);
15    event Approval(address indexed _owner, address indexed
16    _spender, uint _value);
17 }
```

There are two different workflows that can be used with ERC20. The first workflow uses the transfer function to complete a single transaction. This is the common procedure to follow to send tokens from one address to another. The majority of token transactions occur in the transfer workflow. To execute the transfer contract, simply follow the instructions provided. Alice can send 10 tokens to Bob should call the function included in the smart contract that allows to make transfers, he makes the transaction including the value of the transfer and the call to that specific function. The token contract adjusts Alice's balance by subtracting 10 tokens and Bob's balance by adding 10 tokens. A Transfer event is also issued. The second option is a workflow divided in two operations one operation that uses 'approve' function followed by the 'transferFrom' function. This workflow allows a token owner to delegate the control of the tokens' contract to another address. By doing so, they can ensure that their tokens are always being managed in the way they desire, even if they are not able to do so themselves. The delegation of control to a contract for the distribution of tokens is a common use case for this technology, but it can also be used by exchanges.

In this section, we have introduced tokens that can be interchangeable. They are fungible and they have no distinction one from another. The next section will be devoted to the non-fungible tokens which play a big role in this work.

3.9 Non Fungible Tokens

All the tokens that we have talked about so far are easily interchangeable, in this section we introduce the [Non Fungible Tokens \(NFT\)](#).

NFTs works as a notary for digital and physical assets. They record the property of that particular asset on the blockchain, making it comply with the consensus rules of the network. The ownership of an NFT is verified and tracked through the use of blockchain technology, which also allows for the NFT to be sold or traded by its owner. NFTs can be created by anyone with basic coding skills. NFTs often contain references to photos, files, videos, etc. NFTs are unique assets that can be identified individually, unlike cryptocurrencies, which are interchangeable.

3.9.1 The ERC-721

We can find out the difference between ERC20 and ERC721 tokens by just examining the internal data structures used in each type of token. ERC721 tokens use a different data structure than ERC20 tokens, which allows for more flexibility and functionality.

```
1 // Mapping from deed/obligation ID to owner
2 mapping (uint256 => address) private deedOwner;
```

The ERC721 interface specification is:

```
1 interface ERC721 /* is ERC165 */ {
2 event Transfer(address indexed _from, address indexed _to, uint256
   _deedId);
3 event Approval(address indexed _owner, address indexed _approved,
   uint256 _deedId);
4 event ApprovalForAll(address indexed _owner, address indexed
   _operator,
5 bool _approved);
6 function balanceOf(address _owner) external view returns (uint256
   _balance);
7 function ownerOf(uint256 _deedId) external view returns (address
   _owner);
8 function transfer(address _to, uint256 _deedId) external payable;
9 function transferFrom(address _from, address _to, uint256 _deedId)
10 external payable;
11 function approve(address _approved, uint256 _deedId) external
   payable;
12 function setApprovalForAll(address _operator, boolean _approved)
   payable;
13 function supportsInterface(bytes4 interfaceID) external view
   returns (bool);
```


14 }

Decentralized storage

In the effort to create the decentralized application as decentralized as possible, the way to store the certificates is using a decentralized storage. This can be defined as a file storage sharing system that consists of a peer-to-peer network of user operators who hold parts of the overall data, making it resistant to data loss. These can be in blockchain based applications or any peer-to-peer based network. [29]

The properties that we should look for when choosing among the different decentralized storage providers should be:

- Persistence / redundancy mechanisms
- Data retention policy
- Degree of decentralization
- Consensus

Two of the most popular decentralized storage systems are [IPFS \[30\]](#) and [FileCoin \[15\]](#). This two services are used in this work and they are explained in the next sections.

4.1 IPFS

IPFS is a distributed system that has been nurtured from the aspects of successful previous peer-to-peer like BitTorrent, Git, DHTs, and SFS. IPFS has the potential to simplify and improve upon existing techniques for writing and deploying applications, as well making easier the distribution and tracking of version of large storage data. IPFS could potentially evolve the web as we know it. IPFS is a peer-to-peer network, where all nodes are equal. The nodes store the objects from the network in their local storage. Nodes communicate with each other by exchanging information through established connections. These objects represent different types of data structures, such as files.

The decentralized property of the system in a real case scenario applies even when you are downloading a file. Let us say that you download a copy of a popular blog from a distant node that has it, whenever someone nearby you tries to download the same popular blog, it may download it from you. That is the way IPFS incentives decentralization and speed. It also provides resistance to censorship, so that if the information is stored in many anonymous nodes, thus it makes it more difficult to completely delete it. Storing the information in multiple machines, makes the information more resilient to attacks and server crashes.

When using IPFS, it is important to remember that the system is designed to be used collaboratively and that all users play a role in its success. If no one using IPFS has anyone else's access to the content identified by this address, you won't be able to retrieve it. A content cannot be deleted from IPFS if someone is still hosting that content, making it accessible to the network. This can be an beneficial or a problem depending of the repercussion of the file. It is the same thing as social media, whenever something is uploaded to the let say Twitter, is really difficult to make it unavailable forever.

Speeding up the web can be beneficial when you are far away or disconnected from the main network. If you can retrieve a file from someone who is geographically closer to you, you can often get it faster. This is even more important to devices and/or machines that have limited access to the internet, that is they cannot access the entire internet. This can happen because of censorship of a specific government agency or simply a bad internet connection. The decentralized storage system may solve this problem better than centralized systems since you do not have to reach a unique destination (centralized system) to get your information and you may get it faster because of proximity.

The IPFS address do not identify the node, they are linked to the file content that it is store. This address is form by the cryptographic hash of the contents. Even that, the hash is short compared to the size of whole file, is it unique for each file. As an example, the commonly used SHA-256 hash function, outputs a hash of 256 bits, which can generate 2^{256} which is in decimal base $1.157e77$. In comparison, the number of atoms in the observable universe are estimated to be between $10e78$ to $10e82$.

The IPFS protocol is divided into a bunch of sub-protocols, each of which is responsible for different functions:

- **Identities:** Administer the node identity generation and verification. They are identified by the parameter `NodeId`, created with `Kademlia` static cryptographic hash table. The result of that `Kademlia` function get us the public key of the node. The users of the network can change their address on demand, though they will lose some benefits on changing it.
- **Network:** Uses the underlying network protocols to manage connections between peers. The following characteristics are provided by this module:

- Transport: IPFS supports any transport protocol, though is best suited for WebRTC.
 - Reliability: IPFS can provide this characteristic if the underlying protocols are not able to.
 - Connectivity: IPFS provide [Interactive Connectivity Establishment \(ICE\)](#) for negotiating NAT traversals when establishing peer-to-peer communication sessions.
 - Integrity: There is the option to check the integrity of the hash checksum of the messages.
 - Authenticity: It can provide HMAC verification of the messages with sender's public key
- Routing: The routing strategy followed in IPFS is based on S/Kademlia and Coral and by using [Distributed Hash Tables \(DHTs\)](#).

- Exchange - The BitSwap protocol is the protocol that governs the block distribution efficiently. It is inspired in the BitTorrent protocol. In BitSwap like in BitTorrent, offer a set of blocks in exchange (`have_list`) of a set of blocks (`want_list`) that other peers are seek to acquire. Unlike BitTorrent, the minimal tradeable units are not files, they are the blocks.

The BitSwap protocol works as a marketplace where a node can get the blocks it need, independently from the file the block were got from. That is the blocks that compose a file can come from completely unrelated files in the file system.

- Objects: On top of the [DHTs](#) and BitSwap, IPFS builds a Merkle [Directed acyclic graph \(DAG\)](#), which is a data structure that uses cryptographic hashes to link objects together. This makes it more secure and efficient than other data structures. This is a more elegant and persuasive way to store data in Git. This type of data structure comes with the following properties:
 - Content Addressing: All content on the IPFS network is uniquely identified by its multihash checksum, which includes links to other content.
 - Tamper resistance: A checksum is applied to the content to verify if the data is tampered or corrupted.
 - Deduplication: All objects with exactly the same content are equal and stored only once.
- Files: This layer is settled on top of the Merkle [DAG](#), it defines a set of objects for modeling the versioned filesystem. The object model is comparable to the object model of Git:

- Block: Variable-size block of data
 - List: Collection of blocks or other lists.
 - Tree: Collection of blocks, list or other trees
 - Commit: Snapshot in the version history of a tree.
- Naming: A self-certifying mutable name system. It allows for the publication and retrieval of immutable objects. This objects can be tracked by version history. However, it lacks a critical component: mutable naming. If we did not have IPFS, we would have to communicate new content using other methods that are not as efficient. IPFS links make it possible to send content quickly and easily. What is required is some way to retrieve mutable state from the same location. It is important to note that, even though we went to great lengths to build an immutable Merkle DAG, there are situations where mutable data is necessary.

The IPFS is a really good system to keep files for a certain period of time although is not recommended for long term storage. This system does not have a built-in incentive scheme, a hybrid solution that introduces the use of contracts to incentive the persistence in the long run should be implemented for long term persistence.

Pinning services provide a way to keep data stored on IPFS even after it would normally be removed. Thus the data persistence can be compromised. Therefore, in this work we have included a contract-based decentralized storage solution FileCoin.

4.2 FileCoin

The Filecoin network is a decentralized file storage system that incentivizes users to store files reliably over time. In Filecoin, users are charged a storage fee for storing their files on a storage provider's network. The storage fee is used to incentivize storage providers to keep users' files stored on their networks. Storage providers can join Filecoin to earn rewards for providing storage to the network. The price and availability of storage is not determined by any one company. Instead, Filecoin facilitates an open market for file storage and retrieval that anyone can participate in. The Filecoin network is a decentralized blockchain ledger with its own native cryptocurrency (FIL). Storage providers earn FIL tokens by providing storage space for user data on the network. The Filecoin blockchain contains a record of all transactions involving the sending and receiving of FIL, as well as proof from storage providers that they are storing files correctly.

As we can see in the image, the network is mainly divided in two actors: users and storage miners.

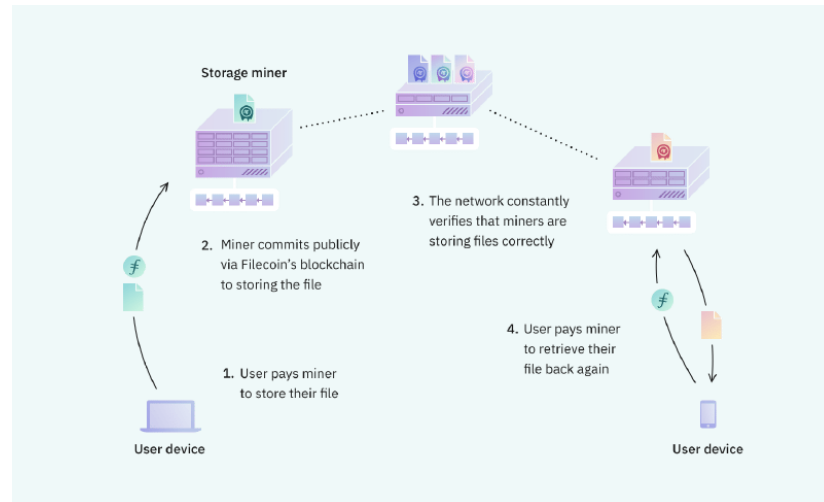


Figure 4.1: FileCoin workflow

4.2.1 Users

The network offers the users to make your selection between the main constrains of the storage systems, redundancy, cost and speed by choosing the storage provider. This tailored suit reduces reduces the cost when of the characteristic are not needed, thus offering really competitive prices. Another advantage of the network is that everyone that is participating on the network should "talk the same language" that is, every node of the network implements the same protocol. That makes the communication between users and different storage providers portable.

4.2.2 Storage miners

Filecoin is a decentralized storage network that allows storage providers to sell their free storage space on a open market. This enables providers to compete on price, making storage more affordable for users. In fact, FileCoin pricing is quite competitive. Storage providers are people and organizations that manage storage networks, earn Filecoin tokens for providing this service, and a Filecoin storage provider can be any Internet-connected computer with spare disk space, or a dedicated system with large amounts of storage dedicated to Filecoin. The Filecoin protocol incentivizes storage providers to contribute useful storage to the internet, rather than engaging in wasteful proof-of-work computations. When a provider implements the Filecoin protocol, they can connect to the Filecoin network and offer their services to users. The Filecoin protocol and network provide a marketplace for storage providers to advertise their services and connect with users. This ecosystem of storage providers is decentralized and removes entry barriers for independent providers.

4.3 How it works

The Filecoin Network composed by peers that have different roles. Secure channels enable peers to communicate with each other and distribute information efficiently, even when there are thousands of peers involved.

Each node on the network is synchronized with each other and every node validates the messages in every block. After applying the verification, a global state is provided to the network. Nodes are able to handle storage and retrieval requests from FileCoin storage providers. They can also pay them as they execute the deals.

A FileCoin node is a task that does not require a low amount of resources but it should be running non stop.

4.3.1 Storage providers

The storage providers play an important role in the network by executing different types of deals and appending new blocks to the chain. They are rewarded with FIL, the token of the network, for their efforts.

4.3.2 Deals

Filecoin operates through two types of deals: storage deals and retrieval deals:

Storage deals are agreements between clients and storage providers that enable clients to store data on the network. Once a deal is initiated, and the storage provider has received the data to store, it will repeatedly demonstrate to the chain that it is still storing the data in accordance with the agreement so that it can collect rewards. If not, the storage provider will be penalized and lose FIL.

The agreements made to retrieve the data from the network are called retrieval deals, they are made between clients and retrieval providers to extract data that's stored within the network. These deals are fulfilled off-chain, that's out of the network infrastructure, using payment channels to incrementally procure the info received. This reduces the employment of the network resources thus reducing costs of operation.

4.3.3 Proofs

As mentioned above, storage providers must prove that they're storing the info per the terms of a deal. Meaning that:

- All the information received from the client should be stored
- This data should be stored throughout all the amount that's stated within the deal

To fulfill these requirements, cryptographic proofs are used.

The **Proof of Replication (PoRep)** mechanism allows the storage providers to prove that all the data has been received and that it is encoded in a way that is unique for that specific storage provider. This operation is done at the start of a deal, then the sealing operation is done.

The storage provider will use **Proof of Spacetime (PoSt)** to demonstrate that it is still storing the data associated with a particular deal, for the duration of that deal's active lifetime.

In order to prove data availability for Proof-of-Space-Time, storage providers must show that randomly selected portions of the data they are storing are still intact. The Filecoin network relies on clients and storage providers to continuously verify the proofs included in each block, in order to maintain security and penalize storage providers that do not uphold their agreements.

4.3.4 Gas

Proofs or transactions are included to the network by using messages. This operation of aggregating new messages to the network consumes computational and storage resources. Gas is a unit that measures the amount of computational and storage resources required to execute a message or transaction. The gas cost of a message affects the amount the sender has to pay for it to be stored on a blockchain.

In other blockchains, miners have typically specified a gas fee in terms of the native currency. This approach has a number of advantages, including providing a clear incentive for miners to continue processing transactions and ensuring that transactions are processed in a timely manner. They then pay the miner that includes the transaction to the ledger a priority fee based on how fast they want the transaction to be included in the transaction pool, since miners will select the most cost effective transactions first, and they include the transaction base fee. The Filecoin network functions similarly to Ethereum, except that a portion of the transaction fees are burned in order to compensate for the resources expended by the network nodes. This is based on Ethereum's EIP1559.

The Filecoin network features a dynamic BaseFee that is automatically adjusted based on network congestion parameters (block sizes). This ensures that an appropriate amount of fees are burned, providing elegant and persuasive fee management. This allows for the network to more efficiently manage fees and prevent network congestion. The current value of a cryptocurrency can be obtained from a block explorer, which is a website that provides information about cryptocurrency transactions, or by inspecting the current head block.

4.4 PowerGate

Running a client that can connect to the IPFS and FileCoin network can make an overhead in the resources for small devices like IoT. This work utilizes a dockerized private API that can be deployed in any system that supports docker. That removes the overhead of running that on the small devices which now have to communicate to an API through HTTP.

Powergate is defined as a set of tools and configurations that can be used together or separately to integrate Filecoin into your application or storage system. This tool manages Filecoin wallets, storing each address and its configuration data privately for each user.

The main benefits of this API is that it supports both networks, FileCoin and IPFS. It ensures high and easy availability with the IPFS network and long term resilience, storing the data for the long run in FileCoin. It can also handle multiple FileCoin wallet addresses.

The good thing about powergate is that it combines both networks in a way so that one complements the other. IPFS is called "Hot Storage" it is used by powergate to retrieve the archive fast, since it is faster although it cannot be used for long term since it is not reliable. This shortcoming is covered by the connection to FileCoin which is has a cost on each use and it is slower, however it is reliable for long term storage. The distribution of the files between the "cold" and "hot" storage is done by Powergate, making it a great tool for managing decentralized stored files.

This chapter introduced the decentralized systems that we are used in combination. The next chapter is devoted to [PUF](#) and the existing different types.

Physical Unclonable Functions

PHYSICAL Unclonable Functions are like the genome sequence of the silicon chip of a device. This kind of functions use the variations on the integrated circuit during the manufacturing process of the chip to uniquely identify it. Even if the whole process of manufacturing is carefully monitored, these imperfections are introduced. This unique characteristic could be used for identification and even authentication of a particular circuit which can be particularly useful in RFID and SmartCards. The specific features of a circuit are usually formalized in terms of input-output behavior. Strong PUFs have a requirement that each circuit has a large number of features, making it virtually impossible for an attacker to copy and imitate all of them. When an electrical signal is transmitted on two circuits that are completely symmetrical, there can be significant differences in the delay times incurred. SRAM cells have totally different start-up values attributable to method variation, as well as planographic printing variations in effective feature size and random threshold voltages. In general, the purpose of PUF is to extract the randomness of these manuals.

The idea of physical unclonable functions was first introduced in 2002 in this article. [9] As for now, they are being introduced in IoT devices and industrial devices where low energy consumption and lack of computational resources is the normal operational environment.

Throughout all of this chapter we will introduce the different common types of PUFs at present, that include Arbiter PUF [31], Ring Oscillator PUF (RO PUF) and SRAM PUF. Afterwards, we will discuss the different benefits and drawbacks on each one. In this work we will not go deep into the details and error correction mechanism of the different physical unclonable functions. A simulation provided by the library pypuf [32] of one type of PUFs, Bistable Ring PUF will be used in this work.

5.1 Arbiter-based PUF

An arbiter PUF utilizes the inherent timing differences between two symmetrically designed circuit paths to produce a single bit of the response at the output of the circuit. It consists of multiple selectors connected between each other in the way that the output of previous is the input of the next and an arbiter at the end. Each stage of the challenge consists of two outputs and three inputs. The single bit of the challenge is combined with the two outputs from the previous stage to create the final output. The inputs of the first stage are connected to a common enable signal.

The outputs of the last stage are connected to an arbiter, which determines which signal arrived first. Based on the responses on all of the stages, the arbiter outputs a single bit response which is known as the response. The schema of this type of PUFs can be seen in Figure 5.1.

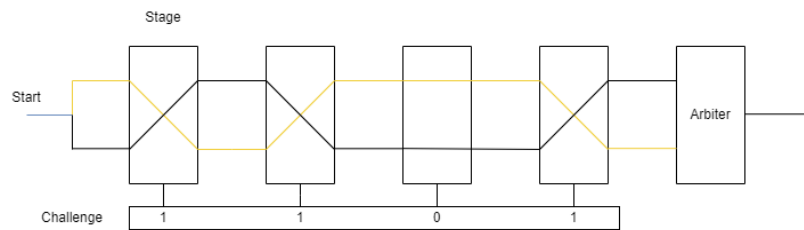


Figure 5.1: Structure of an PUF arbiter

The PUF arbiter's circuit has an X input with multiple bits, and it calculates a 1-bit Y output based on the delay difference between two paths of the same length. The challenge bits, the ones feeding the MUXes from the bottom in the picture, control the delay paths, therefore producing different responses given the same input. Here, a pair of MUXes controlled by the same input bit $C[i]$ work as a switching box.

The selectors have to go through two delay signals from the left side if the input control bit $C[i]$ is zero. Otherwise, the top and bottom signals are switched. In this way, the circuit can create a pair of delay paths for every input C . To judge the output for a specific input, a rising signal is given to both paths at the identical time, the signals race through the 2 delay paths, and also the arbiter (latch) at the tip decides which signal is quicker. The output is one if the signal to the latch data input (D) is quicker, and 0 otherwise.

This PUF delay circuit has two methods of generating a response of n -bit size from this 1-bit output circuit. First, one circuit can be used k times with different inputs. A challenge is employed as a seed for a pseudo-random number generator (such as a linear feedback shift register). Then, the PUF delay circuit is evaluated k times, using k different bit vectors from the pseudo-random number generator serving of the input X to line the delay path. It's also

possible to duplicate the single-output PUF circuit itself multiple times to get k bits with one evaluation.

5.1.1 Discussing Arbiter PUFs and its compositions

The arbiter physical unclonable functions have the particularly that they could be implemented using the common CMOS manufacturing process [10] so they are relatively cheap. However, this kind of PUF are prone to machine learning attacks, which might emulate the response as our library is doing. Machine Learning needs an oversized data set of challenge-response pairs CRP to accurately predict the response. Non-linearities can be introduced in the response of the PUF to reduce the quality of the responses of the machine learning based approximations. [33]

That is why stand-alone APUF by itself is not deployable because of its severe vulnerability to modeling attacks, PUF arbiters still offer unique advantages although they are simple by design, low hardware overhead, and being amenable to be VLSI implementation. [34]

As a result, APUFs are still used as a foundation for more secure APUF compositions, the most popular of which are XORPUFs, Lightweight Secure PUFs (LSPUFs) [35], Multiplexer PUFs (MPUFs) and their variants cMPUF and rMPUF.

5.2 Lightweight Secure PUF

As we saw in previous sections the PUF arbiters are vulnerable to machine learning attacks. The combination of permutations and XORs has been proven to reduce the resistance against machine learning attacks. The name after this technique is called "Lightweight secure PUFs". However, in [36] it was found that this type of PUF was not as resistant as it was thought. In fact, they can be simulated as easy as PUF XOR arbiters.

5.3 Ring Oscillator PUF

The PUF delay circuit in Figure 5.2 [37] is made up of many ring oscillators that are all laid out in the same way. Each ring oscillator is a circuit that produces a periodic signal with a specific frequency. Due to manufacturing differences, each ring oscillator oscillates at a slightly different frequency.

In order to generate a fixed number of bits, a fixed sequence of oscillator pairs is selected and their frequencies are compared in order to generate an output bit. The output bits from the same sequence of oscillator pair comparisons will vary from one chip to another. If the oscillators are arranged in the same way,

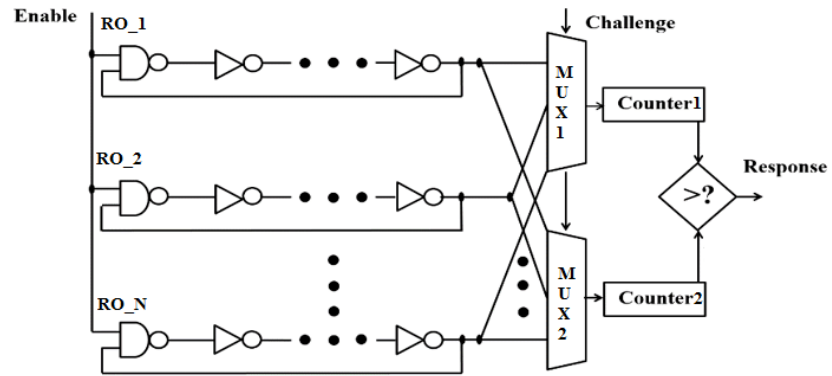


Figure 5.2: Ring oscillator structure

This is the type of PUF used in this work since they are the most secure of the one that can be simulated with the pypuf [32] library.

Infrastructure for certification of IoT devices

THIS work was born from the idea of decoupling the certification process from the CA. The IoT devices were taken as the devices for certification since they are being rapidly introduced and there are increasingly being the target for attacks. Most of them are simple devices and many of them use proprietary protocols that do not implement or implement weak secure communication mechanisms. [38]

The blockchain technology, that is nowadays almost common knowledge for everyone, it offers the magical consensus rules between anonymous parties in a decentralized system. The old Byzantine Fault Tolerance (BFT) problem was solved by this consensus algorithms since every participant in the network can verify the information contained in the blocks in a common and objective way. Besides, it offers data integrity and authenticity of the data trail added to chain of blocks by using hashing and signing algorithms and application code that can be securely executed and verified on-chain.

From that properties of the blockchain, the following question arose to me: *why not associating an IoT device directly to a blockchain address?* It will provide the device with a public key, data integrity and data authenticity. Well, communicating using the data storage mechanisms of a blockchain network like Ethereum can be quite expensive and inefficient. The time that it takes to add a new block of data to the chain is at least 15 seconds which is unfeasible for communications. Traditional protocols of communication like DTLS and TLS which are heavily supported and tested may be used in combination with the blockchain properties.

Let us say that we are going to use usual communication protocols and we associate the device with an address so that we have a public key linked to that address. The public key linked to the blockchain address can be used for data encryption and signing, from that address we can obtain a public key that can be used for secure communications and we have our problem solved. Well, it is not that easy. One of the key aspects in the security of IoT devices

is the authentication and identity management [38]. The IoT device needs to be able to confirm that the person behind an address of a blockchain is authorized to access or communicate with the device. Authorization and access control is key in establishing a secure connection between multiple devices and services. The identification of the real entity or person behind a BCA is not an easy task in blockchain networks.

The Ethereum network was built based on the idea of decentralization, making a difficult task to confirm the real identity behind a blockchain address. In fact, the identification of the user that is behind a certain blockchain address is almost impossible without the intervention of a third trusted party or some consensus between parties. This is the same solution traditional solution as the proposed with the CA, however, there is another way to identify devices without third parties.

PUFs are the solution to identify the devices, they offer a way to uniquely identify a device using their hardware characteristics. The imperfections introduced in the silicon chip during the manufacturing process work as the true identification number of the device. This functions have a challenge-reponse mechanism that can be used to generate private keys [39]. However, it is not one hundred percent problem-free, as seen in the previous chapter, some transformations should be applied to make them as secure as possible [40] and usable for key generation.

Error corrections mechanism should be applied to dissipate the noise of the response. After applying such mechanisms, the response can be use as a private key. Using that private key, a public key can be generated and a public blockchain address can be derived from it.

Now, we have a blockchain address for our device, it is linked to a blockchain address through the use PUF. It is time to set the ownership of the device by using the NFT tokens explained in the chapter number three. The role of ownership of a device can be used identically to the administrator role and can be programmed in the devices' logic. The ownership can be tracked during time and cannot be tampered with since it is ruled to the consensus rules of ETH.

In other to attach, the owner blockchain address to a device, the "ownable-iot" contact explained in Section 6.2.1 is used to act as a notary, registering the ownership without the use of a third party. Thus, possibly eliminating the need of CA.

The Root of Trust (RoT) is shifted from the CA to the manufacturer and physical devices. This is explained in later section 6.2

The infrastructure proposed is mainly based in this article [11] but it removes some of the characteristic proposed that we consider as limiting. The need of actively participate in the blockchain is some feature that may not be desirable for some devices and the obligation of registering each user that communicate with the device in the smart contract's logic can ruin scalability. In fact, the smart contract's programming logic limits the number of users that

a device can talk to, to be one. There is a single variable in each token that tracks the user address. Since, a single token can only be associated to a device, a single user can only talk to a device. A mapping can be used to track the user's that can talk to a device. However, we decide to remove this workflow as we consider it an overhead. This work delegates the user authentication to the commonly used x509 certificates authentication.

Besides, some of the variables are recycled and have been given another use. The timeout variable is no longer used to verify if the device is available. The device is forced to update its state which the timestamp bound, thus having to interact with the blockchain every timestamp period of time. The timeout parameter now is used to set the maximum age of the shared secret.

Moreover, it is included a new functionality that allows the owner to reset the shared key between owner and device since [11] did not implement it. A new function is included in this proposal that allows the owner to reset the device shared secret.

The certificates creation, and distribution is also handle by this infrastructure in a way the the integrity and authenticity of them are preserved. Having each device linked to a certificate, allows us to implement many other, like [Network Access Control \(NAC\)](#) or 802.X EAP authentication to robust the security of the enterprise infrastructure.

This work comes with an user guide is for the interfaces and the process of securing a device is included in the appendix.

6.1 Actors

Three main actors are defined in the infrastructure:

- Device's manufacturer: The manufacturer is the owner of the smart contract. It is the responsible of deploying the contract and creating the secure tokens. This secure tokens represent the ownership of the device.
- Device's owner: The owner of the device. It can perform the engage operation, set the devices certificate signature and transfer the ownership to another address.
- Device: The secure device that is represented by the token.

The device is supposed to interact directly to the blockchain however we just included the simulated device into the manufacturer web interface since we do not have access to a real PUF and it does not add any additional value having it decoupled from it in our proof of concept. The manufacturer has a web interface that implements the functions associated to its role. Another web interface, is provided to the owner to perform his/her operations.

6.2 How it works

The infrastructure is composed by an smart contract deployed in the Ethereum network, a Powergate machine that offer connection to the decentralized storage systems and two web interfaces that interconnect the different components and handle the actor's functions logic behind the scenes. The two interfaces that are exposed are used by the owner and one used by the manufacturer.

The smart contract handles the key agreement and the ownership workflow of the device. It works as a notary, like a trusted party that assure that the process is correctly done. It provides authentication by stating in its logic an access control mechanisms where the only the specified actors can interact with the functions in it. Integrity is also provided to certificates since the storage of the hash digest of the certificates is done in the contract. Thus, every device that connects to that device can verify in the contract that the hash that is has received is correct.

Authenticity is also provided even if we are using the hash certificates. This *int256 hash* stores the certificate's digest. You may be wondering why we are storing the hash and not some other data like an HMAC or a signature that is more secure. Well, the truth is that the data is added to the contract through a transaction that is signed and later an access control mechanisms verifies that the sender of the data is the owner of the device, so that authentication is provided and there is no need for a signature or similar. Moreover, the hash perfectly fits of a uint256 variable, thus reducing the gas cost of the operations.

The connection to the [IPFS](#) and FileCoin networks is offered by Powergate. The dockerized version of this product is used and it is installed and running in a machine that is devoted of only handling the connection to the decentralized storage systems. This functionality is used for storing, for each device, the ciphered certificates and permissions file that states the devices to which that device can talk to. In the diagram of Figure 6.1, the infrastructure can be visualized.

Having the visualization of the process in Figure 6.1, the following workflow of registering works as follows:

1. The manufacturer deploys the contract
2. The manufacturer creates a token associating the owner address and the device address
3. The owner starts the engagement process
4. The device completes the engagement process
5. The owner creates a certificate associated to the device
6. The certificate is uploaded to the decentralized storage systems through Powergate API

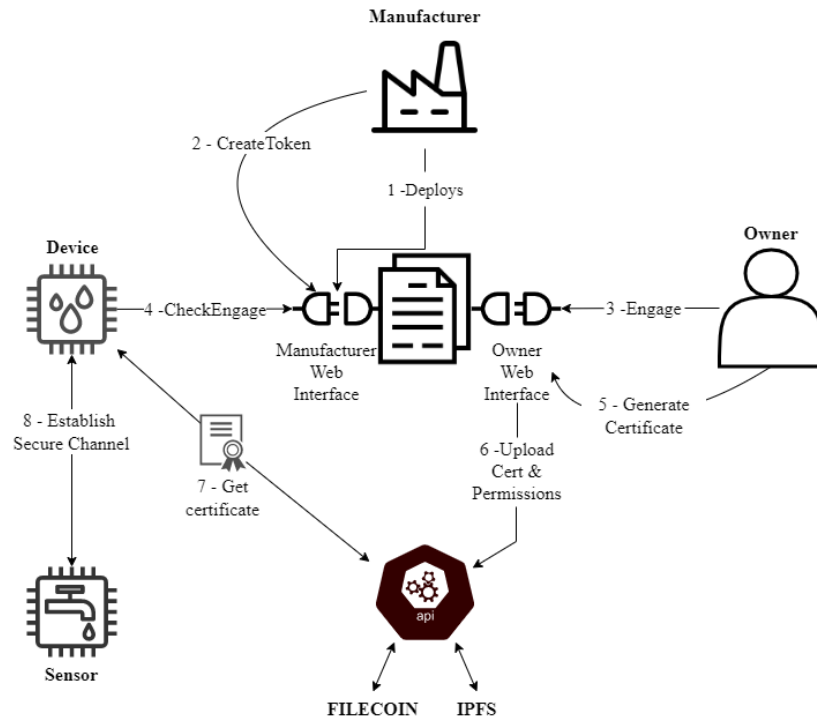


Figure 6.1: Infrastructure proposed

7. The device gets the certificate and verifies it.
8. A secure channel can be established with other secured devices

This schema can be similar to the Intel’s SDO substituting the centralized parties for decentralized parties. [41]

These steps are the generalization of the process of securing a device. Each component will be treated separately to analyze its behaviour in the next sections.

6.2.1 Smart Contract

The `ownable-iot.sol` smart contract, inherits from the interfaces `ERC721` and `smartNFT`.

```
contract OwnableIoT is ERC721, smartNFT
```

The `ERC721` is the standard interface for NFT token which was used as the base for this contract. The other functionalities added with the `OwnableIoT` contract provide the key agreement mechanism that the standard interface does not provide. Let us compare the variations in the structure of the `SmartNFT`, `OwnableIoT` and `ERC721` in Table 6.1.

The process of engagement starts right after the token is deployed. The state of the secure token afterwards is set to “Waiting for Owner” and the device is waiting for owner authentication. The owner then, authenticates against the contract using the [Elliptic-curve](#)

Data Type	Name	Defined in contract
<i>uint256</i>	tokenId	SmartNFT & OwnableIoT & ERC721
<i>address</i>	owner	SmartNFT & OwnableIoT & ERC721
<i>address</i>	SD	SmartNFT & OwnableIoT
<i>States</i>	state	SmartNFT & OwnableIoT
<i>uint256</i>	hashK_OD	SmartNFT & OwnableIoT
<i>uint256</i>	hashK_UD	SmartNFT & OwnableIoT
<i>string</i>	dataEngagement	SmartNFT & OwnableIoT
<i>timestamp</i>	timestamp	SmartNFT & OwnableIoT
<i>timeout</i>	timeout	SmartNFT & OwnableIoT
<i>address</i>	user	SmartNFT
<i>string</i>	signature	OwnableIoT

Table 6.1: Specification of the block

Diffie–Hellman (ECDH) key exchange mechanism. The device *public key* and owner *public key* are used to generate a shared secret. The good thing about about this method is that no one has to disclose its private key to share a common secret.

Note that, we have emphasized *public key* words and it is because it may not be confused the with the **BCA**, they are related but they are not the same. As a recall from the section "Transactions" of chapter number three, the transactions in Ethereum included the parameters v,r,s that are the components of the public key. The blockchain address is derived from hashing that public key, therefore we cannot obtain the public key from the blockchain address.

In fact, when the device is on the "Waiting for Owner" state it can happen that no transaction was issued by the device and public key cannot be known. That is why the manufacturer web interface has the ability to show the public key of the device so that it can be shared with the owner.

Now that we have our public key, and the shared secret, this cryptographic primitive is hashed and this hash is registered in the contract by the owner. The same secret should be generated in both ends, the HashK_OD and the hashK_UD should be identical. The device then, gets the dataEngagement variable saved by the owner. The key is in the compressed format and it is used to generate the hash of the secret.

Finally, the smart contract checks whether the hashes are equal and if so the device is set to the state "Engaged with owner".

The user role has been removed since the SmartNFT contract only allowed to link one user to each token. This can be a limitation to a device. If a new user wanted to communicate with the device, he or she will have to go through all the process of key exchange each time the user changes. Besides, it is more of an overhead than a benefit.

The timestamp in the OwnableIoT contract has a different use than the one that was originally proposed. In this work, there is no need for timestamp for the verification of the availability of the device, in some cases is not recommended to have need the devices 24/7 connected to the blockchain network. Therefore, this attribute is used as the maximum age of the shared secret kind of like a expire date for the key.

The diagram in Figure 6.2 exemplifies the workflow of the engagement process.

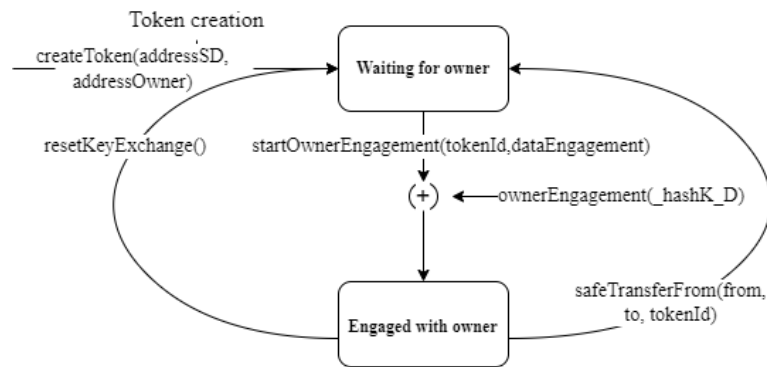


Figure 6.2: States of the contract

6.2.2 The code of the contract

The functions introduced by this work are `getCertificateHash`, `setCertificateHash`, `resetKeyExchange`, and `getDataEngagement`. The first two functions are introduced to keep track of the devices' certificates. It is required by the set function to be executed by the owner of the device. The code of these functions is quite simple:

```

1  function setCertificateSignature(
2      address _addressSD,
3      string memory _certSignature
4  ) external override {
5      require(msg.sender == ownerOfSD[tokenIDofBCA[_addressSD]]);
6      Secure_Token[tokenIDofBCA[_addressSD]].signature =
7      _certSignature;
8  }
9
10 function getCertificateSignature(address _addressSD)
11 external
12 view
  
```

```
12     returns (string memory)
13     {
14         return Secure_Token[tokenIDofBCA[_addressSD]].signature;
15     }
```

The resetKeyExchange function is added to allow the owner to refresh the key on demand.

```
1 function resetKeyExchange(address _addressSD) external {
2     require(msg.sender == ownerOfSD[tokenIDofBCA[_addressSD]]);
3     Secure_Token[tokenIDofBCA[_addressSD]].digest = 0;
4     Secure_Token[tokenIDofBCA[_addressSD]].dataEngagement = "";
5     Secure_Token[tokenIDofBCA[_addressSD]].state =
        States.waitingForOwner;
6 }
```

The last function aggregate is the getDataEngagement function which is just a simple view function to retrieve the engagement data so that is it easy for developers to get the owner public key.

The rest of the functions that are defined in the smart contract of this proposal are also defined in [11].

Table 6.2 shows the mapping of the functions in the OwnableIoT, ERC721 and SmartNFT contracts.

Functions	Defined in
<i>function transferFrom(..)</i>	SmartNFT & OwnableIoT & ERC721
<i>function ownerOf(..)</i>	SmartNFT & OwnableIoT & ERC721
<i>function balanceOf(..)</i>	SmartNFT & OwnableIoT & ERC72
<i>function createToken(..)</i>	SmartNFT & OwnableIoT
<i>function startOwnerEngagement(..)</i>	SmartNFT & OwnableIoT
<i>function ownerEngagement(..)</i>	SmartNFT & OwnableIoT
<i>function startUserEngagement(..)</i>	SmartNFT
<i>function userEngagement(..)</i>	SmartNFT
<i>function setUser(..)</i>	SmartNFT
<i>function tokenFromAddress(..)</i>	SmartNFT
<i>function userOfFromAddress(..)</i>	SmartNFT
<i>function userOf(..)</i>	SmartNFT
<i>function userBalanceOf(..)</i>	SmartNFT
<i>function userBalanceOfAnOwner(..)</i>	SmartNFT
<i>function updateTimestamp(..)</i>	SmartNFT & OwnableIoT
<i>function setTimeout(..)</i>	SmartNFT & OwnableIoT
<i>function checkTimeout(..)</i>	SmartNFT & OwnableIoT
<i>function getCertificateSignature(..)</i>	OwnableIoT
<i>function setCertificateSignature(..)</i>	OwnableIoT
<i>function getDataEngagement(..)</i>	OwnableIoT
<i>function resetKeyExchange(..)</i>	OwnableIoT

Table 6.2: Specification of the block

6.3 The workflow of the actors

In this section, that it is divided in two subsections, we are going to take a deeper look at the relations between the different actors. The first one, will be devoted to the device-manufacturer relation and the second to the owner-device relation.

6.3.1 Manufacturer

The first operation in this process is the generation of the PUF response from the device. The manufacturer sends a challenge to the device and this sends a respond back with the cryptographic primitives. This values then should go through a mechanism of cell values classification to eliminate those cells that do not produce the same output when the same challenge is applied. Afterwards, error correction mechanisms to eliminate any noise that could from the environment. The output of this is the input of a hash function that generates the private of the BCA

This private key is then used to create the public key and derive the blockchain address. This address is now used by the manufacturer to instantiate the token with the parameters of the device address and the owner address.

Afterwards, the smart contract returns the tokenId if is the device is not already registered. If it is already registered, that is the token is already created, the function also returns the tokenId

6.3.2 Owner

The workflow for the owner starts when the owner calls the engagement function. The owner knows the public key of the device and uses it in combination with the public key to generate the shared secret. Then, this shared secret is included in the function that start the process of engagement as a sha-256 digest. The device afterwards, takes the data engagement, the public key of the owner, and sets the hash of the secret. If everything goes smoothly, the same secret of the other. Thus, we have both ends sharing the same key, so that the following formula holds:

$$\begin{aligned} K_O &= SK_{OD} * PK_{DEV} = SK_{OD} * (SK_{DEV} * P) \\ &= SK_{DEV} * (SK_{OD} * P) = SK_{DEV} * PK_{OD} = K_D. \end{aligned} \tag{6.1}$$

During these process of key establishment there is no process of key exchange scheme agreement. The SECP256k1 elliptic curve is used as the default since it is the one used in the network.

After the key establishment, the owner generates a certificate that will be ciphered with the shared secret. This ciphered certificate then is uploaded to the decentralized storage so that the device can retrieve. A token is issued and posted ciphered on the smart contract. Then, the device gets the ciphered token, decrypts it with the shared secret and it will grant access to the powergate API. Thus, the device gets access to the decentralized storage system.

6.4 Web Interfaces

The web interfaces of the manufacturer and the owner are quite similar. They have been implemented using flask for the Web technology and file routing service, this library allows us to specify the routes with decorators of above the function declaration and to render HTML templates which can be reused. The option is also available to run code in it so that we can create dynamic pages.

The programs' logic can be divided in three operative modules: the module dedicated to blockchain operations, the functionality and logic of the web application and the connector to the decentralized storage. The web3.py library is used for this goal, it allows us to connect to HTTP providers and compile our solidity code using python. The rest of the logic of the program is built using general purpose libraries in combination with python code.

The way to store the data of web application, like Users and password is thought sql using a module provided by Flask, SQL Alchemy. Simple relations and entities compose the structure of the database, so it not worth it to stop on the design of it.

The last component of the program is the connectivity to the decentralized storage, the library used for this purpose is Pygate gRPC library. This is really easy to use and it abstracts all the complexity of the communication with API. However, there is a lack of documentation for it.

6.4.1 Powergate integration

The powergate API is running in a Debian local machine in a simulated local environment. The library used for integrating the flask web application and the calls to the API. In fact, the developers design this library with the concept of Web3.0 in mind and they have included in their scheme the flask module.

PowerGate besides the "hot" and "cold" storage balancing benefits, has the ability to create users that can use the API with different priviledges. In this case, the privileged user would be the owner of the devices who creates the different accounts with its tokens associated. This tokens then are distributed to the different devices so that they grant access to their certificates.

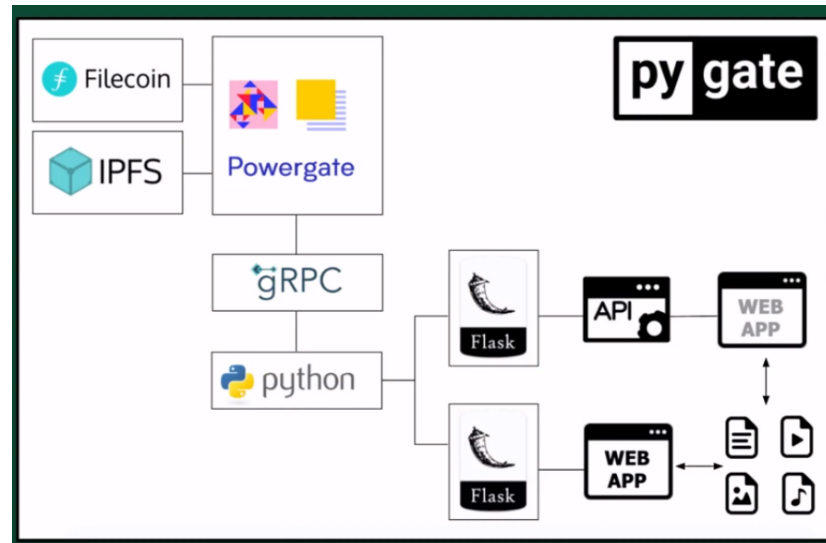


Figure 6.3: Integration with Powergate

6.5 Security aspects

The private keys of the blockchain address of the owner and manufacturer are stored ciphered in a Json format. The key is ciphered using the [Advanced Encryption Standard \(AES\)](#) algorithm in [Counter mode \(CTR\)](#) mode, which is fast and outputs a pseudo random cipher text for each plain text. This mode does not provide authentication nor integrity although it is assumed that is the key is not accessible from the outside.

It will be a preferred choice to use a hardware wallet to store the keys however this approach is out of scope of this work. The key used in the encryption is derived from the user's login password. Scrypt algorithm is used to get the 128 bit key length required by [AES](#). This algorithm is both FPGA and ASIC resistant and has been there for years, since 2009, thus its security and efficacy has been heavily tested. The Bcrypt algorithm is used to generate the password hashing of the user's password that is stored in the database.

6.6 Root of Trust

The Root of Trust was usually represented as a tree where the Root role is played by the [CA](#) that are considered to be the trusted parties. From top to bottom, we have the intermediate certification authorities that are been monitorized by the Root [CA](#) and the bottom of the hierarchy the usual web servers are placed.

The schema of this work, follows the opposite certification path, the [RoT](#) is not deposited in the centralized parties, it is the device itself. Thus, the security of the device resides in the

hardware, posing a challenge to security of the devices. The devices know have to implement mechanisms that assure their safety from stage zero of the booting process.

For that matter, secure booting mechanism and signature mechanisms provide a way to verify the firmware and code that is running in the device so that it has not been corrupted.

6.7 Description of the environment

We mainly have three components in our component in this work, although it can be modified by the person in charge of the deployment adjusting it to the specific needs.

On one hand, we have the secure device that is simulated in a Raspberry Pi 4B that is also running the manufacturer web interface, we already mention in previous sections why we merge both actors. The device is in charge of the PUF response and the manufacturer workflow.

In the other hand, we have the owner that it is running in a windows OS in my personal laptop. The owner side could be run in any workstation that the deployer wants to.

Finally, the machine that is running Powergate in a local dockerized instance is running in a VMware Virtual machine over Debian 11 OS. It is worth noticing that *Powergate is an open source brand new service and it is prone to errors and crashes.*

6.8 Development of the project

This project was developed using the spiral model methodology, the scrum agile methodology was preferred at a first glance, however, there is only one person in charge of the development of this work and there is a lack of roles for this approach. The spiral model was chosen for this development since it is the one that fits better to our project. It provides risk management which can be really determining in this type of projects that play with services and libraries that can be prone to errors and crashes.

In a nutshell, the spiral model is divided in four stages and those stages are iterated until the product is finished. Though it cannot be confused with the concept of iterating the waterfall model over and over.

The first phase of the iterations was devoted to planning of the requirements of the web application in the very first iteration. In this first iteration, we defined the minimal viable product which was composed of the engagement process of the devices. The next iterations this phase was used to realize the possible risks and rethink the objectives of the software. The users handling, contract deployment and the rest of the characteristics were included in latter planing phases.

The second phase was used to solve the possible risk that could happen during develop-

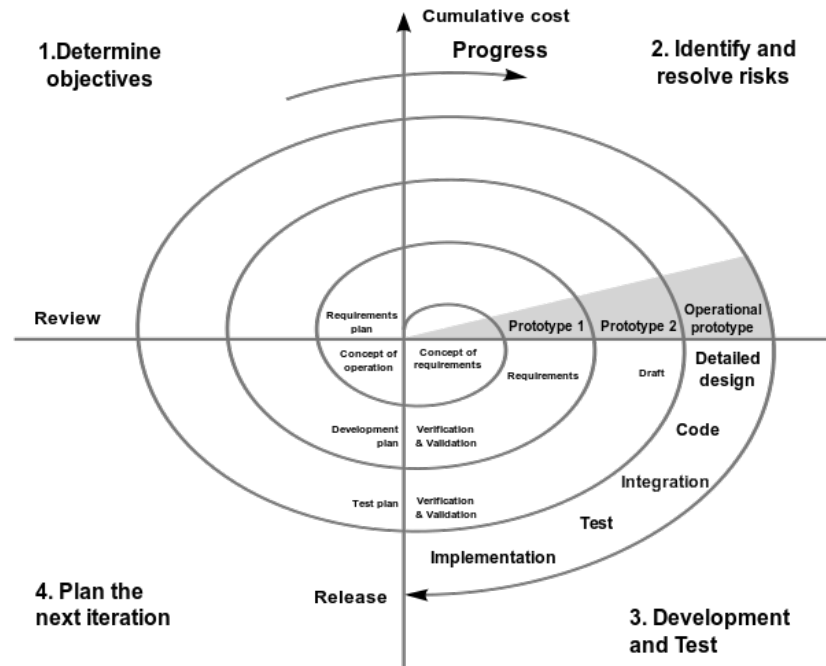


Figure 6.4: Spiral model Boehm

ment or solving risk that were identified in the previous iterations. In the first iterations the risk were related to the integration the web interface with the blockchain and the compatibility of the data types between them. This problem was identified and correctly targeted using the Web3py library and the identification of the inputs and outputs of the functions in use. The latter iterations also suffer from the same risk except from the last one which the risk was the correct integration with Powergate. The lack of documentation of the python library added more risk to the current issue and the problem was solved by testing many times the connection phase to the machine.

The third phase was devoted to development and testing. Testing was performed by filling the values in the form like a normal user would do it. If testing have to be repeated, the request was saved and repeated with a software like curl or Postman.

The last phase was reserved to planing the next iteration and make an overview of the iteration.

It is worth noticing that the most time consuming part of this work was not the development or the planning of the development. It was the design of the infrastructure that is out of the scope of the software engineering tasks.

6.9 Pricing

The cost of the Ethereum operations in gas units are the following for each function:

- createToken(): 167,263 gas cost
- startOwnerEngagement(): 69,216 gas cost
- transferFrom(): 64,556 gas cost
- ownerEngagement(): 47,962 gas cost
- setTimeout(): 28,874 gas cost
- checkTimeout(): 26,429 gas cost
- updateTimestamp(): 28,162 gas cost
- setCertificateDigest(): 21,916 gas cost

The cost of storage of the certificates is minimal since the certificate storage size is less than 10KB and the cost of storing 1TB/year in Filecoin is less than 5\$. So it is minimal. Besides, the cost of storing in IPFS is zero.

6.10 A possible continuation

This project was developed with the mindset of just exposing that the blockchain, the hardware identification of the devices and the common communication protocols can be combined in such a way that the beneficial characteristics of some make up for the deficiencies of others.

We hope that this infrastructure is continued by time. It can have a big potential if a good User Experience is given to the user though I am cybersecurity evaluator not a UX designer.

Conclusion

THE goal of the this proposal was at first to get a deep knowledge of the blockchain technology; how the transactions work, how the different parameters in the transactions play their role, the consensus algorithms used by the different network, the new concept of the Web3.0 and the smart contracts. I enjoyed the whole process of coming up to this idea and also suffer some nights without decent sleep but I think all that I have learnt pays the price. I have always curious about this technology and how it may change the future and probably the Web as we know it. After this short reflection, I would like to make a few comments about the project.

This project is no focused on being a ready to run infrastructure that you can set up. It was created as a proof of concept. It can come with many bugs since no input validation has been performed for the input fields. So it comes with no guarantee. Besides, the tools used are developed following the open source guidelines and there is no guarantee of security neither responsibility.

Regarding the development process, I have found myself stuck multiple times in the decision of the representation of the data and the data handling whether the data was securely stored, transferred and verified. However, one of the most blocking points of this project was communicating with the simulated instance of Powergate, it happens so that sometimes it did not work at all even if it was perfectly running. I maybe attribute this problem to one of the layers that constitute this application, sometimes just got stuck and stop answering correctly the requests. It is worth noticing that most of the time was expend on research and designing the infrastructure and therefore the focus of this work was not on the user experience but on the theoretical properties that arise from combining the technologies studied.

The physical unclonable functions and blockchain technology are really powerful concepts that can make a difference in the field of cybersecurity. Combining them together can two concepts can bring us closer and closer to an almost insurmountable level of security if applied correctly. Though they have are quite new and they have to refined to constitute

a fundamental branch in cybersecurity, that I think by the time will be. Hardware wallets should also be used to store the private keys and sign the transactions offline so that the keys are never loaded to a memory of a device that is exposed to the internet in some way.

More than ever before, the industry is going online and more Programmable Logic Controllers are being exposed to the Internet. Thus, the scalability and cost should be considered so that more and more critical devices are being connected to the internet. Implementing relatively cheap solutions like these ones

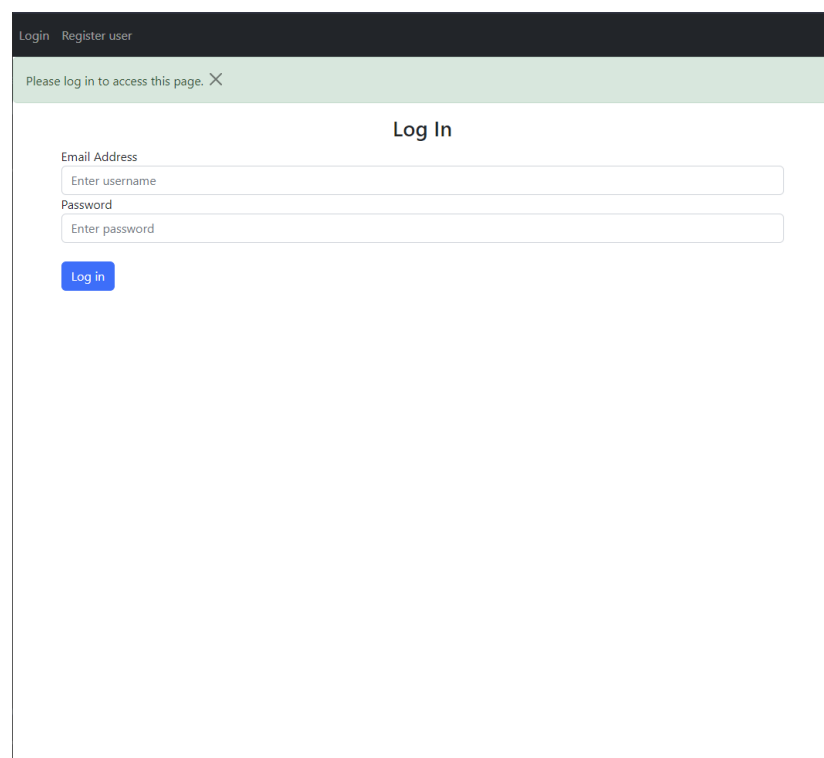
Appendices

Appendix A

User Guide

A.1 Manufacturer

THIS is the user manual for the manufacturer role First we are prompted with a message in the log in page that says that we should log in to access the functionalities. Then



The screenshot shows a web interface for logging in. At the top, there is a dark navigation bar with 'Login' and 'Register user' links. Below this is a light green banner with the text 'Please log in to access this page. X'. The main content area is titled 'Log In' and contains two input fields: 'Email Address' with a placeholder 'Enter username' and 'Password' with a placeholder 'Enter password'. A blue 'Log in' button is positioned below the password field.

Figure A.1: Log in page

we go to sign up, choose an username and a password and register our manufacturer user. Once, one user is registered, you should be logged to create other users. The same method

of registration cannot be used. We are prompted to a home page, where we should select the contracts drop-down menu and select deploy contract

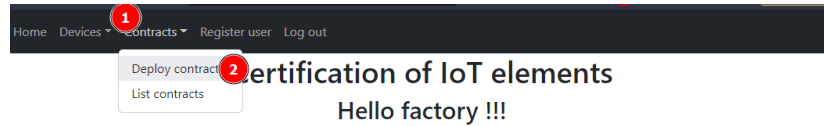


Figure A.2: Contract deployment

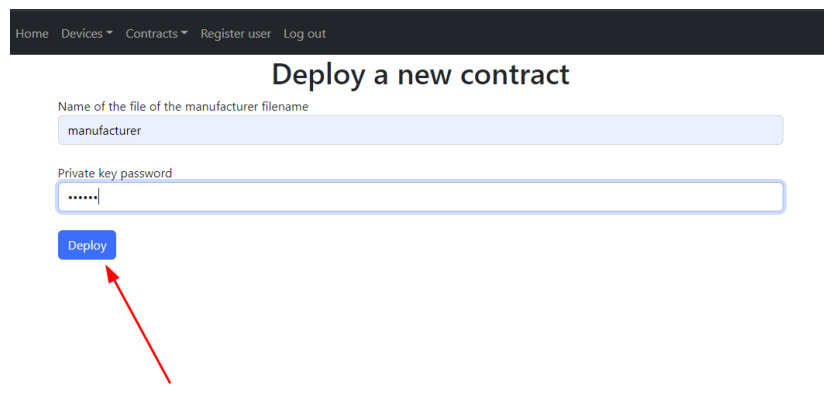


Figure A.3: Contract deployment

Then we are prompted with a message that says that the contract is created, now we have to register our device. We select the devices drop-down and click create device

On the register device page Figure A.5, we fill the inputs and select the contract where we want to register our device. Then, we will have our device registered.

We can see in Figure A.6 that right after registration our device has the "Waiting for Owner" state.

Afterwards, the owner has performed the triggered the engagement and we can see that the hash is the same as the owner's Figure A.7.

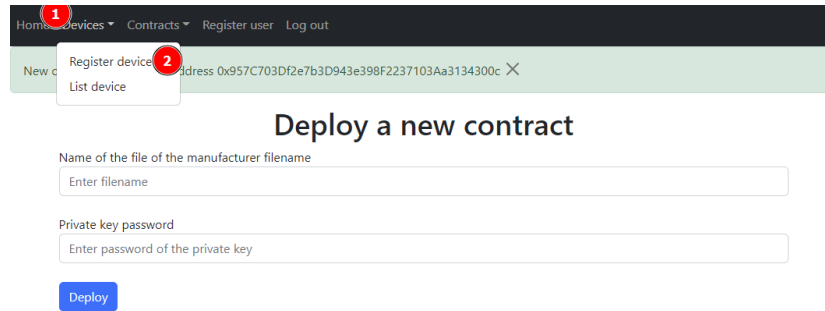


Figure A.4: Contract deployment

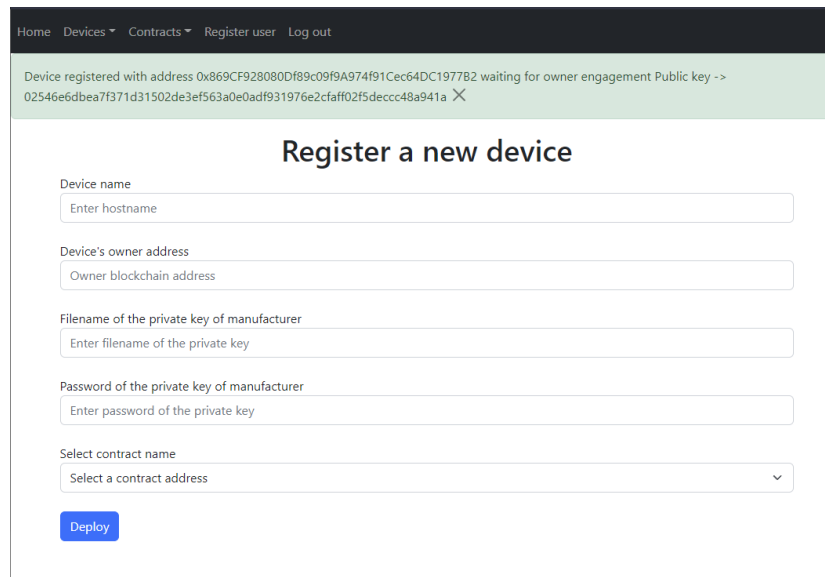


Figure A.5: Register device

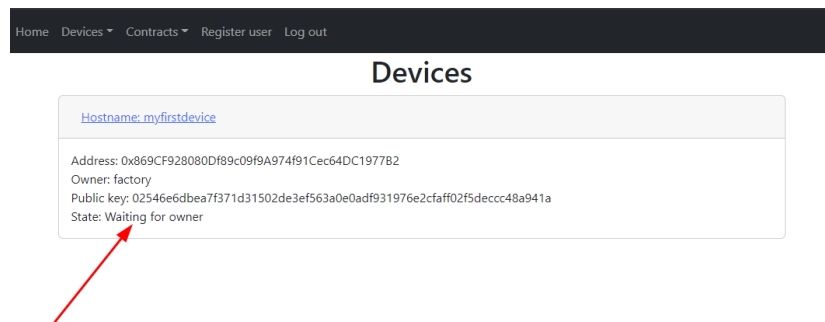


Figure A.6: List devices

A.2 Owner

The owner role follows the same path as the manufacturer for registering a user. He or She register a user that plays the role of admin.

Figure A.7: Registration of manufacturer

The account is created as we can see in Figure A.9 however the contract address is not the correct one so we have to change. The option is available to change the address.

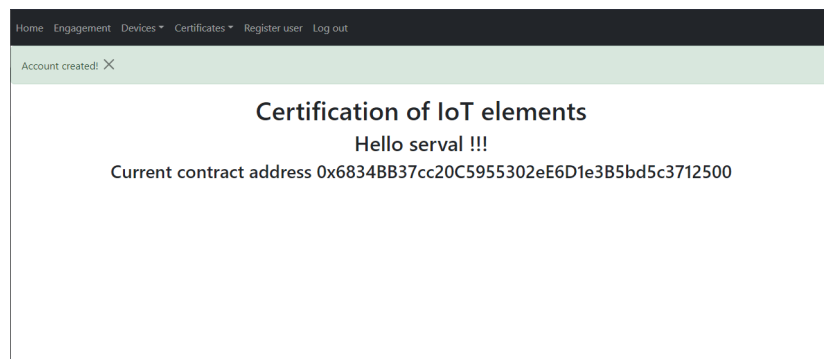


Figure A.8: Account created

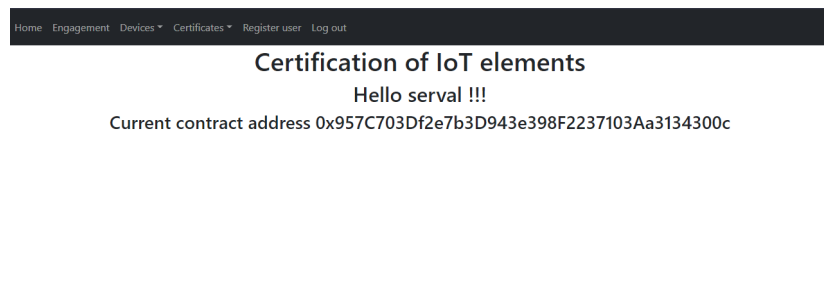


Figure A.9: Change of address

Then, we have to register the device that we want to engage to. We need for that the public key of the device and the blockchain address that we can get from the list devices interface of the manufacturer. As we can see the device is at the "Waiting for owner" state

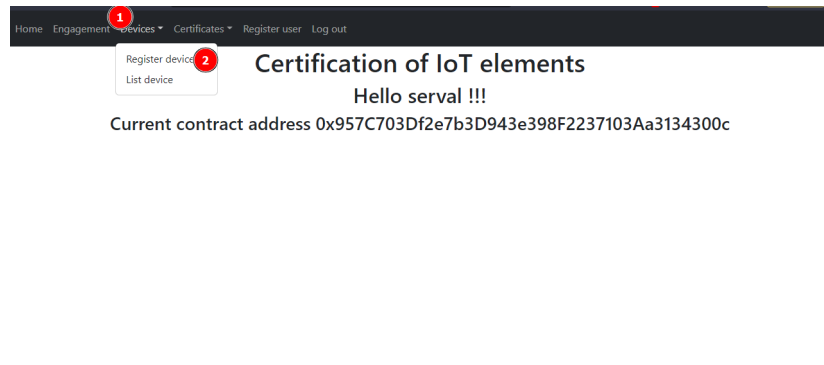


Figure A.10: Registration of device

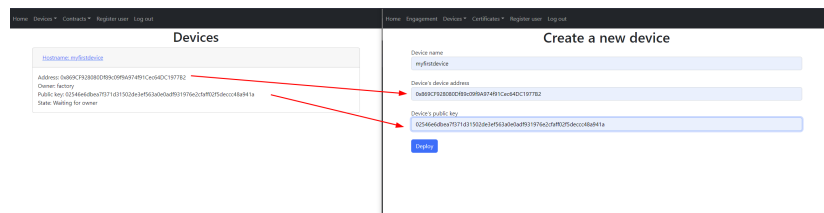


Figure A.11: Device's registration form

We click on submit and our device is registered so we can complete the engagement process.

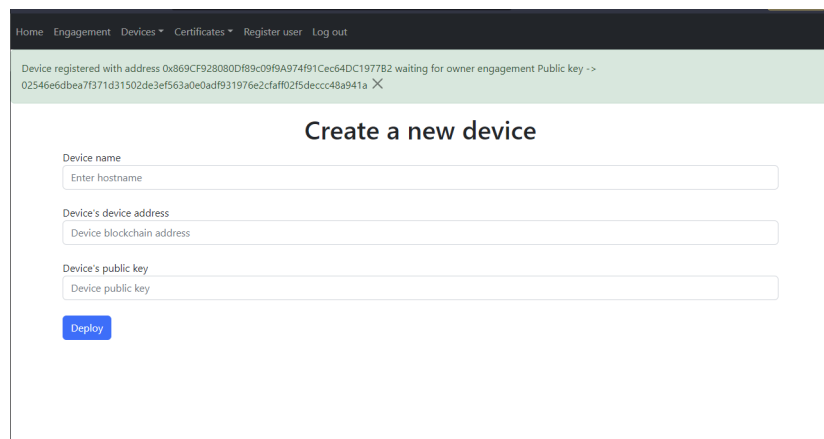


Figure A.12: Registration success

We need to go the engagement section, select the device along with owner's private key

configuration. As we can see, the password of the private key is ask in many operations, that is because we do not want to store the password or password hash of the private key since we do not have access to a hardware wallet. Therefore, authentication should be made on each operation that requires the use of the private key.

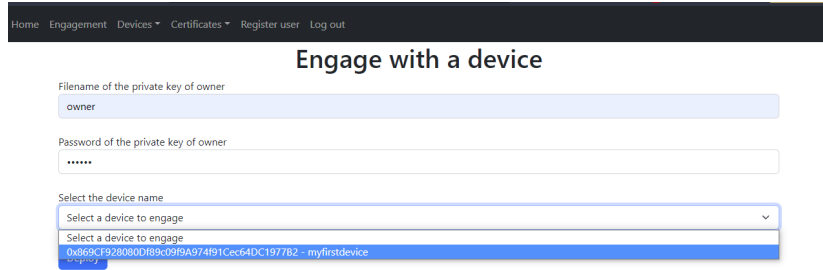


Figure A.13: Request engagement

When finished, it can be seen that a message saying "Engaged successfully" is prompted. We can see that the hashes of the owner and device match. If we go and see the listing of the

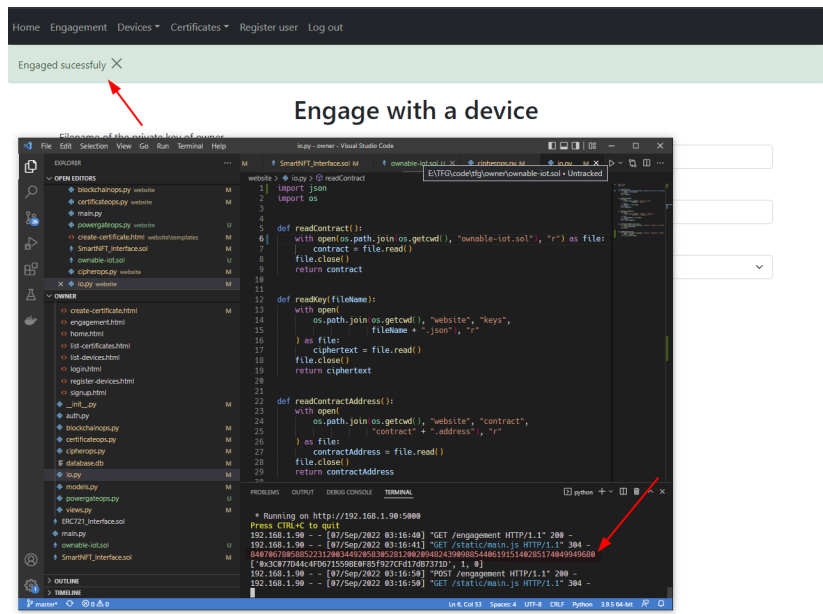


Figure A.14: Engagement finished

manufacturer, we can see that the state of the device is "Engaged with owner". The process of engagement is finished.

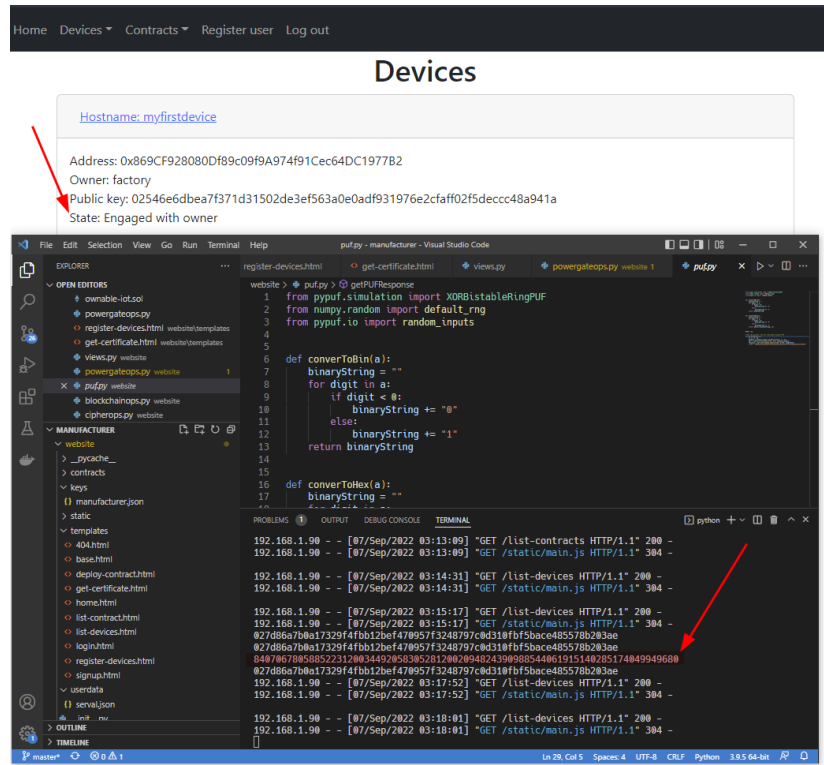


Figure A.15: Full engagement

Then our device is prepared to be assigned to a certificate, we go the certificate section, click on create and fill the form. Then a message will be prompted with the CID and user token associated to it.

Now we go to the device interface and retrieve the certificate

Home Engagement Devices Certificates Register user Log out

Create certificate List certificates **2** Deploy a new contract

Country name
ES

State or province
Enter state-province

Locality Name
Enter locality

3 Organization name
Enter organization

Common name
Enter common name, we can use the hostname of the device

Email address
Enter email

Validity in seconds
Enter validity

Password of the private key
Enter password

Select the device name
Select a device to engage

Create certificate **4**

Figure A.16: Certificate form

Home Engagement Devices Certificates Register user Log out

New certificate for device 0x869CF928080Df89c09f9A974f91Cec64DC1977B2 created ee2089d9-8e26-4777-b250-45a4cf8db0f6 CID QmYEysXpVzTgapV8TAINxVL7bytVF3d9bi1rXbC1cgpn X

Create a new certificate

Country name
Enter country name

State or province
Enter state-province

Locality Name
Enter locality

Organization name
Enter organization

Common name
Enter common name, we can use the hostname of the device

Email address
Enter email

Validity in seconds
Enter validity

Password of the private key
Enter password

Select the device name
Select a device to engage

Figure A.17: Certificate created

The screenshot shows a web interface with a dark navigation bar at the top containing the following links: Home, Devices, Contracts, Get certificate, Register user, and Log out. Below the navigation bar, the main heading is "Get certificate". Underneath the heading, there is a label "Device name" followed by a text input field containing the hexadecimal string "027d86a7b0a17329f4fb12bef470957f3248797c0d310fbf5bace485578b203ae". Below this is a dropdown menu with the selected value "0x869CF928080Df89c09f9A974f91Cec64DC1977B2" and a downward arrow. At the bottom of the form is a blue "Deploy" button.

Figure A.18: Certificate retrieved

List of Acronyms

- AES** Advanced Encryption Standard. 61
- BCA** Blockchain Address. 3, 51, 55, 59
- BFT** Byzantine Fault Tolerance. 50
- BTC** Bitcoin. 2, 5, 8
- CA** Certification Authorities. 1, 3, 50, 51, 61
- CRP** challenge-response pairs. 48
- CTR** Counter mode. 61
- DAG** Directed acyclic graph. 40
- DApps** Decentralized Applications. 26
- DHTs** Distributed Hash Tables. 40
- DoS** Denial of Service. 30
- DTLS** Datagram Transport Layer Security. 3, 50
- ECDH** Elliptic-curve Diffie–Hellman. 54, 55
- ECDSA** Elliptic Curve Digital Signature Algorithm. 5
- EIPs** Ethereum Improvement Proposals. 2
- EOA** Externally Owned Account. 18, 24
- ETH** Ethereum. 2, 7, 51
- EVM** Ethereum Virtual Machine. 2, 21–23, 25, 27, 32

- ICE** Interactive Connectivity Establishment. 40
- IoT** Internet of Things. 1, 50, 51
- IPFS** InterPlanetary File System. 38, 53
- NAC** Network Access Control. 52
- NFT** Non Fungible Tokens. 36, 51
- OTP** One-Time-Programmable Memory. 2
- PoA** Proof of Authority. 17
- PoRep** Proof of Replication. 44
- PoS** Proof of Stake. 2, 14, 16, 17
- PoST** Proof of Spacetime. 44
- PoW** Proof of Work. 2, 11
- PUF** Physical Unclonable Functions. 3, 45, 47–49, 51, 52, 59, 62
- ROM** Read-only memory. 21
- RoT** Root of Trust. 1, 51, 61
- RPoW** reusable proof-of-work system. 8
- TLS** Transport Layer Security. 3, 50

Glossary

bytecode Bytecode is a type of computer code that is read by an interpreter and converted into binary machine code. This allows a computer's hardware processor to read and execute the code. The interpreter is typically implemented as a virtual machine (VM) in order to translate the bytecode for the target platform. This makes the interpreter much more elegant and persuasive. 21

hash A cryptographic hash function is an algorithm that takes an arbitrary amount of data input—a credential—and produces a fixed-size output of enciphered text called a hash value, or just “hash”. 2

Kademlia Kademlia is based on performance and reliability. The network's structure is specified by how information is exchanged between nodes through lookups. Kademlia nodes communicate efficiently and effectively using UDP. A virtual or overlay network is a network that is formed by participant nodes. This type of network has many benefits, including the ability to provide better security and performance. A unique ID is assignate to each node. The node ID not only serves as identification, but also as a way to locate values using the Kademlia algorithm. The node ID provides a direct map to file hashes and stores information on where to obtain the file or resource, making it an elegant and persuasive solution.. 39

mutex A mutex is a program object that allows multiple threads to share a resource by taking turns accessing it. For example, a mutex can be used to control access to a shared file.. 29

nonce A nonce is a number that is generated for a specific use without any specific pattern (random or pseudorandom). This area of study is concerned with the secure exchange of information and the use of technology to support this process. A nonce is a number that is used only once, typically in cryptography.. 9

Bibliography

- [1] W. F. Silvano and R. Marcelino, “Iota tangle: A cryptocurrency to communicate internet-of-things data,” *Future Generation Computer Systems*, vol. 112, pp. 307–319, 2020, last accessed 2022-09-11.
- [2] C. Dwork and M. Naor, “Pricing via processing or combatting junk mail,” in *Advances in Cryptology — CRYPTO’ 92*, E. F. Brickell, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1993, pp. 139–147, last accessed 2022-09-11.
- [3] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” May 2009, last accessed 2022-09-11. [Online]. Available: <http://www.bitcoin.org/bitcoin.pdf>
- [4] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” last accessed 2022-09-11. [Online]. Available: <https://github.com/ethereum/yellowpaper>
- [5] P. Gaži, A. Kiayias, and D. Zindros, “Proof-of-stake sidechains,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 139–156, last accessed 2022-09-11.
- [6] C. Ye, G. Li, H. Cai, Y. Gu, and A. Fukuda, “Analysis of security in blockchain: Case study in 51%-attack detecting,” in *2018 5th International conference on dependable systems and their applications (DSA)*. IEEE, 2018, pp. 15–24, last accessed 2022-09-11.
- [7] T. Lisankie, @michaelmccallam, M. McCallam, @TomLisankie, Richie, @devtooligan, P. Wackerow, @wackerow, J. Vianello, @jhonyvianello, J. Ste, @julian st, K. Ziechmann, @kziechmann, A. Beregszaszi, @axic, A. Maiboroda, @gumb0, S. Richards, @samajammin, J. Cursi, @joncursi, MashhoodIjaz, @MashhoodIjaz, Griffin, and @gichiba, “Ethereum virtual machine (evm),” August 2022, last accessed 2022-09-11. [Online]. Available: <https://ethereum.org/en/developers/docs/evm/>
- [8] W. Entriken, D. Shirley, J. Evans, and N. Sachs, “”eip-721: Non-fungible token standard,” ethereum improvement proposals, no. 721,” January 2018. [Online]. Available: <https://eips.ethereum.org/EIPS/eip-721>

- [9] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, “Physical one-way functions,” *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002, last accessed 2022-09-11. [Online]. Available: <https://www.science.org/doi/abs/10.1126/science.1074376>
- [10] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, “Silicon physical random functions,” in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS ’02. New York, NY, USA: Association for Computing Machinery, 2002, p. 148–160, last accessed 2022-09-11. [Online]. Available: <https://doi.org/10.1145/586110.586132>
- [11] J. Arcenegui, R. Arjona, R. Román, and I. Baturone, “Secure combination of iot and blockchain by physically binding iot devices to smart non-fungible tokens using pufs,” *Sensors (Basel, Switzerland)*, vol. 21, April 2021, last accessed 2022-09-11.
- [12] “Ieee standard for local and metropolitan area networks—port-based network access control,” *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, pp. 1–289, 2020, last accessed 2022-09-11.
- [13] J. Benet, “IPFS - content addressed, versioned, P2P file system,” *CoRR*, vol. abs/1407.3561, 2014, last accessed 2022-09-11. [Online]. Available: <http://arxiv.org/abs/1407.3561>
- [14] Y. Psaras and D. Dias, “The interplanetary file system and the filecoin network,” in *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*. IEEE, 2020, pp. 80–80, last accessed 2022-09-11.
- [15] P. Labs, “Filecoin: A decentralized storage network,” July 2017, last accessed 2022-09-11. [Online]. Available: <https://filecoin.io/filecoin.pdf>
- [16] D. Johnson and A. Menezes, “The elliptic curve digital signature algorithm (ecdsa),” Tech. Rep., 1999, last accessed 2022-09-11.
- [17] G. Walker, “Difficulty: A mechanism for regulating the time it takes to mine a block.” March 2015, last accessed 2022-09-11. [Online]. Available: <https://learnmeabitcoin.com/beginners/difficulty>
- [18] —, “Target: The number you need to get below to mine a block.” March 2015, last accessed 2022-09-11. [Online]. Available: <https://learnmeabitcoin.com/technical/target#:~:text=The%20target%20is%20used%20in,every%2010%20minutes%20on%20average.>
- [19] W. Ethereum, “Ethereum whitepaper,” *Ethereum*. URL: <https://ethereum.org> [accessed 2020-07-07], 2014, last accessed 2022-09-11.
- [20] A. Antonopoulos, G. Wood, and G. Wood, *Mastering Ethereum: Building Smart Contracts and DApps*. O’Reilly Media, Incorporated, 2018, last accessed 2022-09-11. [Online]. Available: <https://books.google.es/books?id=SedSMQAACAAJ>

- [21] A. Beregszaszi, “Hardfork meta: Spurious dragon,” ethereum improvement proposals, no. 607,” April 2017, last accessed 2022-09-11. [Online]. Available: <https://eips.ethereum.org/EIPS/eip-607>
- [22] @pk910, last accessed 2022-09-11. [Online]. Available: <https://goerli-faucet.pk910.de/>
- [23] W. Bi, X. Jia, and M. Zheng, “A secure multiple elliptic curves digital signature algorithm for blockchain,” *arXiv preprint arXiv:1808.02988*, 2018, last accessed 2022-09-11.
- [24] N. Szabo, “Formalizing and securing relationships on public networks,” *First monday*, 1997, last accessed 2022-09-11.
- [25] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, “Finding the greedy, prodigal, and suicidal contracts at scale,” in *Proceedings of the 34th annual computer security applications conference*, 2018, pp. 653–663.
- [26] J. Khor, M. A. Masama, M. Sidorov, W. Leong, and J. Lim, “An improved gas efficient library for securing iot smart contracts against arithmetic vulnerabilities,” in *Proceedings of the 2020 9th International Conference on Software and Computer Applications*, 2020, pp. 326–330, last accessed 2022-09-11.
- [27] M. Rodler, W. Li, G. Karame, and L. Davi, “Sereum: Protecting existing smart contracts against re-entrancy attacks,” 12 2018.
- [28] S. M. Bartram and F. Fehle, “Competition without fungibility: Evidence from alternative market structures for derivatives,” *Journal of Banking & Finance*, vol. 31, no. 3, pp. 659–677, 2007, last accessed 2022-09-11. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378426606001518>
- [29] jamongeon1, S. Richards, @samajammin, B. Kashyap, @bskrksyp9, Joshua, @minimalism, MollyM, @momack2, N. Carbone, @drumnickydrum, P. Wackerow, @wackerow, zeroservices, @zeroservices, K. Leffew, @keleffew, A. Malik, @abdulmalik97, K. Ziechmann, @kziechmann, J. Chow, @kraxx, PatrickAlphaC, @PatrickAlphaC, katie, @okdonga, W. Entriken, and @fulldecent, “Decentralized storage,” last accessed 2022-09-11. [Online]. Available: <https://ethereum.org/en/developers/docs/storage/>
- [30] J. Benet, “Ipfs - content addressed, versioned, p2p file system,” 07 2014, last accessed 2022-09-11. [Online]. Available: <https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf>
- [31] S. Tajik, E. Dietz, S. Frohmann, J.-P. Seifert, D. Nedospasov, C. Helfmeier, C. Boit, and H. Dittrich, “Physical characterization of arbiter pufs,” in *Cryptographic Hardware and*

- Embedded Systems – CHES 2014*, L. Batina and M. Robshaw, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 493–509, last accessed 2022-09-11.
- [32] N. Wisiol, C. Gräbnitz, C. Mühl, B. Zengin, T. Soroceanu, N. Pirnay, and K. T. Mursi, “pypuf: Cryptanalysis of Physically Unclonable Functions,” 2021, last accessed 2022-09-11. [Online]. Available: <https://doi.org/10.5281/zenodo.3901410>
- [33] N. Wisiol and N. Pirnay, “Short paper: Xor arbiter pufs have systematic response bias,” in *Financial Cryptography and Data Security*, J. Bonneau and N. Heninger, Eds. Cham: Springer International Publishing, 2020, pp. 50–57.
- [34] P. Santikellur, A. Bhattacharyay, and R. S. Chakraborty, “Deep learning based model building attacks on arbiter puf compositions,” Cryptology ePrint Archive, Paper 2019/566, 2019, <https://eprint.iacr.org/2019/566>. [Online]. Available: <https://eprint.iacr.org/2019/566>
- [35] *ICCAD ’08: Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 2008.
- [36] M. Majzoobi, F. Koushanfar, and M. Potkonjak, “Lightweight secure pufs,” in *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD ’08. IEEE Press, 2008, p. 670–673, last accessed 2022-09-11.
- [37] F. Amsaad, M. Niamat, A. Dawoud, and S. Kose, “Reliable delay based algorithm to boost puf security against modeling attacks,” *Information*, vol. 9, p. 224, 09 2018, last accessed 2022-09-11.
- [38] M. Abomhara and G. Køien, “Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks,” *Journal of Cyber Security*, vol. 4, pp. 65–88, 05 2015.
- [39] R. Maes, A. V. Herrewewege, and I. Verbauwhede, “Pufky: A fully functional puf-based cryptographic key generator,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, pp. 302–319.
- [40] R. Maes, V. v. d. Leest, E. v. d. Sluis, and F. Willems, “Secure key generation from biased pufs,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 517–534.
- [41] E. B. INTEL, “Intel® secure device onboard,” *More secure, automated IoT device onboarding in seconds*, pp. 1–4.