POLITECNICO DI TORINO

Master of Science in Computer Engineering

Master Degree Thesis
in
Embedded Systems

Design of a Blockchain-enabled gateway for IoT

Supervisor
prof. Maurizio Rebaudengo

Co-Supervisor:
prof. Bartolomeo Montrucchio

Candidate
Jacopo Grecuccio

Company supervisor
FoodChain
dott. ing. Marco Vitale

25th October 2019

I hereby declare that the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

.....

Jacopo Grecuccio

Turin, 25 October 2019

Ai miei genitori

Abstract

During the last years, the interest around the Blockchain concept has grown faster and, as a consequence, several studies about the possibility of exploiting such technology in different application domains have been conducted. As a result, most of them highlighted the benefits that its use could bring in those context where the integrity and the authenticity of the data are important, e.g. for reasons linked to regulations about consumer's healthcare. In such cases it would be important to collect data in real-time through sensors and then store them in the blockchain, so that they can become immutable and tamper-proof. The thesis project aimed to the design and development of a software framework that allows Internet-of-Things (IoT) devices to interact directly with an Ethereum-based blockchain. The project has been carried forward in collaboration with the FoodChain S.P.A. company, which operates in the field of food-traceability through a blockchain-based mechanism. The company cooperates with the Quadrans Foundation which has powered its own Distributed Ledger Technology (DLT) based on the Ethereum Blockchain. The designed system has been integrated in a use case for temperature monitoring of fish products within a warehouse. Gathered monitoring data have been stored in a tamper-proof DLT. These data have been made accessible to the final consumer or to some certification authority, which can ensure that the temperature parameters imposed by the law have been satisfied. The obtained result demonstrates one of the many possible use of DLTs in the IoT domain.

Contents

Introduction

During the last ten years, the industrial processes commonly applied in any business field, from manufacturing to agriculture and many others, have been subject to a radical evolution lead by the arising of new technologies interconnecting the digital and the physical world and making up the so called Cyber-Phisical-Spaces (CPSs)[1]. This new industrial revolution has been basically built upon the concept of *Industry 4.0*, which was introduced for the first time in 2011 in Germany, with the presentation of the project "Industrie 4.0" with the goal of digitalizing the Hannover Messe manufacturing. The large interest demonstrated by the whole global industrial sector enabled the original concept to grow faster and to be applied and adapted not only to the manufacturing business, but to a large variety of applications including agriculture and breeding. The idea on which the Industry 4.0 paradigm is based consists in *making use of the IT technologies in order to interconnect different plants and systems together, to gather a great quantity of data from them and then to exploit the computational capabilities available today for analyzing these data and eventually take corrective actions in order to reach the desired process and product quality.* Moreover, the trend of embedding interconnected electronic devices capable of interacting with the environment in which they live in and enabled to communicate over the Internet, has spread not only in the industrial and manufacturing fields but it also influenced people's daily life, leading the IT market to develop this kind of products in other application fields going from healthcare and wellness to security and home appliances. The resulting scenario is known under the context of Internet-of-Things (IoT)[2], which can be defined as a digital overlay of information over the physical world. Each object (*"thing"*), connected in such network is uniquely identifiable and able to sense and react with the environment, as well as with other users or objects. [1]. The IoT devices market nowadays represents a

[1] A CPS consists of a system of collaborating computational elements controlling physical entities, in which mechanical and electronic systems are embedded and networked using software components. They use a shared knowledge and information of processes in order to cooperate for accomplishing a specific task.

[2] The term *Internet-of-Things* was coined in 1999 by Kevin Ashton

huge portion of the whole Information-Technology (IT) and electronic devices market with more than $235 billion spent in 2017, and it is expected to double in 2021, growing up to $520 billion [2].
However, this increasing and pervasive collection of data coming from people's private lives or business processes gives rise to important concerns about security and privacy. The most critical aspects are represented by the lack of a central control, the heterogeneity and the wideness of the possible attack surfaces, which come directly from the intrinsic features of the IoT [3]. Some examples of vulnerabilities in common off-the-shelf IoT devices, offering home control and automation features, have been analyzed in [4], which highlights how even simple attacks, like "man-in-the-middle" attacks, can represent a dangerous threat for people's privacy and safety.
At the state of art, most of the proposed solution addressing this problem relies mainly on centralized architectures or cloud services that allow internet-connected devices to communicate with them by means of digital certificates. This kind of architectures require from one side a *trust relationship* between the device's owner and the data-storage provider, and from the other a high maintenance cost because of its centralization. Moreover, these architecture can be potentially vulnerable to attacks on a single point-of-failure, which could lead to compromise one or more networks attached to it.
An alternative solution, aiming at overcome this and other issues, can consist in exploiting arising Distributed Ledger Technology (DLT) such as the Blockchain (BC).

A Blockchain (BC) is a distributed data structure that is replicated and shared among the members of a Peer-To-Peer network. Each entity in the network, known as *node*, interacts with the BC with a pair of private and public keys and, through an asymmetric cryptography authentication mechanism, can send its signed data performing *transactions*. These are broadcasted to other nodes that will have to *validate* them. The data storage consists in a linked *chain* of *blocks*, each of them contains the set of all transactions occurred since the previous block and the hash of the prior block itself, in addition because of the decentralized architecture, nodes have to commonly agree on the *state* of the chain using a distributed consensus mechanism. These basic principles upon which the technology is based reflects on the principal features offered by the Blockchain:

- **Immutability** : As each block contains the hash of its ancestor in the chain, and all peers have to agree on a single version of the latter, it is impossible, or at least extremely hard, for a malicious actor to halter a data previously stored in the network.
- **Authentication and non-repudiation** : Each node is uniquely identifiable through its public key, and from the other side when a node sends a transaction it *signs* the data using its private key, which is known only to him, and therefore cannot repudiate

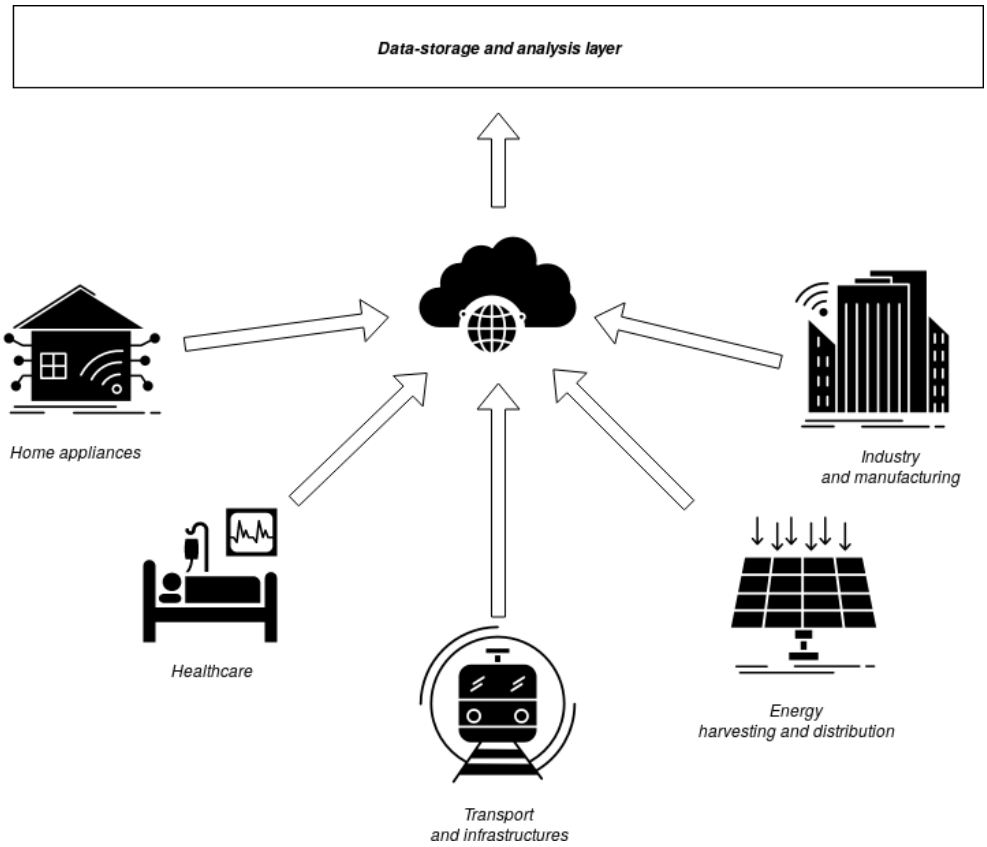


Figure 1: An example of the concept of Cyber-Physical-Space.(Image created using https://draw.io)

such data in future.

- **Privacy** : Nodes in the network share only their public keys, keeping their sensitive data private.

In addition, some implementations of this technology provide the possibility of writing automatic tasks, known as *Smart-Contracts*, that are automatically executed each time a transaction satisfies user-defined conditions.
The integration of the Internet-of-Things with the decentralized and trustless nature of the BC can solve the problems of centralization and security highlighted above and nevertheless enhance the actual IoT applications posing the basis for the implementation of machine-to-machine payments, Decentralized-Application (DApp) development, assets unique identification, ownership and data-authenticity assessment and more others.
The goal of the work presented in this thesis was to design and implement a gateway module to be used in IoT applications development allowing a single device, or an entire local network of devices, to directly and securely interact with an Ethereum-like blockchain. The project has been carried out in collaboration with the FoodChain SpA company and

the Quadrans Foundation.
Following in this document, will be first given the background concepts on the blockchain technology, starting from its history and fundamental concepts (Chapter 1) and then exploring with a good level of detail the architecture of the Ethereum framework and the cryptography on which it is based (Chapter 2). Then in Chapter 3 the designed system will be presented, and the proposed architecture will be analyzed in detail, giving explanations about the implementation choices with respect to the application requirements. Using the developed system, a proof-of-concept have been experimented on a real-life use case about the traceability process in the food chain. Thus in Chapter 4 the issues arising in this business sector will be described as well as the demonstration of how the integration between IoT and blockchain can help companies to overcome them.Finally the use case experimented in collaboration with FoodChain will be described and formalized through proper representations (UML diagrams) and the obtained outputs will be shown.

Chapter 1

The Blockchain technology

Since the early '80s idea of a decentralized digital currency has been considered by some people in the computer science community. Various anonymous electronic-cash protocols have been proposed during the 1980s and 1990s and they mostly relied on a cryptographic primitive known as blind signature[5] for providing a currency with a high degree of privacy. However these protocols didn't spread as expected because of their high dependency on a centralized intermediary.
In 1998 Wei Dai proposed the *b-money* protocol, which was the first one introducing the idea of creating money through the resolution of decentralized puzzles as well as decentralized consensus, but details about the implementation of the last aspect were scant.
Then, in 2005, Hal Finney introduced the concept of reusable proof of work, a system which uses some ideas from the b-money protocol together with Adam Back's computationally difficult hashcash puzzles to create a new concept for a cryptocurrency. Nevertheless the reliance of a trusted computing backend did not allow the system to be successful and widely accepted.
Three years later, in November 2008, a white-paper titled "Bitcoin: a Peer-to-Peer Electronic Cash System" was sent to the subscribers of a mailing list about cryptography by a mysterious author known under the pseudonym of Satoshi Nakamoto. Then in 2009 the first version of the Bitcoin client have been released by the same author. Today the real identity of Satoshi Nakamoto, who contributed to the project until 2011, is still unknown.

1.1 Bitcoin: the first Blockchain

The idea of Satoshi Nakamoto [6] combined established cryptographic primitives for managing ownership through public key cryptography with a consensus algorithm for keeping track of who owns the coin, known as "proof of work", and gave the birth to the blockchain technology.
In such system the electronic coin is defined as a chain of digital transactions and each owner of a coin can transfer it, performing a new transaction, to another participant in the network by digitally signing a hash of the previous transaction of the coin and the public key of the target new owner, adding such information at the end of the coin, thus building up a chain. Then the payee can check the signatures to verify the chain of the ownership. However, taking this idea as it is, it introduces the problem of "double-spending", which means that a payee can not verify if a coin has not been spent previously in an earlier transaction to another beneficiary. The possible solutions to this problem are two: the first one is to rely on a central trusted authority that verifies all transaction and prevents to spend the same coin twice, the second one instead requires each transaction to be publicly announced and a system of participants that can agree on a single history of the order in which they were received. Of course, as the goal of the technology was to create a decentralized system, the first alternative had to be discarded. The second solution instead represents the basic idea behind a Blockchain.

1.2 Basic concepts

The decentralized network proposed by Nakamoto is based on a Peer-To-Peer architecture, where all the nodes have the same functionalities and provide the same services without the distinction of roles as it happens in Client-Server architecture. Each peer in the network, known as **node**, stores locally a copy of all the history of the **transactions** published on the network.
A Blockchain can be formally defined as a structured database of **blocks**, each one containing several transactions, linked together by including in each block the cryptographic hash of the prior block in the chain. This allows to check the integrity of the whole chain going backwards until the genesis block, that is the very first block of the chain. Sometimes separate blocks can be produced concurrently leading to the creation of a temporary fork of the chain. In order to solve this problem any BC defines an algorithm for scoring different version of the history, so that only the one with the highest score is considered valid and selected over the others.

The mechanism used for verifying new transactions to be added to the chain is called **mining**, and not only allows all the users to agree on a unique version of the chain, but at the same time allows to create new coins that can be spent in the network, exploiting a **reward mechanism**.

1.3 Bitcoin as a state-transition-system

From a technical point of view, a cryptocurrency ledger, such as Bitcoin, can be described as a state-transition-system, where the "state" S consists in the ownership status of all existing coins and there is a "state-transition function" TX that takes as input a state and a transaction and outputs a new state S' as result. In particular, in Bitcoin, the state is the collection of all *Unspent Transaction Outputs* (**UTXOs**) that have been mined and not yet spent, with each of them having a denomination and an address which is essentially a cryptographic public key.
A transaction contains one or more inputs, each one containing a reference to an existing UTXO, a valid cryptographic signature, produced using the owner's private key, and a set of one or more outputs containing new UTXOs to be added to the state.
The state transition function *APPLY(TX,S) -> S'* can be then roughly described as follows:

1. For each input TX, check if the referenced UTXO is not in S and return an error if not. Then check if the provided signature for the UTXO matches the owner's one and return an error if not.

2. If the sum of all the denominations of all inputs UTXOs is less than the denominations of all outputs UTXOs, return an error.

3. Return S' with all input UTXOs removed and all outputs UTXOs added.

In practice, in the first step it is prevented to let the sender spend coins that do not exist, in the first half, and to spend coins not owned by him, in the second half. The second step instead, enforces the conservation of value. In other words, it allows to transfer non-integer values of coins: suppose that Alice wants to send 11.7 BTC to Bob. She will first find a set of available coins that she owns but such set would not be fractional realistically. Thus she has to perform a transaction having 12 BTC as input, which is the smallest value of coins she can get, and as outputs 11.7 BTC with Bob's public address and 0.3 BTC with its public address, representing a sort of "change".

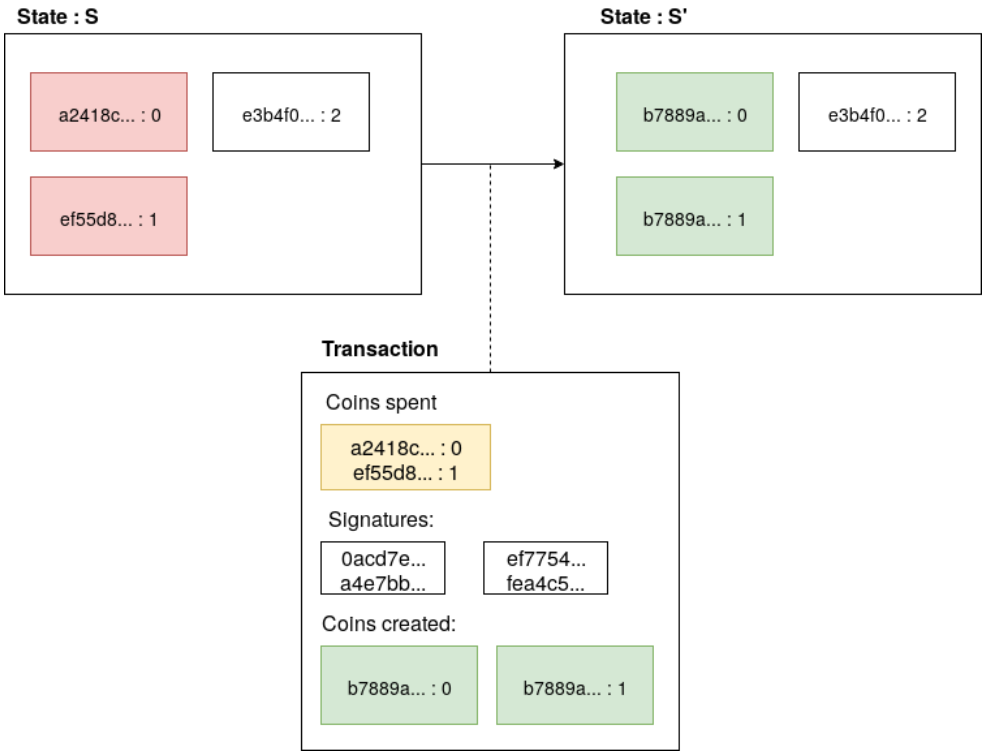


Figure 1.1: A representation of the state transition mechanism. In state S there are three coins respectively owned by a2418c... , e3b4f0... , ef55d8... . A a certain point a2418c... and ef55d8... decide to spend their coin, thus they provide their valid signatures for performing the required transactions. The miner b7889a... detects the transactions and resolves the PoW for adding a new block, containing these two transactions to the chain, and as reward he becomes the owner of the two coins created in the process. Thus, in the resulting state S' the coins owned by a2418c... and ef55d8... have been removed, while those earned by the miner have been added. (Image created using https://draw.io)

1.4 Mining

The state-transition-system described previously describes in technical terms the theoretical operations and primitives for managing a cryptocurrency ledger in general. If such ledger has to be maintained by a single centralized authority, the above description would be sufficient for achieving the required functionalities. However Bitcoin, and every BC system born right after it, are not intended to be centralized and for this reason they can be formally referenced as Distributed Ledger Technology. The lack of a central authority implies that all the network participants must agree on a single version of the chain through a consensus process, which is what has been previously defined as mining.

The mining process consists basically in solving the Proof-of-Work (PoW) algorithm for validating a set of new transactions, and thus leading to the creation of a new block in the chain. The PoW algorithm is selected to be difficult and computationally intensive to be solved, but at the same time easy in verifying the correctness of its solution. Due to

the costs, in terms of computational resources and power, required by the resolution of the Proof-of-Work each BC defines a reward mechanism for the miners.
In Bitcoin, for example, a predefined value of coins are earned each time a miner creates a new block. It is worth to notice that such coins were not existing in the previous state of the chain, but they were "created", or "mined", at the time when the block has been validated by the network for compensating the effort spent by the miner who created such block (*figure* 1.1). Obviously, this system has to be accordingly balanced in order to keep stable the value of the e-cash, and for these reason in Bitcoin the more the chain grows, meaning more blocks added, the more the PoW resolution becomes difficult and at the same time the reward associated to the creation of a new block decreases.
In particular, as the number of possible coins is finite (close to the maximum capacity of a 64-bit floating point number), Satoshi Nakamoto defined an exponential law (*formula* 1.1) for spreading the creation of all possible Bitcoins within a time window of 16 years. This law, known as *halving law*, requires the reward-per-block to half every 210000 blocks, corresponding to a time interval of 4-years, according to the first Satoshi's prediction.

$$\frac{\sum_{i=0}^{32} 210000 * \left\lfloor \frac{50*10^8}{2^i} \right\rfloor}{10^8} \tag{1.1}$$

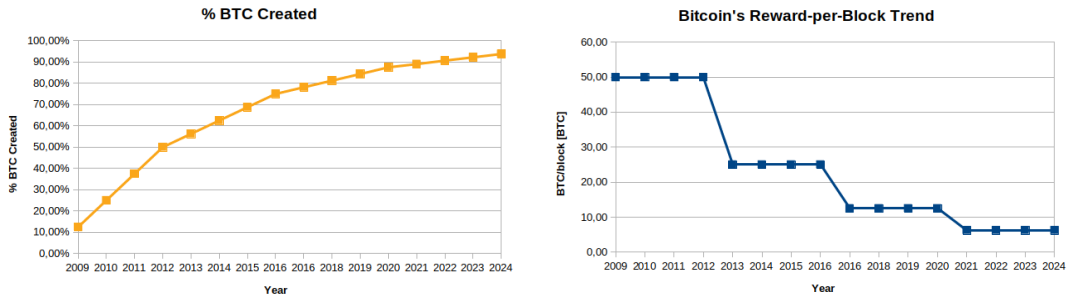


Figure 1.2: *Left*:Reward-per-block predicted trend from 2009 to 2024. *Right*: Percentage of existing BTC (mined) among the maximum possible number of them
Source: https://en.bitcoin.it/wiki/Controlled_supply

The limitation on the possible number of coins that can be mined poses obviously the problem of the maintenance of the system in a long-term perspective. Indeed, if a miner can not be rewarded for its contribution in new blocks creation, it would not be interested in investing its computational resources in such task. For these reason, along with the "new" mined coin, each miner is also rewarded with a certain fee associated to each transaction included in a valid block. In this way, miners will be interested in keeping their activity in the network even when there will be no more coins to be discovered.

1.5 Blocks and transactions

As it appears from the previous section, the fundamental elements upon which a Blockchain is built are blocks and transaction. The need for each node of the network of storing locally the entire chain implies the usage of an efficient data structure, that has to occupy a small quantity of memory and at the same time to guarantee that the data stored in the chain are immutable and tamper-proof.
In order to be scalable, in Bitcoin BC, each block's hash is the hash of its **block-header** (a data structure containing a timestamp, a nonce and the hash of the previous block) and the root hash of a multi-level data structure known as Merkle-Tree. A **Merkle-Tree** is a binary tree composed of a set of nodes with a large number of leaf nodes containing the underlying data at the bottom. Starting from the leaves each intermediate node stores the hash of its two children up to the root node, that is at the top of the tree.
The main purpose of this data structure is to allow to retrieve the required data efficiently and eventually piecemeal from different nodes in the network. Indeed, when a node wants to verify a transaction, it can download the associated block header from one source, then the small, relevant part of the tree from another one and at the end still be sure that the obtained data is correct. Moreover, the hash mechanism behind this structure ensures that the entire chain can not be changed or haltered.
Let's assume that a malicious node in the network wants to halter a transaction that has been previously verified and added to the chain. In this case the malicious node should modify the data-block associated to the transaction that is located at the bottom of the Merkle-Tree of the block in which the transaction is contained, but this modification will lead to modify the content of all nodes linked to the leaf up to the root node because of the hash propagation. On its turn, the change in the hash of the root node would reflect on the hash of the block and as a result a totally different chain, with respect to the "good-one", will be obtained which, most probably, will not be accepted by the other peers in the network.
However, nowadays the effectiveness of this protocol is arguable for long-term sustainability. Indeed, as of June 2019 the total space occupation of the Bitcoin Blockchain is roughly 226 GB and it is keeping growing faster, making unaffordable the implementation of a node on devices with limited storing and computing capabilities like mobile and embedded devices.

1.6 Bitcoin's scripting language

The Bitcoin implementation assumes that coins can be owned not just by entities associated to a public key, but also by a more sophisticated script expressed in a simple stack-based programming language. In this paradigm, a transaction spending such coins must provide a valid set of input data that satisfies the script. Indeed even the simplest mechanism, implementing the public key ownership, is represented by a script that requires as input a valid elliptic curve signature, verifies it against the transaction and the address that owns the coin and returns if the verification succeeded or not. Using this functionality more sophisticated scripts can be implemented for handling situations like multi-signature transactions for corporate accounts and so on. However, the Bitcoin scripting language has some important limitations that make it unfeasible for its use in the arising concept of DApp.

The first limitation stands in its lack of Touring-completeness, which means that there are some basic operations that can not be described, such as loops. The impossibility of writing loops in Bitcoin scripts comes mainly for avoiding infinite-loops that can indefinitely stall mining nodes. Theoretically this can be seen as a surmountable problem for programmers, as loops can be simply broken into a finite sequence of consecutive instructions.

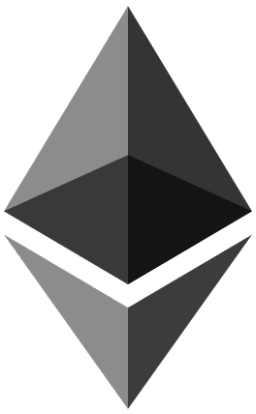
Another limitation of the Bitcoin scripting language stands in the lack of state for coins. A coin can only be spent or unspent, and there is no-opportunity to implement any multi-stage contract or script which can keep any other internal state beyond that.

In addition, the language does not allow to have any source of randomness that could come from Blockchain data such as nonce or timestamp, removing the possibility of developing gambling and several other categories of applications requiring such feature.

Overcoming such limitations is possible by following one of three approaches: building a new BC from scratch, using scripting on top of Bitcoin and building a meta-protocol on top of Bitcoin. Of course the first approach is the one that can give the highest flexibility at cost of development time while on the contrary, the second one would be easy and fast to implement but will give on its turn very limited capabilities. At the end, the third one will still be easy to develop but can suffer from faults in scalability. Several extensions or alternatives to the Bitcoin BC have been proposed and implemented using one of these three approaches during the past year, obtaining different results in terms of success, acceptance and diffusion in the Internet community.

Chapter 2

The Architecture of the Ethereum blockchain



After the advent of Bitcoin and the resulting increase of interest around cryptocurrencies, several new experiments have been developed using the Bitcoin's approach or totally different ones. Even if most of the times the term Blockchain is associated only to cryptocurrency and finance applications, some of these experiments aimed at breaking this link and exploiting the technology, which is just an implementation of a distributed ledger, for new types of applications.

Ethereum is then an Open-Source project that has to be intended in this direction, which has been developed aiming to the creation of a public BC for distributed computing.

It is born in 2013 thanks to Vitalik Buterin, a canadian developer and researcher in the field of cryptocurrencies. Through a crowdfunding campaign he was able, in 2014, to implement his idea which has then been published and available online the following year.

The intent of the project was to create an alternative protocol for building decentralized applications, that allows a different set of trade-offs to be used in those applications where rapid development time, security and the interaction between different distributed applications or systems is crucial. Following this idea Ethereum provides a fundamental layer composed by a Blockchain and a Turing-complete programming language so that anyone can write its own DApp, implementing its arbitrary rules for ownership, transaction formats and state-transition functions. The most interesting and explored functionality of this new architecture regards such applications known as Smart-Contracts. **Smart-Contracts**

can be seen as cryptographic entities containing a value, that can be represented by a certain amount of currency or information or both, unlocked only if an application-defined set of conditions are met. For example a Smart-Contract can be written for handling the transfer of goods and assets X between two actors A and B.
Let's assume that the information " *A owns the asset X*" has been previously registered on the ledger, and that at some point B requests to buy A's asset paying to the latter 3 ETH, divided in two fees of 1.5 ETH each and expiring respectively 30 days and 60 days after the day in which B accepts such proposal. If B does not accept the proposal no founds will be taken from its balance. Moreover the seller can also add further conditions to such deal:

- if 30 days after B's agreement, he has not sufficient founds for paying the first fee then the deal will be automatically canceled
- if 60 days after B's agreement, he has no sufficient founds for paying the second fee then the deal will be automatically canceled and the first fee will not be refunded, as penalty

At the end, 60 days after the contract's signature between A and B, if all the conditions were respected, A will receive the entire sum and B will be the new ownership of the asset. This situation is common in real life use cases as the sell of a car or a house.
Digitalizing this kind of processes using Ethereum is quite straightforward as it requires A,B or a third party different from them, to write a simple Smart-Contract implementing these conditions and consisting of just few lines of code. Moreover, the integration of a cryptocurrency with Ethereum's architecture allows to handle the payments without relying on a centralized authority like a bank, and not least, the involved transactions remains publicly accessible and transparent so that other parties can verify that the "rules" defined in the contract were respected .
It is also worth to notice that:

- the digital-signature mechanism on which the DLT is based ensures high levels of security and authenticity
- the Blockchain's architecture ensures the immutability and untamperability of the asset's change of ownership, so that once B becomes the owner of the asset there is no way to modify such information unless B decides to sell or donate the asset again, implying to sign and issue a transaction containing this information.
- as B is uniquely identified in the BC using its public address, that is in its turn in a one to one correspondence with the related private key, he can not repudiate his agreement to the contract because he should be the only one knowing the private key

used to sign such agreement, as it happens with traditional hand-drawn signature. It is also valid for A as well.

In order to achieve the goal of creating a simple framework for building powerful decentralized-application, the basic philosophy followed in Ethereum development is based on few principles:

- **Simplicity** : the protocol has to be as simple as possible, even at cost of some inefficiency in terms of execution time or data storage. The idea is that an average programmer should be able to follow and implement the entire specification.
- **Universality** : Ethereum should not provide features. Instead, it has to provide a complete fundamental layer upon which any kind of application can be easily built.
- **Modularity** : the framework has to be structured in modules, as separate as possible.
- **Agility** : protocol's architecture should not be extremely fixed, and any meaningful further proposal that can allow to bring significant benefits or to improve scalability and sustainability is accepted.
- **Non-discrimination** and **non-censorship** : the protocol should not attempt to actively restrict or prevent specific categories of usage.

Ethereum's fundamental currency is **Ether** (**ETH**), and is used for paying **gas**, which represents in its turn the unit of computation used in transaction and other state transitions, such as Smart-Contracts execution [7].

2.1 Fundamentals

2.1.1 State-transition system description

At this point, as done for Bitcoin, it is possible to describe Ethereum's Blockchain as a state transition system knowing that the states *S* and *S'* represent the state of all accounts respectively before and after the execution of the transaction *TX*, that is the state transition function. In particular the state transition function *APPLY(TX,S) -> S'* can be defined as follows:

1. Check if the transaction is well-formed, the signature is valid and the nonce matches the one of the sender's account. If not return an error.
2. Calculate the transaction fee as *STARTGAS*GASPRICE* and determine the sender address from the signature. Subtract the fee from the sender's account balance and increment the corresponding account nonce. If there is not enough balance to spend, return an error.

3. Initialize *GAS=STARTGAS* and take off a certain quantity of gas per byte to pay for the amount of data in the transaction.

4. Transfer the transaction value from the sender's account to the receiving account. If the receiving account does not yet exist, create it. If the receiving account is a contract, run the contract's code either to completion or until the execution runs out of gas.

5. If the value transfer failed because the sender did not have enough money, or the code execution ran out of gas, revert all state changes except the payment of the fees, and add the fees to the miner's account.

6. Otherwise, refund the fees for all remaining gas to the sender, and send the fees paid for gas consumed to the miner.

From the above description appears clearly the fundamental role of the "gas" in Ethereum's environment and the corresponding *start gas* and *gas price* parameters contained in each transaction object. Indeed, paying the "fuel" for code execution, data storage and bandwidth occupation prevents possible cases, accidental or hostile, of Denial-of-Service (DOS) that can be caused by infinite loops or computational wastage in the code. Usually each computational step costs 1 gas, but some operations which are more computationally expensive or increase the amount of data that has to be stored as part of the state cost higher amounts of gas. Moreover there is also a fee of 5 gas to pay per each byte contained in the transaction data field.
Ethereum's BC architecture is very similar to those of Bitcoin a part for some differences. First of all in this context block do not only contain the copy of the transaction list, as in Bitcoin, but they also store the most recent state. Aside from that, two other values are stored in each block: the *block number* and the *difficulty*. Basically the first one is a scalar number equal to the number of ancestor block, as the genesis block has block number equal to 0 by definition, the second one instead is a scalar number corresponding to the difficulty level of the block and which can be calculated from the previous block's difficulty level and the timestamp.This latter parameter is used by the PoW algorithm for increasing the difficulty of the puzzle as long as the chain grows.

2.1.2 Accounts

In Ethereum's BC, the fundamental elements composing the state are objects known as **accounts**, each one having a 20-byte **address**. Referring to the yellow paper [8], the **world-state** consists in the mapping between addresses and account's states, which are made of contains four fields:

- The **nonce**, an increasing counter used for ensuring that each transaction is processed once
- The account's current **balance**
- The account's **contract code**
- The account's **storage**

In general two types of accounts are specified: **externally-owned accounts** and **contract accounts**. The first type identifies those that are controlled by a private key and are able to send messages or transfer value to other accounts by creating and signing transactions, and having empty contract code field. On the other hand contract accounts are those storing some code, written in Ethereum Machine Language (EML), that is executed each time they receive a message. Such code can read or write the account's internal storage and send other messages or create other contracts in its turn.
Contracts can be seen as autonomous agents, living inside the Ethereum execution environment, and executing some piece of code each time they are queried by messages or transactions and having direct control over their balance and internal storage so as to keep track of persistent variables.
From this description it might seem that the Ethereum Blockchain's implementation is inefficient and not even less scalable, but it is not true in reality. Indeed storing the state in a tree-structure, whose root hash is stored in the blocks, implies that at each transaction only a small part of this tree has to be changed and so, in general, and so between two adjacent blocks the majority of the tree remains the same. Thus, instead of replicating the same data it can be imagined that it is only referenced using something similar to pointers. The Modified Merkle Patricia Tree (*trie*) data structure is used in Ethereum for exploiting this concept, and allows to obtain and efficiency comparable to that of Bitcoin.
Moreover, the trie mechanism not only applies to the whole world-state data structure, but it is also used in its internal components. Therefore instead of storing the entire account's storage or contract code as variable-sized byte array, they are encoded using Patricia Trees allowing to store only the root hash of such data structure and leading further optimizations in terms of memory savings.

2.1.3 Transactions and messages

The term **transaction** identifies signed data packages that store a message to be sent from an externally owned account. In particular a transaction object is made of six fields:

- the *recipient's address*
- the *sender's signature*

- the amount of *value* to transfer from the sender to the recipient
- an optional *data* field
- a *start gas* value, representing the maximum number of computational steps that the transaction's execution is allowed to take
- a *gas price* value, representing the fee that the sender is willing to pay per computational step

While the first three fields are quite standard in any cryptocurrency ledger, it is worth to focus on the last three.
The **data** field has no function by default, so as it can be used by any application running in Ethereum's environment for its specific purposes. Let's imagine a contract implementing an on-blockchain Domain Registration System, such as Namecoin[1]. In this context, if a user wants to buy a certain domain name, it has to tell to the contract managing the system the name to register and the IP address to register it to. Therefore, the user can sign and issue a transaction containing these values in the data field, as well as the amount to pay for the domain registration.
As seen before, contract accounts are also able to send **messages** to other contracts. These are virtual objects that are never stored in the chain as it happens for transactions, instead they exist only within the Ethereum Execution Environment and can be seen as a form of inter-process communication mechanism.
A message contains:

- The sender address
- The recipient address
- The amount of gas to transfer alongside the message
- An optional data field
- A *start gas* value

Essentially a message is very similar to a transaction, except for the fact that they are produced by contracts and not external actors, and as a transaction a message leads to the recipient account running its code. Note that when an external actor A sends a transaction

[1] **Namecoin** is born from the first proposal of inserting data into Bitcoin's Blockchain. It was basically implemented from a fork of Bitcoin itself and it was intended to be an alternative Domain-Name-Server (DNS) authority against censorship and opposite to the most common ICANN. Namecoin provides the registration domain names under the Top-Level-Domain .bit, as well as decentralized TLS solutions.

to a contract B, which at some point of its execution calls (sending a message) another contract C, then A has to pay the cost for the execution of the code of both contracts B and C. Thus, in such case C can be seen as it behaves like a subroutine of B.

2.1.4 Blocks and validation algorithm

A **block** in Ethereum is the collection of all the relevant information related with the architecture along with information of the comprised transaction in such block. In more detail, a data structure, known as **block header**, contains:

- The *parent block hash*.
- The *beneficiary address* to which all fees collected from the successful mining of the block are transferred to.
- The *state root hash*: the hash of the root node of the trie representing the new state after all transactions contained in the block are executed and finalized.
- The *transaction root hash*: the hash of the root node of the trie representing the list of transactions contained in the block.
- The *difficulty*: a scalar number corresponding to the difficulty level of the PoW algorithm for the block.
- The *block number*: a scalar representing the number of ancestors for the block, with the genesis block having value 0 for this field.
- The *timestamp*: representing the Unix's time() output[2] at the moment of the block creation.

In addition to such information, other additional ones are stored within a block in order to implement complex mechanisms such as events, logs and fast retrieve of a transaction execution outcome.
The basic block validation algorithm in Ethereum work as follows:

1. Check that the referenced previous block exists and is valid
2. Check that the timestamp of the block is greater than that of the referenced previous block and less than block time[3].

[2] A scalar number representing the number of seconds elapsed since the midnight of January 1^{st} 1970, known as *epoch date*

[3] The **block time** is a pre-defined parameter for the chain that defines the maximum time between the addition of two blocks in the chain. In Ethereum it is equal to 15 minutes

3. Check that the block number, the difficulty and the other parameters are valid
4. Check that the Proof-of-Work is valid
5. Let $S[0]$ be the state at the end of the previous block and TX the list of the n block's transaction. For $i : 0 : n - 1$ set $S[i + 1] = APPLY(S[i], TX[i])$. If any application returns an error or the total gas consumed exceeds the block's gas limit return an error.
6. If no error is returned in the previous state, let $S\_FINAL = S[n]$ be the new state.
7. Check if the tree root state of S_FINAL is equal to the state root provided by the block header. If it is the block is valid, otherwise it is invalid.

It is important to remark that the algorithm described above is not the PoW algorithm, that is resolved only once for each block by the block's miner. Instead it is the algorithm that each node in the network runs every time a new block is broadcasted and has to be validated before adding it to the "good" version of chain of each node.

2.1.5 Code execution environment

The code execution environment of Ethereum is based on a stack-based bytecode language known as **Ethereum Virtual Machine code**, and therefore the application that is in charge of running such code on the node is called **Ethereum Virtual Machine** (**EVM**). Such code consists of a set of bytes each one representing an operation. In general the code execution is an infinite loop which subsequently carry out the operation at the current program counter and then increments the latter by one until it reaches an error or a *STOP* or *RETURN* instruction. The operations can access three types of data structures in which to store information:

- The **stack**, a last-in-first-out data structure to which values can be pushed or popped
- **Memory**, an infinitely expandable byte array
- The contract's **long term storage**, a key/value store area. Unlike the previous two, this is a long-term storage area, and its content is preserved and does not reset after the computation ends.

In addition, the code can also access the value, sender's address and data of the incoming message, as well as block header data. Moreover the code can also return a byte array of

data as an output[4].
Formally, the execution model can be described with a tuple *(block_state,transaction,message,code,memor*
where the *block state* is the global state containing all accounts and including their balance and storage.
At each execution step the current instruction is extracted from the byte pointed by *pc* and each of such instructions affects the tuple in a different way. For example and *ADD* instruction pops two items of the stack and pushes their sum, then reduces *gas* by 1 and increments *pc* by 1. As it can be observed the logic behind the execution of Ethereum's code is very simple and somehow similar to the first generations of microprocessors architectures and their corresponding assembly languages, following the architecture of the Turing's Machine. Although there are many ways to optimize EVM execution via Just-In-Time Compilation (JIT Compilation)[5], its basic implementation of this mechanism can be done in few hundreds lines of code.

2.1.6 Ethereum's basic hash function and data serialization

As described in previous paragraphs, information related to blocks, state and transactions are compacted and encoded using hashes and the Patricia Tree structure. Hashes are computed applying the Keccak-256 hash[9] function over variable-size byte streams. These streams are in their turn a compact representation of nested arrays of binary data encoded using the Recursive-Prefix-Length (RLP) encoding algorithm, whose full specification can be found in Ethereum Wiki [10].

2.2 The digital signature process

Previously in this chapter the mechanism of transactions, upon which the blockchain concept is built, has been described. However for ensuring the properties of *authenticity* and *integrity* of the information, as in the intent of the technology, it is essential to find a way that ensures, deterministically and without any ambiguity, that a specific transaction has been produced only by a single, uniquely identifiable, entity in the network.
This is achieved thanks to the use of digital signatures schemes. A **digital signature**

[4]Smart-Contracts input and output data are encoded using a specific byte-stream oriented format, known as Application-Binary-Interface (ABI), which will be discussed further in this chapter

[5]**JIT Compilation** is a way of executing program's code that involves compilation during the execution of the program, at run-time, rather than prior to execution. It is a form of dynamic compilation and allows adaptive optimizations, leading in theory to a faster execution with respect to static compiled code

is a mathematical scheme, that makes use of asymmetric cryptography, for verifying the authenticity of digital information. It is typically composed by three algorithms:

- A *key generation* algorithm, in charge of selecting a **private key** among a set of uniformly random distributed possibilities. Along with the selected private key, the algorithm outputs also a **public key** that is directly related with the former.
- A *signing algorithm* that, given as inputs a private key and a message, produces a **signature**
- A *signature verifying* algorithm, that given as inputs a message, a signature and a public key, either accepts or rejects the message claim's authenticity.

Thus a digital signature scheme has to ensure that: first of all the authenticity of a signature generated from a fixed message and a fixed private key can be verified by using the corresponding public key, secondly it should be computationally infeasible to generate a valid signature for a party without knowing that party's private key.
The first concept of a digital signature scheme was described for the first time in 1976 by W. Diffie and M. Hellman, publishing a paper introducing a method for exchanging a shared secret between two parties over an insecure channel [11]. Then, soon afterwards, Ronald Rivest, Adi Shamir and Len Adleman invented the RSA algorithm [12].
Following in this section, the notions and the concept behind asymmetric key cryptography and Elliptic Curve Digital Signature scheme, used in Ethereum, will be given and then the transaction's signature method for Ethereum's blockchain will be outlined.

2.2.1 Background

Asymmetric key cryptography

Asymmetric key cryptography, also known as **public key cryptography**, is a cryptographic scheme that makes use of a pair of large numbers (*keys*) that are somehow related together but they are not identical. One key, referred as *private* because it has to be known only by the owner and not shared with any other parties, is used for decrypting a message received over an insecure channel. Instead, the other key of the pair, referred as *public* because the owner shares it other parties, is used by the message's sender for encrypting the message to be sent to the other party. (see the example in *figure 2.1*).
It appears clearly that such mechanism works under three fundamental assumptions:

- Private and public key should be related, but at the same time it should be unfeasible to obtain the private key knowing only the public one.

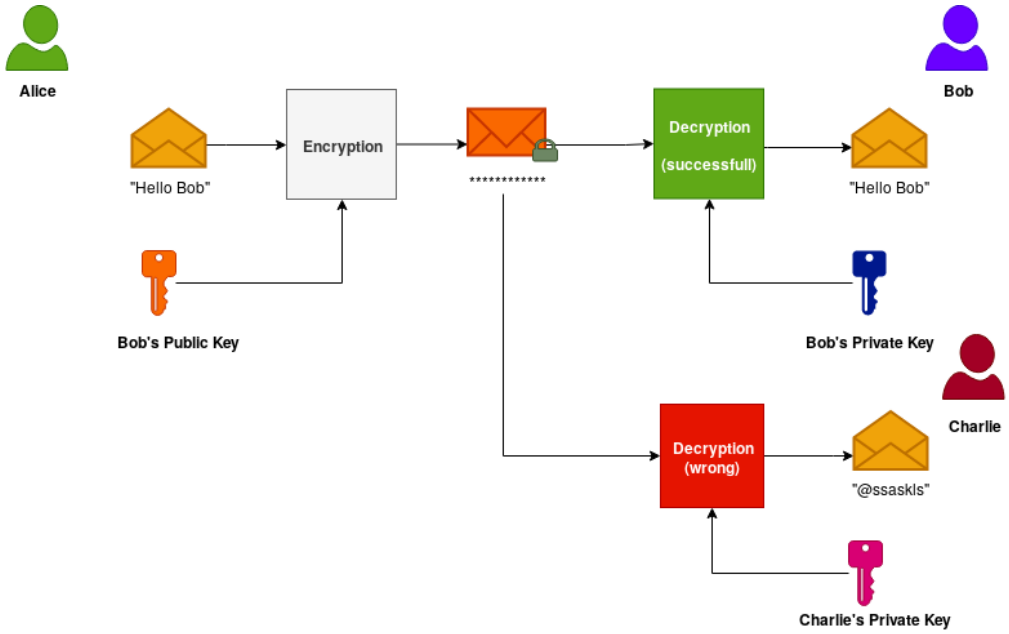


Figure 2.1: Example: suppose that Alice wants to send a secret message to Bob using an insecure connection. Let's assume that in an earlier step Alice and Bob had respectively exchanged their public keys without any encryption mechanism and that their private keys, instead, are not shared with anyone. As first step, Alice encrypts the message using Bob's public key and then she sends it over the channel. When Bob receives the encrypted message, he decrypts it using its private key, that is known only to himself, and when the decryption process has completed successfully he becomes able to read Alice's secret. Now imagine that Charlie, a curious person that is interested in Alice's secrets, is able to "sniff" the channel and intercept Alice's transmission. At this point the only way for Charlie for knowing Alice's secrets is knowing the message recipient's (Bob) private key, but it is not possible because of the assumption made at the beginning. Thus, Alice's secret is safe and known only by Bob and Alice herself. (Image created using https://draw.io)

- The decryption function gives the correct result if and only if the correct private key (i.e. the one related with the public key used for the encryption) is given and for no other different value.
- Private keys should be kept secret and not shared with any other parties.

Therefore asymmetric cryptography protocol implementations differs mostly in the algorithm used for the key pairs generation and the encryption/decryption functions.
For example RSA algorithm is based on the *large integer factorization problem* and, simply speaking, uses the product of two large prime numbers as part of the public key, and derives the private key from the same number as well. The encryption strength relies mostly on the key-size and its robustness increases exponentially as such parameter grows. Actually RSA keys can be typically 1024 or 2048 bits long, but experts believe that RSA1-024 is near to be cracked and it should not be considered secure anymore, especially with the advent of quantum computing.

Elliptic-Curve Cryptography

Another implementation of the public key cryptography's protocol relies on the algebraic structure of elliptic curves over finite fields and is known as Elliptic Curve Cryptography (ECC). It is based on the *elliptic curve discrete logarithm problem* that consists in finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point. The approach of using elliptic curves in cryptography was proposed for the first time in 1985 by Neal Kolbiz and Victor Miller, but it becomes widely used starting from 2004. It is used in blockchain implementations such as Bitcoin and Ethereum as Digital Signature scheme, and that is the reason why it has been decided to give a qualitative description of it in this document.
Before defining elliptic curves, it is worth to introduce the concept of finite fields. A **field** is a set of elements with operations defined over the elements of that set. These operations has to equate to something like addition, subtraction, multiplication and division and elements can in general be numbers or any other entities. Moreover a field is defined if and only if the following conditions are met:

- Both multiplication and addition operations are closed over the set, meaning that if a and b belongs to the set also $a + b$ and $a * b$ belongs to it.
- Both multiplication and addition satisfies associative and commutative properties.
- Both multiplication and addition must have identity elements. For example $a+0 = a$ and $a * 1 = a$
- There must be additive and multiplicative inverse for all elements in the set.
- Multiplication must distribute over addition, meaning that if a,b and c are in the set then $a * (b + c) = a * b + a * c$

So, a **finite field** is defined as a field with a finite number of elements in the set. At this point it can be defined an **Elliptic Curve** as a plane curve over a finite field consisting in the points satisfying the equation:

$$y^2 = x^3 + ax + b$$

and in addition in ECC exists a further restriction imposing that x,y,a and b belongs to the field as well. Along with the point on the curve, an additional point known as **point at infinity** O is defined and used as identity element. On the obtained set (the curve points plus the point at infinite) two operation can be defined:

- **Point Addition**: it takes two given point on the curve and yields a third point on the curve as output. The result of the addition $P + Q$ between two points P,Q lying

on an elliptic curve, is a third point R that is the negate of the point $-R$ at which the straight line passing in both P and Q intersects the curve.

- **Point Doubling**: it takes one point on the curve and yields another point of the curve as output. The operation can be seen as an addition operation for which $P = Q$, thus its result is the point R that is the negate of the point $-R$ at which the tangent of the curve at point P intersects the curve again.
- **Point Multiplication** : it takes as input one point on the curve P and a scalar number k and yields as output the point Q. It can be obtained combining both Point addition and doubling operation, so for example the operation $Q = 5 * P$ can be seen as $Q = 2 * P + 2 * P + P$.

Now, exploiting the operations introduced so far it is possible to explain the basics behind Elliptic Curve Cryptography. Imagine to take a point G on the curve, that can be defined as **generator**, and to multiply this point for by an increasing scalar k starting from $k = 0$ and continuing until the value n at which the operation $n * G$ gives as result the point at infinity O. The obtained set of al scalar product of the point G forms what is defined as a **subgroup** of the **group** formed by elliptic curve's point. The value of n is said to be the **order** of the group, and as stated by the Lagrange's Theorem, the total number of points on the curve N (i.e. number of points in the group) is divisible by n (i.e. number of points in the subgroup), and such division produces as a result a value h known as **cofactor**. Summarizing, for Elliptic Curve Cryptography's algorithms the following parameters are needed:

- The parameters of the elliptic curve a,b
- The size of the finite field p (usually selected in the set of large prime numbers)
- The base point G that generates the subgroup
- The order of the subgroup n
- The cofactor h (knowing this value in advance helps to speed up algorithms)

The security of ECC stands in the fact that, roughly speaking, given a point $k * G$ on the curve it is difficult to go back to the original value of k, this is known as the **Discrete Logarithm Problem**[6].

[6]The Discrete Logarithm Problem is said to be one of the NP-hard problems, and consists in finding the scalar multiplier of a point multiplication $Q = k * P$ knowing only the points Q and P

Key pairs in ECC

From what it has been presented so far about asymmetric cryptography and ECC it is possible to easily deduce how the mathematics behind elliptic curves over finite fields comes in help for finding key pairs satisfying the two crucial assumption behind asymmetric cryptography:

1. Public and Private key must be related together but not similar
2. It should be impossible to go back to the Private key knowing only the Public key, but the inverse should be easy

Let's consider a subgroup G of order n over an elliptic curve. It comes out that there will be $n-1$ valid points (different from the point at infinity) of the curve in the subgroup and that can be obtained as the result of $d * G$ with d in the range $[1, n-1]$. Now, assume to choose randomly the value of d in such range and to generate the corresponding point $P = d * G$. It can be noticed that:

- P and d are somehow related, the former is the point generated multiplying the generator point by the latter. Moreover they are different in their nature as one is an elliptic curve's point and the other is simply a scalar
- Because of the difficulty of the Discrete Logarithm Problem it is computationally unfeasible to find out d knowing only P and G.

At this point becomes evident that if the value of d is chosen to be a private key and the one of P the public one, a good and secure key pair for a public key cryptography protocol has been obtained.

Elliptic-Curve Digital Signature Algorithm

Elliptic Curve Cryptography's concepts can be used for implementing a Digital Signature scheme called Elliptic Curve Digital Signature Algorithm (ECDSA). Assume that Alice owns a key pair where d_A is hers private key and H_A is hers public one. She wants to sign a digital document and anyone should be able to check that the document has been signed by Alice and no one else, but at the same time Alice has to be the only one able to produce valid signatures for herself. Assume also that each one able to verify Alice's signature shares the same domain parameters with Alice.
As ECDSA works on the hash of a message e rather than the message m itself, the first operation to be done is to apply a cryptographically-secure hashing function to the message, and eventually truncate it to the same bit-length of the group order n, obtaining z. At this point the Elliptic Curve Digital Signature Algorithm performed by Alice works as follows:

1. Chose a random integer k among the range $[1, n-1]$
2. Calculate the point $P = k * G$
3. Compute the scalar $r = x_P \mod n$ (where x_P is the coordinate of P on the x-axis)
4. If $r = 0$ select another k and try again
5. Compute $s = k^{-1} * (z + r * d_A) \mod n$ (where k^{-1} is the multiplicative inverse of k modulo n)
6. If $s = 0$ then select another k and try again, otherwise the pair of scalars (r, s) is represents Alice's digital signature for the message m

Roughly speaking, the algorithm first generates a "secret" k and encodes it in the value of r. The secret is hard to be discovered because of the complexity of the DL Problem analyzed above. Then the value of r is bounded to both the message and Alice's private key using the equation in point 5, generating the value s. It is worth to underline that the security and robustness of the algorithm lies in the selection of k, that should be truly random otherwise it could lead to possible break down of the algorithm, as it happened for the Sony Playstation 3[7].Compared to RSA, ECDSA offers the same level of security but with smaller keys.
In the following paragraphs the signature's verification process will be outlined.

Elliptic-Curve Signature Verification Algorithm

Now, let's assume that Bob wants to verify that the document received was really signed by Alice, knowing only Alice's public key H_A and the pair (r, s) that is the signature to be verified. First of all he has to hash and eventually truncate the message m again as done by Alice during the signature process in order to obtain z. At this point Bob runs the following algorithm:

1. Compute $u_1 = s^{-1} * z \mod n$
2. Compute $u_2 = s^{-1} * r \mod n$
3. Calculate the point $P = u_1 * G + u_2 * H_A$

[7]In 2010 an hacker known as fail0verflow exploits a poor implementation of the ECDSA in the Sony PS3 console for generating malicious valid signature and having access to high-level protected operations. The user discovered a bug exploited by the user was a serious mistake in the generation of k in the digital signature algorithm. Indeed instead of using a randomized value of k for every different signature, it was not randomized at all and fixed to a single value, that the hacker was able to detect quite easily

The signature is then considered valid if and only if $r = x_p \mod n$. At first glance the algorithm could seem a bit strange and unrelated with the signature's one, but with a little bit of substitutions things will appear more clear. Let's start from the equation at point 3

$$P = u_1 * G + u_2 * H_A$$

using the definition of $H_A = d_A * G$ substitute it in the formula, obtaining

$$P = u_1 * G + u_2 * (d_A * G) = (u_1 + u_2 * d_A) * G$$

At this point using the definition of u_1 and u_2 given respectively at points 1 and 2 it is possible to rewrite the equation as

$$P = (s^{-1} * z + s^{-1} * r * d_A) * G = s^{-1} * (z + r * d_A) * G$$

Finally taking the definition of $s = k^{-1} * (z + r * d_A)$ and multiplying both side of it by $s^{-1} * k$ in order to obtain $k = s^{-1} * (z + r * d_A)$ and then substituting it in the above equation, it comes up $P = k * G$, that is exactly the first step of the signature algorithm. In other words it has been possible to compute the same point P used for the signature with a different set of equations that do not require the knowledge of the private key.

2.2.2 Digital signatures in Ethereum

As mentioned before, some blockchain implementations, like Ethereum's one, use ECDSA as digital signature scheme.
In particular Ethereum uses the digital signature for guaranteeing the authenticity of each transaction, that represent therefore the message m to be signed. The domain parameters are known under the acronym of SECP256K1.
The transaction signing process in Ethereum work as follows:

1. The sender of the transaction fills properly the fields described above (*recipeint's address,amount of WEI to send, gas limit, gas price, data*).
2. The resulting data structure is then serialized using RLP encoding.
3. The obtained serialized byte-stream is the hashed applying the Keccak-256 hash function.
4. The Elliptic Curve Digital Signature Algorithm is executed over the obtained hash and it outputs as result the value of r,s and another value v, known as the recovery ID.

5. The transaction's data structure is updated with the values representing the signature and the recovery and then it is serialized again.

6. Finally, the serialized byte-stream, known as *rawTransaction*, is broadcoasted to the other peers in the network

Differently from the common ECDSA implementations, Ethereum's one normalizes the the possible values s in the interval $[0, \frac{n}{2} + 1]$ and the exploits the value of v to retrieve the original value of s. The value of v, in the original Ethereum's implementation, is therefore used for encoding the parity of r_y (the y-axis component of r coordinate) and the half of the plane in which the s coordinate lies, and is computed as follows:

```
rec_id = 0;
if( parity(r.y) is odd) {
        rec_id = 1;
}
if(s > (n/2+1)) {
        s = -s;
        rec_id = rec_id xor 1
}

v= 27 + rec_id
```

However, sometime after the first Ethereum's blockchain release the computation of the value of v has been modified in order to protect private and public Ethereum's network against replay attacks [8]. Shortly, in order to solve this problem the modification introduced, known as EIP-155, requires to include the *chain-ID* value in both the serialized byte-stream to be signed and the value of v, which therefore becomes $v = CHAIN\_ID*2+35+rec\_id$ [13].

2.3 Smart-Contracts and their business applications

The term Smart-Contract (SC) was introduced for the first time in 1990s by Nick Szabo, a computer scientist and cryptographer who realized that the Distributed Ledger Technology can be exploited for securing relationships over a network. As explained in the previous chapter, Bitcoin was the first blockchain implementation to support a basic form of Smart

[8] In general a **replay attack** is a valid data transmission that is maliciously repeated or delayed. Extending this concept to blockchains, a replay attack consists in taking a transaction on one blockchain, and maliciously repeating it on another blockchain. For example, an attacker taking someone's testnet transaction, and repeating it on the "real" blockchain, to steal "real" funds.

Contracts through its scripting language, which was however quietly limited. Ethereum project, on the other hand, was born with Smart-Contracts in mind and aiming at realizing a framework for DApps running over a blockchain ledger in a peer-to-peer network.
Just as the term suggests, Smart-Contracts are somehow related to "normal" contracts, with the difference that if the latter ones are signed by partaking partied and enforced by the law, the formers are instead digitally signed and set out parties relationships exploiting cryptographic mechanisms.
Basically a Smart-Contract is a self-executing application that runs on top the blockchain and is therefore immutable. It can be seen as a complex *if-then* statement which is executed if and only if a set of condition is met. Smart-Contract can either produce an output or even trigger the execution of another Smart-Contract for performing a sophisticated task. Because of their decentralized nature and the possibility to exploit blockchain's features for handling trust-less contexts, SC are becoming an attractive topic for various kind of business fields spreading from finance, insurance, manufacturing and many others.
In order to better understand how SCs work and how big can be their potential if exploited in some applications domain, is worth to describe an example use case (*figure 2.2*). Let's assume that a farmer A sells tomatoes to a company B which then uses them for producing tomato's sauce. The transfer of from A's farm to B's warehouse is in charge of a third delivery company C. The actual most common scenario would be the one in which A,B and C have their own digital business layer generally made of a centralized data-storage (in a local Database or using some cloud services) and their own software products that deal with such data structure for registering shipping, invoices, inventory and so on. In addition these software may differ between one company from another making difficult to automatize their co-operation. Let's assume now that the same business scenario should be managed through the use of Smart-Contracts and a blockchain in a Peer-To-Peer network at which all of the three companies participate. A smart DApp developer would implement three Smart-Contracts able to interact together:

- A first Smart-Contract that can handle the transfer of ownership of the batch of tomatoes from A to C, ensuring that the latter would be responsible for any loss or damage during shipping.
- Another Smart-Contract that can handle the transfer of ownership from the delivery company to the recipient B.
- A third Smart-Contract, triggered by the previous one, that handles payments between the parties in something similar to the following statement: *If tomatoes are delivered without any damage a certain amount of funds will be transferred to both the selling and shipping companies. If tomatoes are damaged then the company C*

have to refund the selling company, and so the proper amount of crypto-money will be transferred from C's account to A's one.

Thanks to the blockchain all operations occurred between the parties will be registered in the ledger forever, and so it is straightforward to think to more complex mechanisms to be build upon the above simple scenario (insurance for the delivery process, supply-chain traceability,etc.).
In conclusion during the last few years Smart-Contracts are growing in their popularity allowing to move the blockchain technology away from the bare crypto-currency fields to a large variety of applications.

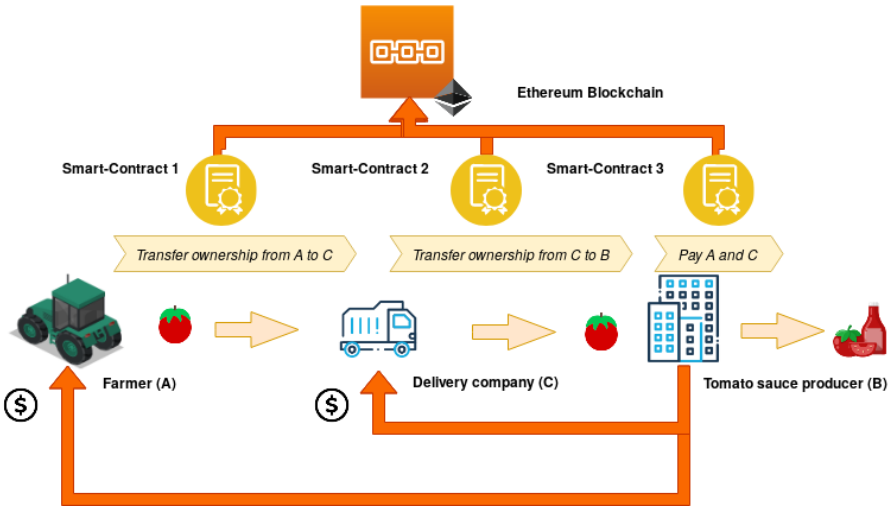


Figure 2.2: A picture illustrating the scenario presented in the example above. (Image created using https://draw.io)

Chapter 3

A gateway toward the blockchain for IoT applications

Now that the basics principles of the blockchain's technology have been described with a good level of detail, it is possible to introduce the real goal of the thesis project. As stated in the introduction, the project aims at finding a new method of interaction between the blockchain infrastructure and the world of internet-connected devices. Actually the most common solution applied for this scenario, consists in a Client-Server communication, between a central server and the IoT devices, and then the server, in charge of gathering the collected data streams from the field, stores these data, or part of them, in the blockchain. Even if this solution is starting to be applied in different proof-of-concepts and business applications, this has some point of weakness:

- The central server may become a single point-of-failure that can inhibit some data to be stored in the blockchain when it is in a fault condition.
- Assuming that redundancy is applied, so that to minimize the probability of failure, still there is the presence of some kind of centralization that makes difficult the interaction between co-operative business parties.
- The data which is then sent in the blockchain is not signed-in-place, in such a way that its authenticity and integrity can be guaranteed starting from the source, but it is signed only when it is received by the central server before posting it into the blockchain.
- Nevertheless such kind of systems are now mostly realized using cloud solution for the IoT field, like those provided by IBM (IBM Watson) or Amazon (Amazon AWS).

Experts believe that the cost for such services is going to increase rapidly and to become a significant component of business cost for most of companies.

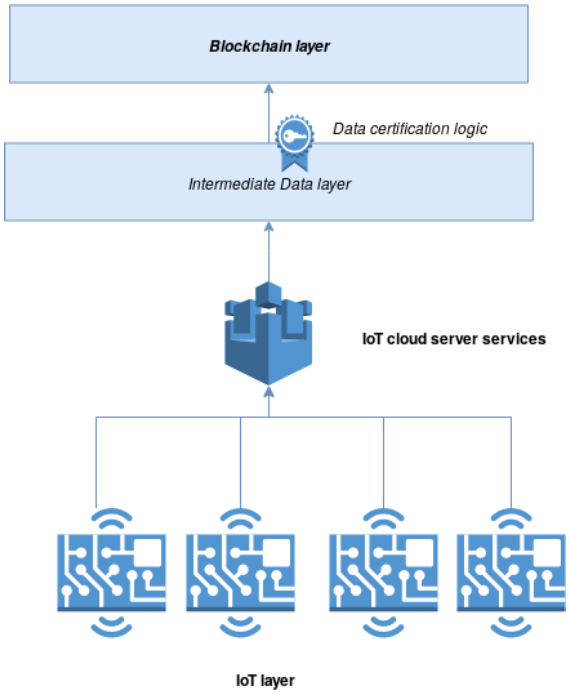


Figure 3.1: The common architecture used for storing data collected from IoT devices into the blockchain.(Image created using https://draw.io)

3.1 Proposed solution

The solution proposed in this chapter is intended for solving the weakness presented above, and possibly to introduce some improvement to the described context. It basically consists in enabling commercial or ad-hoc Internet-of-Things devices to communicate directly with the blockchain infrastructure through a *gateway* device to which they are connected using wired or wireless protocols. Such gateway device should be able to sign in place the data sent from the IoT data-logging device and then to directly push the BC. The most straightforward method of doing so would be to let the gateway be a full-node of the chain (i.e. a node in the peer-to-peer network that is able to store and verify the whole chain and keeping it updated too). However, this is unfeasible as the target application context lies in the embedded-systems domain, which commonly consists of device with limited computing capability and for sure not enough storage that allows one of them to maintain an entire copy of the chain.
A first intermediate solution (*figure 3.2*) that has been explored is based on an architecture

in which the IoT layer communicates directly with an Ethereum full-node placed within the same Local-Area-Network in which the devices are connected. Such local node is then in charge of signing and broadcasting the transactions requested by the devices, as well as keeping their private key storage and mappings (between IoT device id and private key). The implementation of this solution resulted to be fast and straight forward, because the only part on which one has to deal with requires only the communication mechanism between the local BC node and the IoT devices, defining the data-model of the message to be exchanged and eventually some mechanism of encryption for securing the communication channel. However this solution suffers of the same problems related to centralization and security as the common one presented above. Nevertheless it would only be worse with respect to that if no mechanism for device authentication is implemented for the communication with the local full node. Therefore the path followed in the project exploits the

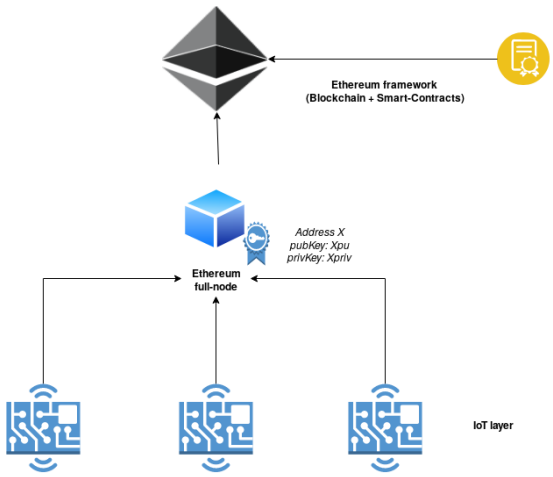


Figure 3.2: A representation of first explored solution. IoT devices communicates with the local full node which is in charge of signing and broadcasting transactions for them. However the IoT layer remains agnostic of the existence of the blockchain. It simply sends data to a local centralized entity.(Image created using `https://draw.io`)

Remote-Procedure-Call (RPC) protocol for enabling an embedded device (acting as gateway) to trigger the execution of some procedure on a remote server exposing such service. Such device has to be physically placed in the very near proximity of the IoT device (eg. as expansion hardware for the device) or even be the IoT device itself. The RPC server exposes to calling clients (i.e. IoT gateways) the Web3 API, that allows to interact with Ethereum's blockchain, so that it can be achieved that any embedded device can access to the information stored in the chain, exchange coins (eg. for implementing machine-to-machine payments) and call the execution of Smart-Contracts for implementing complex

logics on top of the blockchain layer. Moreover, exploiting the events mechanism of Smart-Contract[1], and the possibility to search for those event in the chain by selectively download block headers through RPC calls, it could even be possible to trigger IoT devices directly within distributed application, and keep the history of such request. In order to apply this solution however there should be one strict requirement above the others: *the gateway must be identifiable on the networks through its address, and should also be able to sign the transaction locally and off-line, using its own private key, before sending it to the RPC server*. This functionality is essential in order to avoid the possibility that the transaction is maliciously manipulated when it is sent to the server.
The final overall proposed architecture is shown in *figure 3.3*.

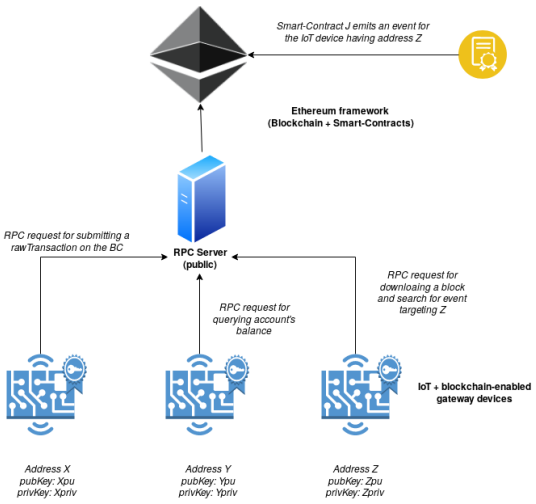


Figure 3.3: A representation of the final proposed architecture. Each IoT devices has its own gateway and is able to sign transaction locally and offline. Each of them is also identified within the blockchain through its address and can be thus a target for possible Smart-Contract events. (Image created using `https://draw.io`)

3.1.1 Target Hardware

The target hardware used for the project consists of a custom board (shown in *figure*) on which are mounted:

- A PIC32MX 32-bit microcontroller
- A 256-byte EEPROM (used for storing keys)

[1]Events and logs are important in Ethereum because they facilitate communication between smart contracts and their user interfaces. In traditional web development, a server response is provided in a callback to the frontend. In Ethereum, when a transaction is mined, smart contracts can emit events and write logs to the blockchain that the frontend can then process [14].

- A Flash memory
- An SD Card reader
- A serial-input-parallel output register (74HC595) and a parallel-input-serial-output register (74HC165) for extending number of available GPIO pins.
- Various wired-serial-communication interfaces (SPI,I2C,UART)
- An Ethernet controller interface.

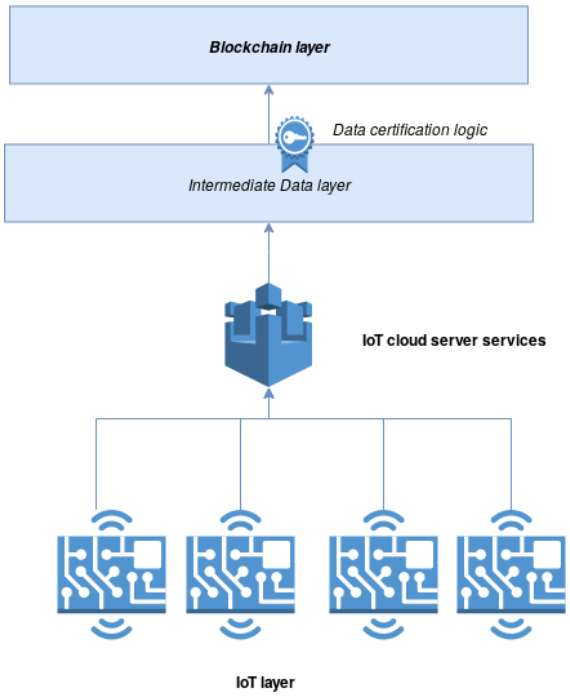


Figure 3.4: The common architecture used for storing data collected from IoT devices into the blockchain.

Most of the peripherals mounted around the MCU allow the software running on the latter and implementing the Blockchain-gateway to communicate with other "daughter" boards of any kind, that can be programmed for performing data-logging tasks (i.e. measurements of physical quantities through specific sensors) or control and automation tasks (i.e. acting on the environment or on the system under control for making it perform a certain action using actuators).

3.2 Software architecture

Most of the complexity for the project has been demanded to the software layer, meaning that the vast of majority of the operations are performed by the CPU without any additional special-purpose peripheral or hardware accelerator.
Thus, the greater part of the work carried out during the project has been the development of a framework, intended to be as easy and user-friendly as possible in order to be used by the most of IoT developers that have a minimal knowledge of the details behind the blockchain architecture. Following this philosophy the resulting developed framework can be represented as a layered architecture (*figure 3.5*) where the top-most software components are those providing the interaction services to the common Internet-of-Things application.
Moreover, referring to the guidelines provided by the Microchip Development Guide all tasks implementing complex functions, or functions that require the interaction with other sub-tasks, have been implemented using the State-Machine model[2].

3.2.1 The MPLAB Harmony Framework

MPLAB® Harmony is a fully integrated embedded software development framework that provides flexible and interoperable software modules that allow you to dedicate your resources to creating applications for our 32-bit PIC® and SAM devices, rather than dealing with device details, complex protocols and library integration challenges. It works seamlessly with MPLAB X Integrated Development Environment (IDE) and the MPLAB XC32 Compiler to enable a smooth transition and maximum code reuse between PIC32 MCUs and SAM MCUs and MPUs [15].
During the project, most of the drivers and software libraries provided by this framework have been used in order to deal with the MCU peripherals using tested and guaranteed software module, as well as to reduce development time. However, for what concerns the Cryptographic module, the implementation of the ECDSA algorithm have been slightly modified with respect to the one provided by the framework, for adapting the algorithm to the ECDSA variant used in Ethereum (which requires the computation of the recovery id and the "normalization" of the range, as described in the previous chapter).

[2]State machines are used to model real-world software when the identified state must be documented along with how it transitions from one state to another.

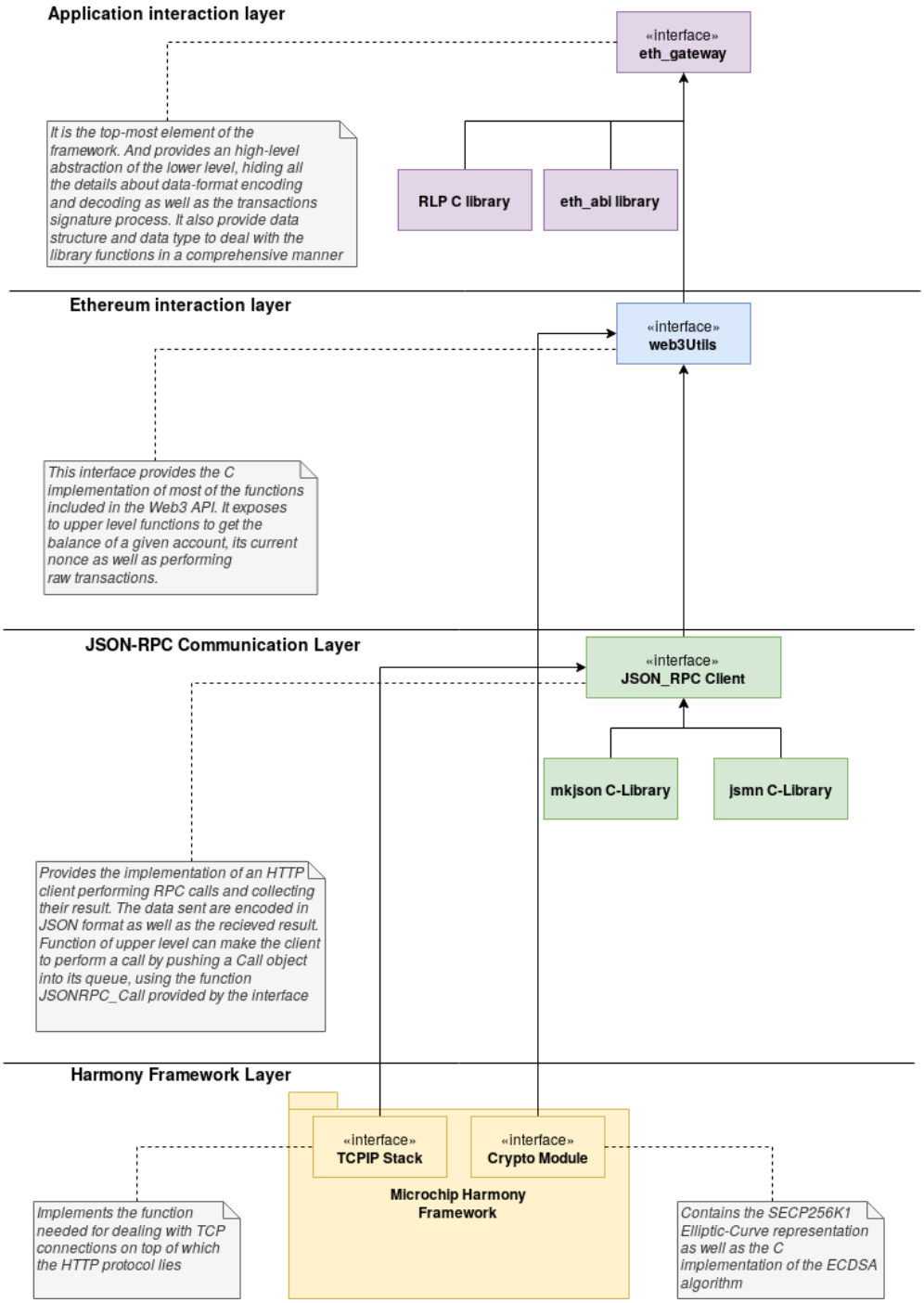


Figure 3.5: The UML Diagram representing the developed framework's architecture. (Image created using https://draw.io)

3.2.2 The JSON-RPC layer

The Remote-Procedure-Call is based on the Client-Server model and is used in those contexts where one program wants to request a service from another one, executed in a remote host. An RPC call is a synchronous event requiring the caller program to be suspended until the remote procedure returns its result. JSON-RPC represents a state-less and light-

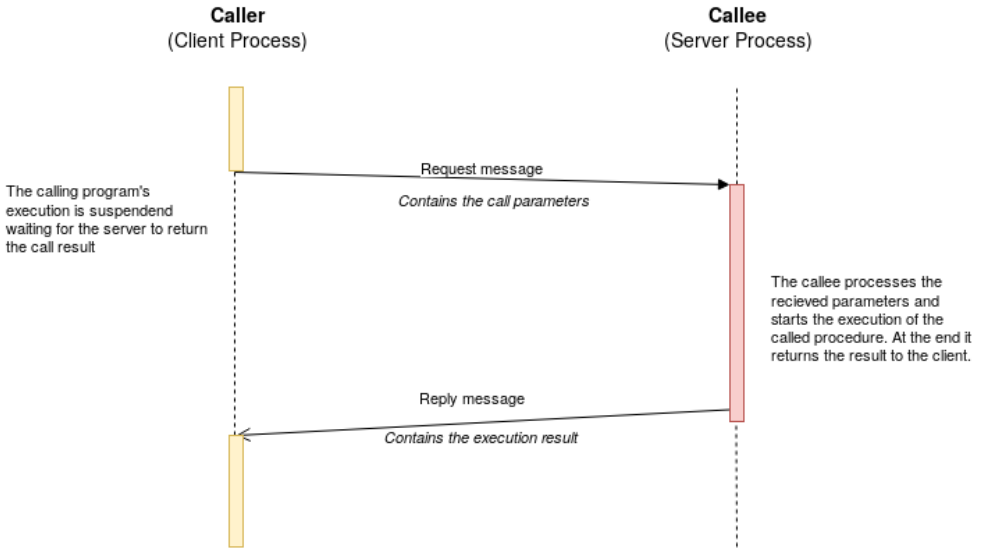


Figure 3.6: A sequence diagram showing the flow of execution of a remote-procedure-call. (Image created using https://draw.io)

weight implementation of a RPC. Its specification defines several data-structure and the rules around their processing and uses the JSON[3] data format [16].
In the developed framework JSON-RPC Client is the software component in charge of performing JSONRPC calls over the TCP socket on which the RPC server is listening. Its implementation basically consists of a queue, where other tasks push a data-structure for each RCP-Call that they request. This data-structure is used to model the standard fields of a Remote-Procedure-Call and in addition to those fields another one is provided in order to store the current state of the call. Each call, in this implementation, can assume the following logical states:

- `QUEUED` : it is the first state assumed by a call-object. It describes a call that has been pushed into the queue by a caller task and it is waiting to be executed.
- `SENT` : the call has been sent to the RPC server successfully

[3] JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999.

- `WAIT_RESPONSE`: the client is waiting the server response for this call
- `SUCCESS` : the call has been executed successfully and its result is now available
- `ERROR` : an error occurred during the call processing
- `DELETED` : the call has been removed from the queue

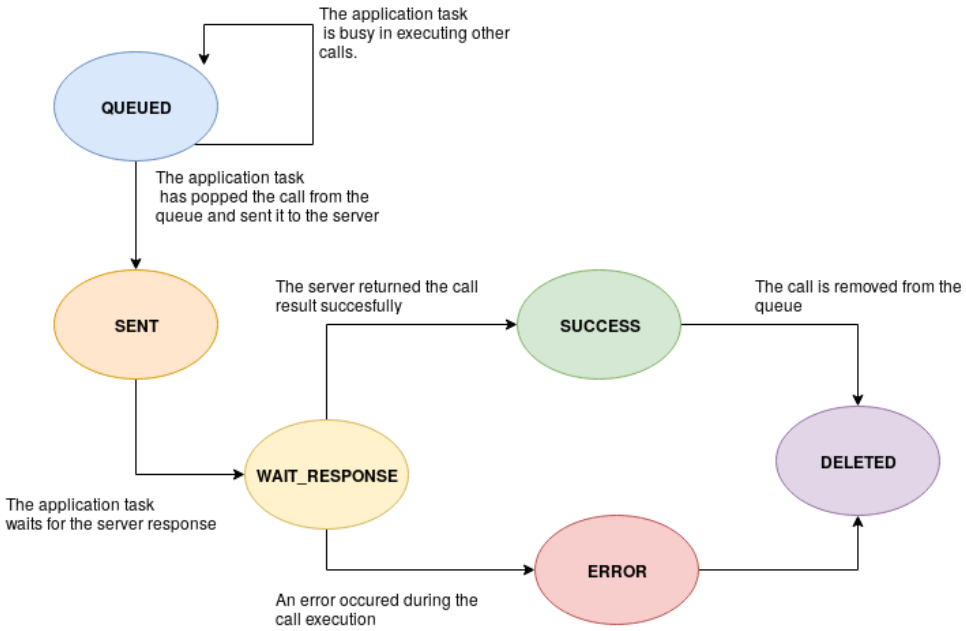


Figure 3.7: The state-transition diagram for a call-object. (Image created using `https://draw.io`)

The calls queue is accessed in a circular-buffer manner and its dimension, in terms of number of calls, is static and do not rely on dynamic memory allocation. The module's user can then set the dimension, according to the application requirements, through a C-`define` pre-compiler instruction and select therefore the optimal trade-off between memory occupation and performances.
Summarizing, the `JSONRPC_Client` interface provides function, to be used by upper layers, allowing the caller to:

1. Push a new RPC call in the queue
2. Retrieve the status of a call previously submitted
3. Retrieve the result of an executed call
4. Delete a call

5. Check the queue status (empty/full)

In addition, the module implements the application-task function (`JSONRPC_Task`) that is in charge of maintaining the queue, convert calls-object into JSON, sending calls over the TCP socket, retrieving the response and parsing it from JSON. The sequence diagram in *figure 3.8* illustrates an example of RPC call execution.

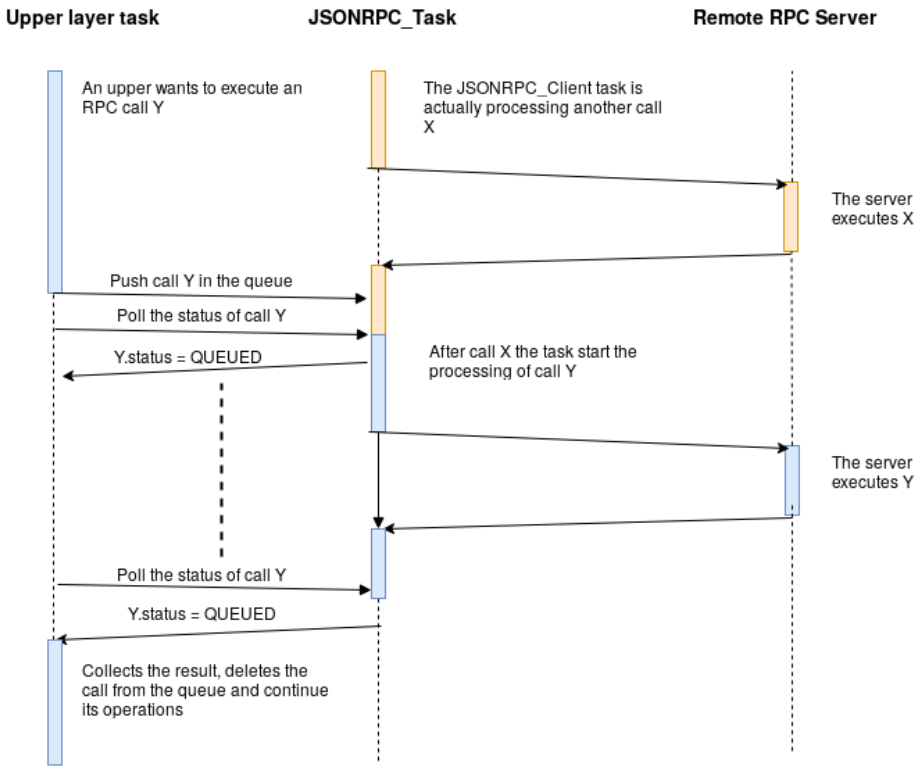


Figure 3.8: A sequence diagram illustrating how the developed JSONRPC client interacts with upper layers requesting remote call to be executed. (Image created using https://draw.io)

3.2.3 The Ethereum interaction layer

The Ethereum interaction layer is the one of the framework that implements a subset of the functions provided by the Web3 API. Each of the provided functions is in charge of encoding the necessary parameters for the correspondent RPC call to be executed, as well as adapting the returned result into user friendly C-types.
Actually the provided functions allow the caller to check the status of an account (i.e. nonce and balance), perform transaction and message calls and request a specific set of block from the chain. The fundamental data-structure of this layer is the `WEB3_ETH_OBJ` that holds the information about the RPC call object as well as the state of the Web3 call.

Indeed, the implementation of these functions follows the state-machine model and so, a given web3-call evolves through the following states:

- `CALL` : Parameters are encoded into JSON format and the proper RPC call is pushed into the queue of the JSONRPC client module.
- `WAIT` : Polling on the call-object status to check when the result is ready or an error is returned.
- `DONE` : The result is ready to be processed.
- `ERROR` : An error occurred.

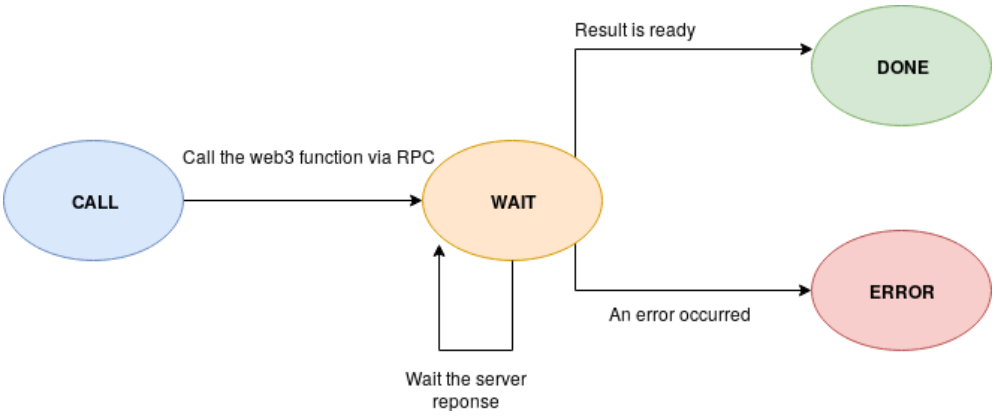


Figure 3.9: The state-transition diagram for a web3 call in the developed framework. (Image created using https://draw.io)

3.2.4 The Application interaction layer

The Application interaction layer represents a wrapper of the Ethereum interaction layer in the sense that it hides the details for performing specific operations on the blockchain (eg. signing a transaction before sending it as raw). It basically wraps all the functions of the layer below and includes performs the pre-processing and post-processing operations needed for some of them.
Again they are internally implemented as state-machines, cycling through the following possible states:

- `WEB3TASK_INIT` : In this state all the pre-processing is performed before the web3 function is called
- `WEB3TASK_WAIT_RESPONSE` : The state machine waits the web3 call to be executed and return its result so that to perform the post-processing operations.

- `WEB3TASK_IDLE` : Nothing to be done
- `WEB3TASK_ERROR` : An error occurred.

Chapter 4

Agri-Food traceability through Blockchain and IoT

Recent studies have demonstrated that customers' loyalty in food companies and in all the products they sell has been drastically decreased year after year.
In 2018 The Center for Food Integrity has published its annual report about consumers habits and feelings when buying food. The investigation showed that an increasing number of customers are concerned about what they eat, thus they are asking for more information about the food they buy, in order to chose the healthy one [17]. However, most of them do not really trust about the information written behind the product's package, especially when it does not directly come from farmers but it went through some transformation processes instead. In addition the awareness of consumers about the damages caused to the environment by some farming and breeding processes, is encouraging them to not buy some categories of products.
As shown in *figures 4.1, the research carried out by IFS and GfK has further confirmed the statements above. 4.2* [18].
The lack of trust in food producers is the consequence of a minimal customers' knowledge about the whole supply-chain and also the fear of a globalized market that may allow to fraudulent producers to sell dangerous or counterfeited products.
In particular, this last aspect is strongly critical both for consumers, who are subjected to risks related to their health, and "honest" producers, who not only suffers the economical damages but are also affected by the side-effects of the distrust of buyers.
In the year 2018 the Italian Central Inspectorate of Quality Protection and Fraud prevention of food products (ICQRF) has registered an increment of 58%, compared to the previous year, of crimes related to food frauds, seizing about 17.6 tons of goods, for an equivalent value of over 34 million euros. The wine sector resulted to be the most affected

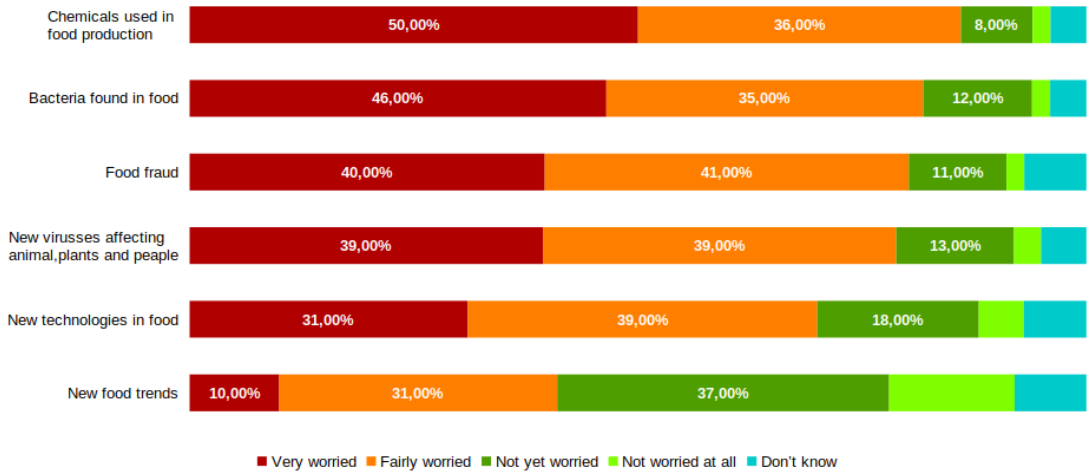


Figure 4.1: When was asked to the consumers their level of concern with respect to some risks related to food, most of them appeared to be aware and significatively worried about such risks.
Source: Etienne, Chirico, McEntaggart, *et al.* - *Consumer perceptions of emerging risks in the food chain*,2018

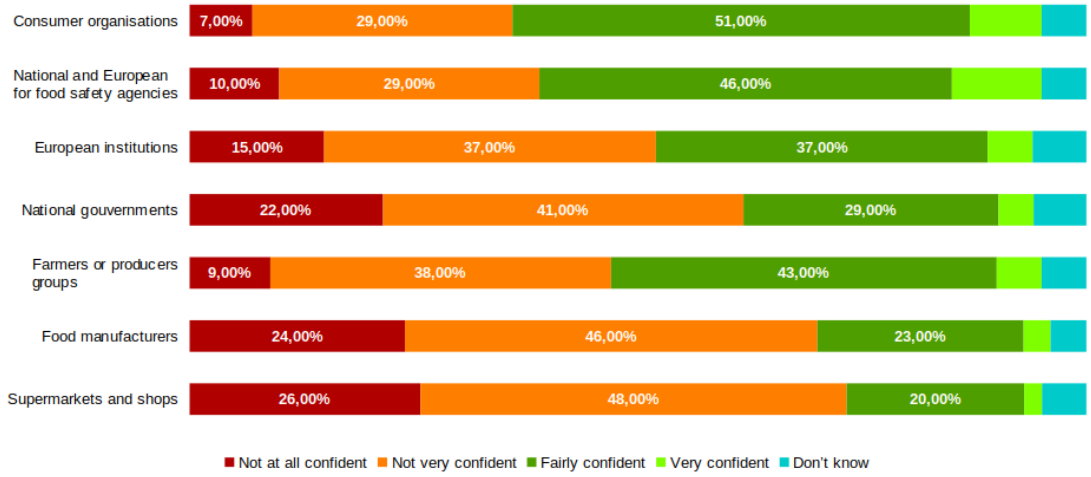


Figure 4.2: When was asked to consumers their level of confidence in different authorities and organisations involved in the agri-food chain, most of them seemed to be very diffident, especially with respect to food producers, final retailers and governments
Source: Etienne, Chirico, McEntaggart, *et al.* - *Consumer perceptions of emerging risks in the food chain*,2018

by this phenomena[19].

4.1 The concept of Food Trust

From the previous analysis, it is clear that consumers' confidence in the agri-food market is the result of their concerns or certainties on various aspects related to the product they intend to buy. One of the most important factor to restore a good level of reputation

of food market is to guarantee the **food-safety**, which is every day undermined by the numerous cases of food hazards.
In July 2018 the European European Food-Safety Authority (EFSA) published a technical report about a case of Salmonella Agona infection possibly linked to Ready-To-Eat (RTE) food. Overall they reported 147 outbreak cases among which 122 registered since January 2017 and 25 dated back to 2014 and 2016. United Kingdom was the first to report the outbreak, and it was also the country with the highest number of cases (123), besides Finland (15) German and Denmark. Biological analysis revealed that the contamination regarded RTE food products containing cucumbers, but, as stated in the report, *"so far it has not been possible to identify the specific point in the production chain where the contamination occurred"*[20].
The lack of information about transformations and events under which the final product has went through during the supply chain, poses the basis for a serious alarm about the health of citizens and, despite the efforts of government in control and prevention, the current infrastructure turns out to be unsuitable and inefficient in the fight against this phenomenon.
Another important aspect of food trust is related to the concepts of **food-integrity** and **food-authenticity**, which have a significant impact both on the healthcare and the economy of countries.
Food-Integrity defines the state of being undiminished and unaltered with respect to its original nature, which means that no unapproved and undeclared transformations and alteration have been made on such product. A recent and dramatic incident related to food-integrity has occurred in September 2012, causing the death of 42 people in Czech Republic, Poland and Slovakia due to methanol poisoning of some spirits. During the same month, all the sales and the exports of liquors were banned by the Czech government, until the identification of the source of contaminated alcohol. A Slovak businessman and a Czech spirit company owner were found to be involved in the fraud and so they had been arrested and sentenced to life imprisonment in May 2014.
The Food-authenticity definition, instead, is more related to economical aspects, like frauds, that generally do not represent a risk for the health of consumers. However food frauds and counterfeiting are hurting the markets of a lot of countries, causing losses for billions of euros per year. The most subjected products to counterfeiting threats are those that earn a great economical value because of their origin territory or brand, like oil, wine and meat. In July 2016 the European Union Intellectual Property Office (EUIPO), has published a research about the economical costs of intellectual property infringement in spirits and wine. The results showed an enormous damage for the legitimate industries, with estimated losses for approximately 1.3 billion euro of revenue annually [21].
The last but not least component of the concept of food trust is linked to customers'

attention with respect to the sustainability of processes and the agri-food chain. Indeed consumers are becoming ever more aware about the role they play in the game for building a sustainable food market. Most of them, especially young ones, base their decision making when buying a product not only on the "classical" factors (price, nutritional values, brand, etc.) but also on the impact that the product they buy has on the environment. The expansion of this new category of customers, known as *Ethical Consumers*, is leading this new market sector to grow very fast, registering in 2015 a growth of 5.3% and of 9.7% in 2017 (UK Market) [22].

4.2 Blockchain for food traceability

It is evident that there is a strong necessity of an enhancement in the monitoring and certification processes for overcoming these issues. In the actual situation most of the compliance data and information is audited by trusted third parties and stored either on paper or in centralized databases. However these approaches suffers from many informational problems such as the high cost and inefficiency of paper-based processes and fraud, corruption and error both on paper and IT systems.
Thus, there is need of a new trustworthy approach for registering information about the food chain and their availability to customers is actually finding a potential powerful partner in the arising blockchain technology. Indeed, the application of the blockchain technology in the agri-food sector could lead not only a better and more trustworthy traceability process, but it also could represent a powerful partner in the assessment of product's value from the point of view of the buyer. For these reason, there are actually several experiments and proof-of-concepts aiming at finding a way to apply the blockchain technology into this business field.

4.2.1 Quadrans: the industrial Blockchain

The Quadrans platform is an open source and public blockchain network that runs Smart-Contracts and decentralized applications. The first implementation of this platform, born as a fork of Ethereum, has been conducted under the Food-Chain SpA brand, and its initial goal was to enable traceability, transparency and authenticity of the information within agri-food supply chains. Then further researches and improvements lead this platform to evolve for serving different type of industries.
Quadrans maintains the full compatibility with Ethereum's architecture enabling an immediate and seamless migration of solution based on the latter.

4.2.2 FoodChain traceability system

Food-Chain's mission is to enhance the traceability process of food's supply chain through the blockchain technology. The company has developed a platform, based on Quadrans' infrastructure, that provides services for both companies and final buyers. Indeed, the former can use the platform as a common ERP software, registering products, batches and processes and in addition registers any information related to product's lifecycle (documents, certificates, media etc.). Such information are immutably registered into the blockchain, on top of which the platform lies, by means of Smart-Contracts. At the end of the product's transformation process the platform generates an Smart-Label (as QR code or RFID tag) which uniquely identifies the product and therefore be scanned by the product's buyer for viewing the entire traceability process.
Another important feature provided by the platform is that it allows many companies to co-operate together within the network, allowing to follow the transformation cycle of a product even when it passes through the production and distribution chains of different actors.

4.3 Combining IoT and Blockchain to empower the traceability process

The platform powered by FoodChain has a big intrinsic potential which can allow to track the story of a food product *from seed to table*, meaning that the traceability process can be verified starting from its origin until to the final customer's table. This potential can be truly unlocked by enabling the platform to collect, and register on the ledger, data gathered directly from the environment (that could be of any kind like farm fields, food transformation chains, logistic processes, etc.) by means of Internet-of-Things devices.
Thus, in order to give a proof-of-concept of the interaction between blockchain and IoT worlds, an example use case about traceability has been carried out in collaboration with this company.
The presented use case will cover a single part of the supply chain of fish and seafood products. The scenario in which the use case is applied is about the cold-chain monitoring during logistics operation. In such context the IoT device is assumed to be installed into a refrigerated truck that transports seafood from harbor's warehouses to the final fish-shop. The device is enable to connect to the Internet through a 3G or 4G connection and is in charge of monitoring fridge's temperature and truck's position by means of a temperature sensor and a GPS module respectively. When the data are collected the device then send them to a proper Smart-Contract, in charge of storing them in the blockchain, using the IoT-blockchain-gateway presented in the previous chapter.

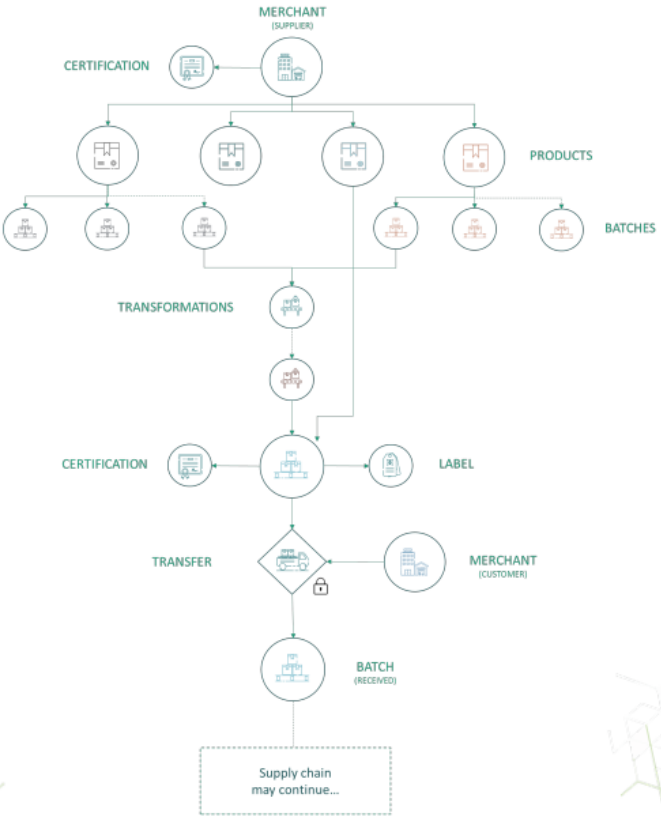


Figure 4.3: The representation of the data-model which abstract the food supply chain in FoodChain's traceability system. It can be noticed that it has a tree structure, where each node has a given typology and is uniquely identified into the ledger, as well as the relationships existing between them. (The use of this image have been conceded by FoodChain SpA

4.3.1 Use case description

When Fresh-Fish's fishing boat arrives at the harbor of Gallipoli every morning, fishermen store the catch into company's warehouses. Here each batch of fish (e.g. one kilogram of mussels) is registered into Food-Chain's platform, through its web interface, and is identified with a unique ID which can be written into an RFID tag. Then, according to fish shops daily requests, each batch is charged into refrigerator trucks in order to be delivered to its final destination. When a batch is charged on the truck, the device installed onto it, scans the RFID tag on the batch package so that the device knows for which product it is going to start the monitoring process. Once the truck starts its delivery path, the IoT device starts to periodically monitor the temperature and the GPS position of the truck and sends data to the proper smart-contract for being registered, providing also the IDs of the batches under the current monitoring process. At the end, when the truck reaches its finals destination, batches' tags are scanned again for registering that the delivery has been completed and

that the recipient retailer is now the owner of such products. At this point, the retailer can use the blockchain platform for generating QR codes for the products he is going to sell, so that its customers can verify the story of the mussels and check if any violation of the cold-chain parameters has occurred.

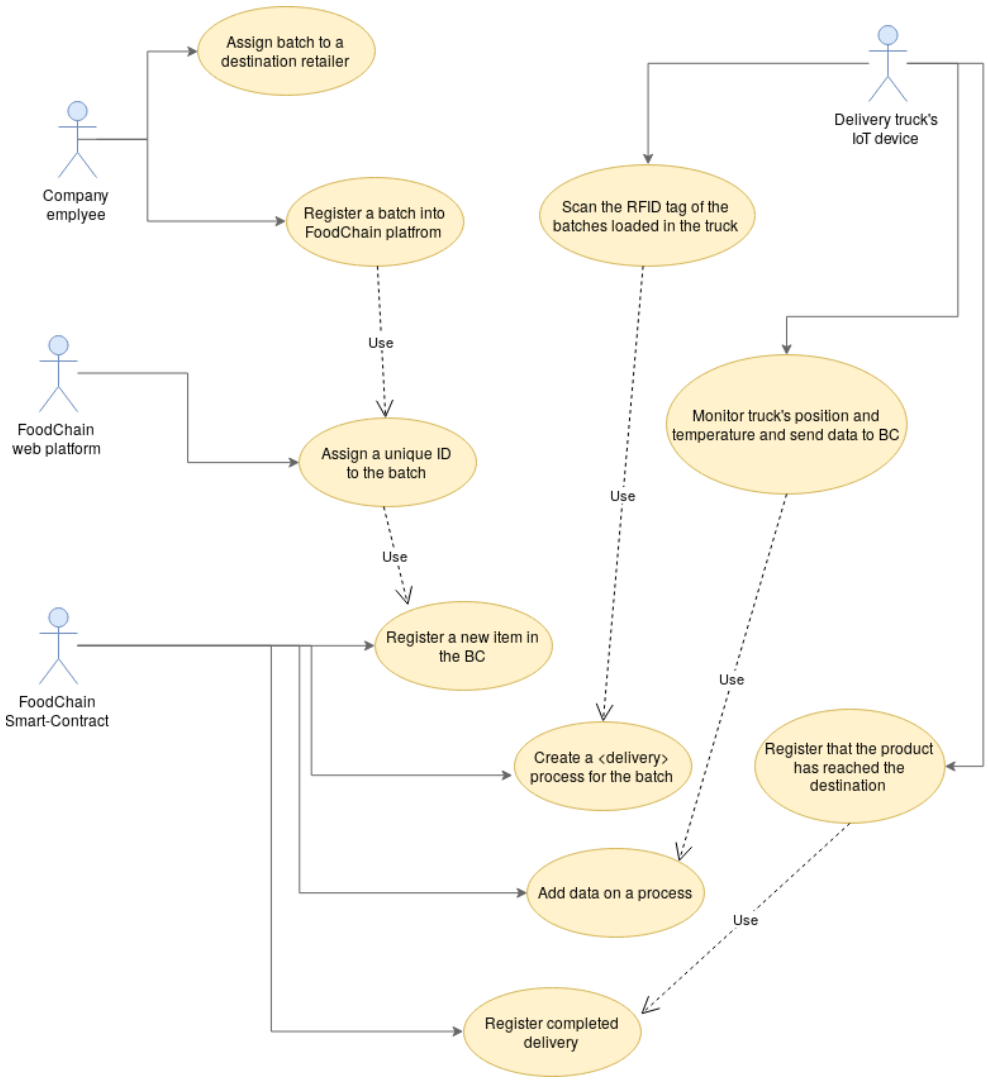


Figure 4.4: The UML uses case diagram of the system. (Image created using https://draw.io)

In the use cases diagram shown in *figure* ?? there can be identified four different actors:

- **Company employee** : is the person of the company in charge of registering every batch of fish that arrives at the harbor's warehouse every morning.
- **FoodChain's web platform** : is the web-base interface provided by the traceability services provider company that allows to interact with its own Smart-Contract,

running on Quadrans infrastructure, in a user-friendly way.

- **FoodChain Smart-Contracts** : are the SC developed by FoodChain in order to realize the decentralized-application that enables industrial traceability processes through the blockchain technology.
- **Delivery truck's IoT device** : is an IoT device that monitors the truck's and temperature position, scans the RFID tag of the items loaded in the truck and interacts with FoodChain Smart-Contracts through the gateway device described above.

A description of some example scenarios for the use cases involving the IoT device is given in *tables* ??

Step	Description
Pre-Condition	*The item X exists and a <delivery>process p has been activated for it*
1	Read Temperature from temp. sensor
2	Get the position from the GPS module
3	Convert data in ABI format in order to send them to the target Smart-Contract
4	Build the transaction object (set the correct value to the transaction's fields)
5	Convert the transaction into object into a bytestream and sign it using the device's private key stored in the EEPROM
6	Build the rawTransaction string and call the eth_sendRawTransaction method on the remote RPC server
8	Wait the server to reply with the hash of the transaction submitted on the chain
Post-Condition	*Gathered data are now immutably stored in the BC for the item with id X.*

Table 4.1: Example scenario's description for the use case *"Monitor truck's position and temperature and send data to BC"*

4.3.2 Application's outputs

Here are shown some screenshots of both the outputs of the web platform and of the IoT device for the scenario presented above.

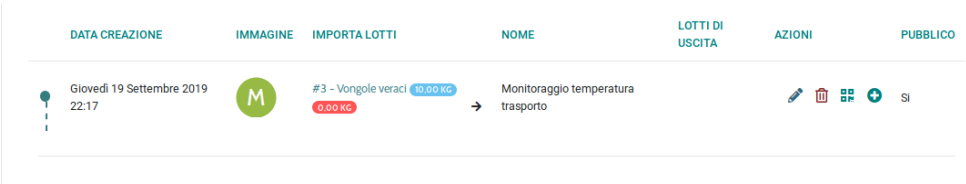


Figure 4.5: A screenshot of platform's web page showing the list of processes for a certain batch

Figure 4.6: A screenshot of the output shown by the web platform when a user click to see the details about a process. Even if most of the information here are presented in their raw format, it can be seen in the picture the hierarchical relationship of item's starting from the company one until the batch's one

```
Ricevuta transazione: 503d32bb03aec8dc8289def9f8fe8cc18073d8e26e5e8a15c939b667d5142f7
Identificativo univoco: f00d0000000000000cb1eeb1e5fa5b40ad80542065f8ea50ea0df93821e645
Timestamp: Giovedì 19 Settembre 2019 22:17
Proprietario: 0x45889efE02C3F7826A7edf7CA57Ea38dE451248f
Numero figli: 0
Numero parenti: 1
Id dei parenti: ['f00d0000000000000821270277773b4074caa5690461133d7623f193b0f77ccfc7']
Flags: 0x2d binary: 101101
UPDATABLE:1
ENFORCE_REVISIONS:0
RETRACTABLE:1
TRANSFERABLE:1
DISOWN:0
ACTIVE:1
Numero revisioni: 1
Ultima revisione IpfsHash: QmWZq7uqAvf4kX35ch1BN4M4BKDcFiriiXhrT3iv3vD2Wf
IPFShash: {}
```

Figure 4.7: Details about the transaction performed by the device to register temperature and position information (in IPFS hashed form)

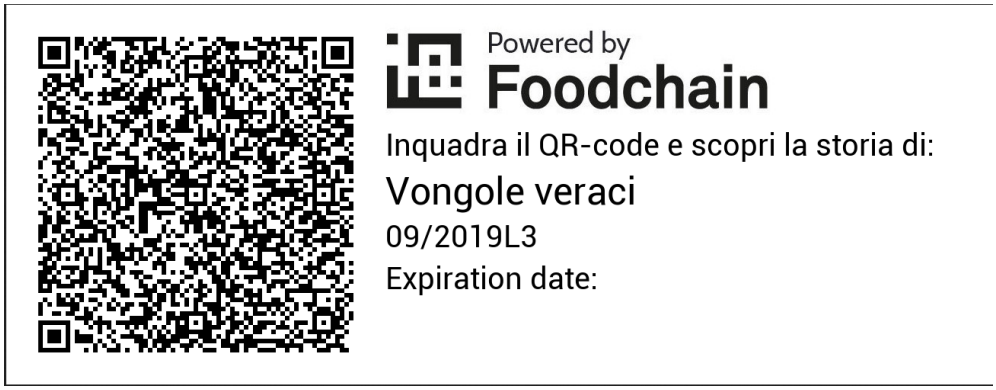


Figure 4.8: The final Smart-Label generated at the end of the process by FoodChain's web app platform

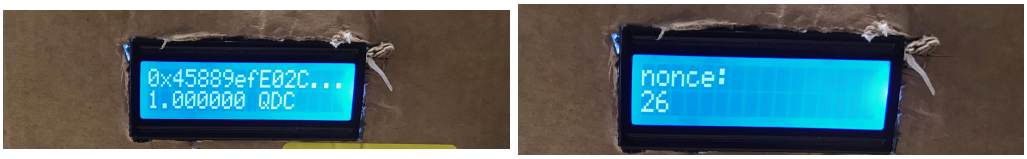


Figure 4.9: The IoT device's account status parameters (balance and nonce). These values are periodically updated by the device by calling respectively eth_getBalance and eth_getTransactionCount functions passing its address as parameter (0x45889e...)

```
INFO [10-09|10:19:04.123] Setting new local account                address=0x45889efE02C3F7826A7edf7CA57Ea38dE451248f
INFO [10-09|10:19:04.124] Submitted transaction                    fullhash=0xee0ccf9336decadb0d7542a0b686e54bb540544483d989684967e366fdedf78a recipient=0x1fCffa1B837422
6C3412773Eb28A8aC7D624D5E
```

Figure 4.10: The message shown on RPC server's console when the eth_sendRawTransaction call requested by the IoT device (with address 0x45889..) has been correctly executed and the transaction has been broadcasted to other peers

Chapter 5

Conclusion

5.1 Results and analysis

Summarizing, the integration between two emerging and disruptive technologies as Internet-of-Things devices and blockchain-based infrastructure for decentralized application can bring benefits in a very wide variety of fields spanning far beyond the simple use case presented in this document. The proposed and developed system gives a proof-of-concept of a possible path to follow for a successful and efficient interaction between Internet-of-Things devices and decentralized applications based on a blockchain ledger. The *blockchain-enabled gateway* presented in this document satisfies the essential requirements for performing secure and reliable data-storage operation onto the BC infrastructure and, at the same time, provides both hardware and software interfaces for an easy and seamless integration even with already developed applications. From the side of the decentralized applications, the possibility to add functionalities that can use data coming directly from the environment, without the doubting of their integrity or authenticity, represents a powerful tool that can be exploited to enhance a lot of processes. Finally, from the point of view of business actors, the services offered by the resulting kind of systems allow the latter to run a new form of marketing, focused on product and processes transparency. However, the possible application of this promising technologies combination can go even further from traceability and business processes and be applied for example to healthcare, infrastructures, services and so on.
In conclusion, with respect to other solution, the one presented in this document allows to develop decentralized applications of any kind that could interact directly with the Cyber-Phisical-Space, keeping costs low because they do not need to rely on any other service for accessing to the distributed network and at the same time, satisfying the constraints about data-integrity and authenticity.

5.2 Further improvements

At the state of art, most of the complex operations performed by the system (hashing, ECDSA, etc.) are implemented in software. Even if from various tests it didn't appear to be a critical aspect, in terms of execution time, it would be worth to explore possible implementations of the system that can rely on hardware accelerators (maybe implemented on an FPGA connected to the micro-controller) which can be used for speeding up the execution of such tasks.
Another important feature provided by the designed architecture, and which should be worth to explore better by developing proper use cases and DApps, is the one allowing Smart-Contracts running on the blockchain to asynchronously trigger some action on specific target IoT devices, identifiable on the network through their public address. Shortly, the idea would be to implement some logic on the IoT devices that periodically downloads the latest N blocks added to the chain, and search into their logs fields if any event has been emitted and is targeted to them. However, it should be evaluated with further studies how these concept can be applied in soft or hard real-time embedded systems by an exhaustive analysis which should take into account both the delay of the network and the architecture itself and also the delay related to the block-time parameter of the blockchain. Probably explorations in this direction may lead to the development of IoT oriented blockchain infrastructures that optimizes their parameters in order to be more adaptable to those contexts where timing constraint are critical and sometimes restrictive.

References

[1] H. Group, *The Internet of Things: Networked objects and smart devices*, 2010.

[2] B. Company, *Unlocking Opportunities in the Internet of Things*, 2018.

[3] A. Dorri, S. Kanhere, and R. Jurdak, *Towards an Optimized BlockChain for IoT*, 2017.

[4] V. Sivaraman, H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, *Network-level security and privacy control for smart-home IoT devices*, 2015.

[5] D. Chaum, *Blind signatures for untraceable payments*, 1998.

[6] S. Nakamoto, *Bitcoin: a Peer-to-Peer Electronic Cash System*, 2008.

[7] V. Buterin, *Ethereum - White Paper, A next generation Smart-Contract & Decentralized Application Platform*, 2013.

[8] G. Wood, *Ethereum: A Secure Decentralised Generalised Transaction Ledger, Byzantium Version e7515a3*, 2019.

[9] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Keccak specifications*, 2009.

[10] C. Chinchilla. (2019). Ethereum Wiki, [Online]. Available: https://github.com/ethereum/wiki/wiki/RLP (visited on 09/30/2019).

[11] W. Diffie and M. Hellman, *New Directions in Cryptography*, 1976.

[12] R. Rivest, A. Shamir, and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, 1978.

[13] (2016). EIP-155, [Online]. Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md.

[14] J. Chow. (2016). Technical Introduction to Events and Logs in Ethereum, [Online]. Available: https://media.consensys.net/technical-introduction-to-events-and-logs-in-ethereum-a074d65dd61e.

[15] Microchip. (2019). MPLAB Harmony Framework.

[16] J.-R. W. Group. (2007). JSON-RPC Specification, [Online]. Available: https://www.jsonrpc.org/specification (visited on 10/01/2019).

[17] "A dangerous food disconnect, When Consumers Hold You Responsible But Don't Trust You", The Center for Food Integrity, 2018.

[18] J. Etienne, S. Chirico, K. McEntaggart, S. Papoutsis, and E. Millstone, "Consumer perceptions of emerging risks in the food chain", 2018.

[19] " ICQRF - 2018 Activity report ", 2018.

[20] "Multi-country outbreak ofSalmonellaAgona infectionspossibly linked to ready-to-eat food", European Food Safety Autorithy, 2018.

[21] " The ecnomic costs of IPR infrangement in spirits and wine ", European Union Intellectual Property Office, 2016.

[22] "Younger consumers drive shift to ethical products", *The Financial Times*, Dec. 2017.

Acronyms

ABI Application-Binary-Interface

BC Blockchain

CPS Cyber-Phisical-Space

DApp Decentralized-Application

DLT Distributed Ledger Technology

DNS Domain-Name-Server

DOS Denial-of-Service

ECC Elliptic Curve Cryptography

ECDSA Elliptic Curve Digital Signature Algorithm

EFSA European Food-Safety Authority

EML Ethereum Machine Language

EUIPO European Union Intellectual Property Office

EVM Ethereum Virtual Machine

IoT Internet-of-Things

JIT Compilation Just-In-Time Compilation

PoW Proof-of-Work

RLP Recursive-Prefix-Length

RPC Remote-Procedure-Call

RTE Ready-To-Eat

UTXO Unspent Transaction Output