Master Internship
Report

Comparative study of EOS and IOTA blockchains in the context of Smart IoT for Mobility

Submitted by

Luc GERRITS
Master 2 ESTel
Université Côte d'Azur

Under the guidance of

François VERDIER
Professor

Department of Electronics
Laboratoire d'Electronique Antennes et Télécommunications
Campus SophiaTech - Bâtiment Forum
930 route des Colles, BP 145
06903 Sophia Antipolis cedex

Internship 2020

Abstract

Blockchain can record data in a decentralized, secure ledger. Smart Contracts are a novel technology extending blockchain capabilities. It allows secure automated execution of business logic inside the blockchain. The increase in connectivity for connected devices allowed blockchain and smart contracts to create new secure and automated applications.

The internship aims to study two blockchain (EOS.IO & IOTA) in the context of the Smart IoT for Mobility (SIM) project. This report expands previous work on the SIM project by highlighting the key challenges of using blockchain and Smart Contracts along with constrained embedded devices.

By exposing SIM use-case requirements, we can compare multiple blockchains (EOS.IO, IOTA, and Hyperledger Sawtooth). This comparison brought us to build custom implementations and make an in-depth study of the programs. The developed programs are also analyzed to conclude on its compatibility with the project.

Contents

Chapter 1

Acknowledgements

It is a pleasure for me at the end of this training course to express my gratitude to :

Pr. Verdier François: the teacher-researcher, who took care of my internship during these 6 months, and who broadened my field of knowledge, whether in the field of electronics (embedded system), in the field of development (SystemC-TLM, blockchains), or in the field of office automation (writing reports, conference papers, speech and making presentations), but also in laboratory life in general. Allow me, therefore, to express my most sincere gratitude.

M. Roland Kromes: a colleague and friend who guided me towards interesting topics and provided me documents, data, and tools to create and develop my internship.

The Smart IoT for Mobility project team: to all the members of the Smart IoT for Mobility project that I have worked with and helped me, thanks

Academic personnel: who during these seven years have taught me a lot in many fields, such as programming, telecommunications, electronics, and many others. Thank you very much.

I apologize for not being able to mention everybody, but I would like to acknowledge all those who contributed to the success of my internship and my studies.

Chapter 2

Introduction

2.1 Workplace

For my last year of the masters degree, I did a tutorship at LEAT (Electronics, Antennas, and Telecommunications Laboratory), a public science and technology institution based in Sophia-Antipolis. The LEAT laboratory works mainly on :

- Antennas, electromagnetism and microwaves
- Communicating objects, Wireless network optimization, Embedded systems and Systems on Chip (SoC)

The LEAT is a laboratory located in Sophia Antipolis, more precisely in the SophiaTech campus at 930 Route des Colles.

Thanks to the laboratory, we had at our disposal a room to work whenever needed.

LEAT
LABORATOIRE D'ELECTRONIQUE
ANTENNES ET TELECOMMUNICATIONS

The internship takes place in the EDGE (**E**dge computing & **DiG**ital **E**lectronic) section of the LEAT laboratory. The EDGE section field of research is embedded systems, precisely, optimization of sensor networks wireless, SoCs, and communicating objects.

This internship involves the following areas of research:

- Distributed networks using the IoT
- Embedded application design for distributed networks
- IoT modeling and analysis
- IoT architecture study
- IoT architecture design/modeling

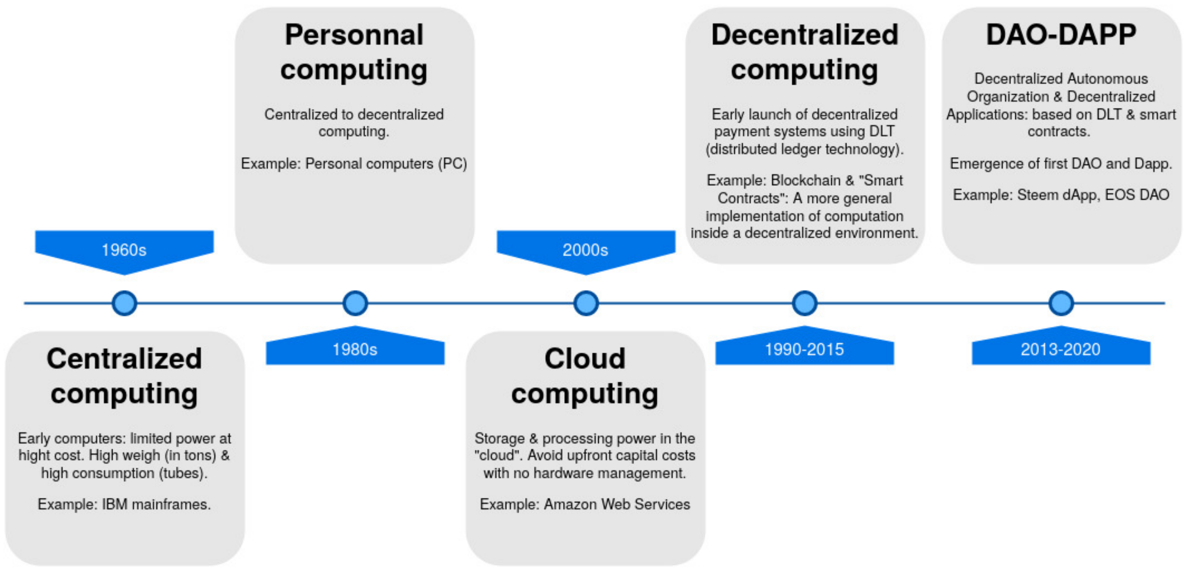
2.2 Rise of Blockchain

2.2.1 Distributed Ledger Technology (DLT)

In the fall of 2009 was the first successful creation of truly decentralized ledger able to manage a currency (Bitcoin [1]). Bitcoin's goal is to offer a pseudonymous, decentralized, immutable, and secure currency solution that could create a new economy. It was also the first practical establishment of the **blockchain**: a ledger where transactions are encapsulated into *blocks*. Each block is linked together with an identifier of the previous block, thus forming a *chain*.

After this event, Vitalik Butterin proposed a blockchain (Ethereum [2]) with **smart contracts**: a layer inside a blockchain that can execute a custom business logic (Nick Szabo [3] was the first to introduce the idea of the smart contract). Smart Contracts are described in Sect. 3.3.

The consensus in Distributed Ledger Technology (DLT) is how all nodes will agree to mark a transaction as valid. Multiple consensus algorithms exist, for example: Proof of Work (PoW), Practical Byzantine Fault Tolerance (PBFT), Proof of Stake (PoS), etc. Consensuses are described in Sect. 3.2.



1960s
Centralized computing
Early computers: limited power at hight cost. High weigh (in tons) & high consumption (tubes).
Example: IBM mainframes.
1980s
Personnal computing
Centralized to decentralized computing.
Example: Personal computers (PC)
2000s
Cloud computing
Storage & processing power in the "cloud". Avoid upfront capital costs with no hardware management.
Example: Amazon Web Services
1990-2015
Decentralized computing
Early launch of decentralized payment systems using DLT (distributed ledger technology).
Example: Blockchain & "Smart Contracts": A more general implementation of computation inside a decentralized environment.
2013-2020
DAO-DAPP
Decentralized Autonomous Organization & Decentralized Applications: based on DLT & smart contracts.
Emergence of first DAO and Dapp.
Example: Steem dApp, EOS DAO

Figure 2.1: Evolution of computer and data processing

Decentralization is an old concept (Fig 2.1). It appeared after centralized mainframes in the 1980s with "Personal Computers" thanks to the technology exponentially reducing transistor size, increasing computation power, and having lower component cost. The goal of decentralization at the time was an answer to mainframes issue. The aim of decentralization is to distribute computer power, reducing costs, diversify systems and allow participation of local individuals. This lead to reducing communication congestion, better security and management of the entire computer ecosystem.

After the year 2005, desktop computers, servers, and embedded devices where able to be interconnected through any network. Mobile phone antenna jumped from a Kbits/sec throughput (2G) to a multi-Mbits/sec connection (3G), and followed by the 4G generation increasing phone connectivity exponentially. Servers around the world

had a similar evolution after the introduction of commercial fiber cables. The internet speed increase brought the countries (and continents !) closer together. This reduction in latency and increase in speed allowed to implement application ideas.
These ideas could now apply *real time* concepts using the internet as exchange support.
⚠ The concept of *real time* is important in decentralized systems such as blockchain. Blockchain nodes must maintain their ledgers updated with the entire blockchain network, thus real time synchronization is crucial in consensus rules.

With the rise of faster and more reliable internet connection, the personal computer was then gradually replaced by "cloud" solutions. The entire world is maintained by a handful of organizations, making cloud politically and economically a centralized system. Nowadays, the idea of blockchain is to re-gain trust, economic equity, and stability by decentralizing and having a consensus that everybody agrees on.

2.2.2 Contradiction with IoT

Internet of Things (IoT) is an everyday objects that contains electronics which allow us to communicate data and interact with our world. These objects are constrained objects with limited energy. Meaning also small (or even no) storage capacity, computational power, and network.

DLTs are often associated with very resource-intensive systems due to the structure itself. Depending on the DLT software :

- The consensus algorithm (and thus the synchronization of the nodes) will use more or less network.
- The security (cryptography) used in the DLT will impact the required computational power needed. It is also the case if using smart contracts or not.
- The blockchain is an append-only ledger, meaning storage scalability is a crucial element to keep the system running

IoT are very limited and constrained device. This contradiction leads us to study IoT properties to implement them with blockchain technology: a technology that seems to have incompatible resource requirements. The use case in section 2.3 represents this challenge.

2.3 Context: Smart IoT for Mobility

2.3.1 Project Details

The internship is part of the Smart IoT for Mobility (SIM) project [4]. The project goal is to go forward with a new economy with a trans-disciplinary approach using the adoption of blockchain and smart contracts. The project targets IoT and industrial use-cases that can use this technology. The use-case is presented in Sect. 2.3.2.
The project partners are :

- KAIROS (I3S, INRIA): Computer science
- GREDEG (MSHS): Research Group in Law, Economics and Management

- LEEN (MSHS): Experimental Economics Laboratory
- DL4T (Lawyers): Deep Law for Tech
- LEAT: Electronics, Antennas, and Telecommunications Laboratory
- Renault Software Labs
- Symag (subsidiary of BNP Paribas)

This internship aims to explore and have a practical approach to the technical implementation of blockchain and smart contracts onto IoT devices.

Apart from the technical challenge, one of SIM project challenges is the definition of a formal language to specify and verify smart contracts, while being able to manage the operations present on the blockchain.

This report will not make an in-depth study of the privacy and the social and economic point of view. As stated by Georgy Ishmaev [5] there is an entire grey area of blockchain usage combined with IoT in markets.

2.3.2 Use case

The global use-case that we have chosen is represented in Fig. 2.2 and represents a vehicle infrastructure. In this use-case, Renault's cars are connected to blockchains deployed on several clouds and can connect each time an accident will happen. We can observe that Renault's cloud is connected with other types of organizations like an insurance company, expertise, police, and car mechanics.

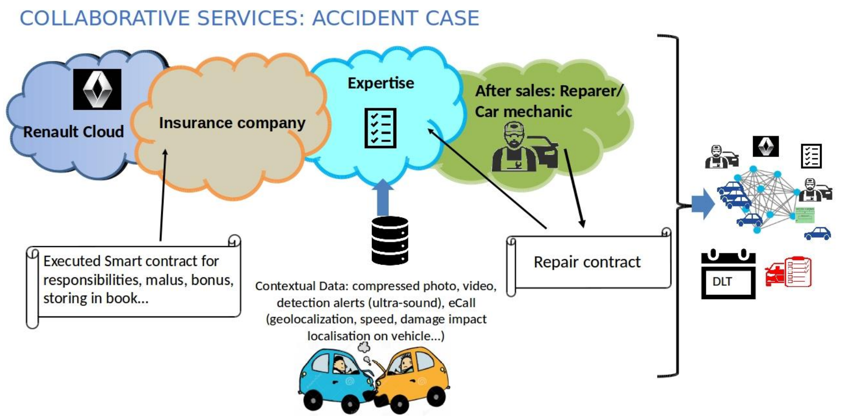

COLLABORATIVE SERVICES: ACCIDENT CASE
Renault Cloud
Insurance company
Expertise
After sales: Reparer/ Car mechanic
Executed Smart contract for responsibilities, malus, bonus, storing in book...
Contextual Data: compressed photo, video, detection alerts (ultra-sound), eCall (geolocalization, speed, damage impact localisation on vehicle...)
Repair contract
DLT

Figure 2.2: Use case of Renault's Smart Vehicle Book

One of the main reasons to use blockchain and not a standard database is to obtain a distributed record that cannot be modified nor deleted. It also prevents anybody from tempering data, enables tracing, and identifying data origin.

Thanks to the latest vehicle developments, connectivity is becoming less an issue for vehicle applications requiring internet connections. An important point to note on this use-case is the fact that each vehicle can be (or not) connected to these clouds. We cannot prove that each IoT is fully connected. If a car enters a no connectivity zone, the IoT on-board has to handle this situation.

Multiple solutions have been proposed, such as communication creating a protocol connecting vehicles between each other or to install a recording "black box" device in the vehicle. The connectivity loss solutions are only temporary alternatives: there is no 100% guarantee that the vehicle will be connected.

2.3.3 Previous work

In previous work, during a master 1 tutorship, we studied the possibilities to implement blockchain on an IoT. The tutorship aimed to explore the vehicle IoT limits by installing blockchain software and analyzing energy consumption. The IoT used during the tests is a Raspberry Pi 3 B+.

The first goal of the SIM project was to **fully** implement blockchain technology on the vehicles. This is the so-called **on-chain solution**: the IoT inside of the vehicle contains a full blockchain node. A full blockchain node means it holds the entire ledger, and the IoT in the car is part of the consensus.
However, if the vehicle's IoT does not contain a full blockchain node it is called an **off-chain solution**. In an off-chain solution, the car communicates with the blockchain (i.e. send transaction, data, and can send transactions that execute smart contracts).

The tutorship results and tests concluded that an embedded system doesn't take full part of the blockchain, and thus the SIM project has to use the off-chain solution. This result is a crucial element to understand why a car in our use-case won't contain a functional node of the blockchain, but rather a *client application* - executed by the IoT - that will exchange data with the blockchain.

Fig. 2.3 shows the difference between the two solutions. The **car contains the IoT**. The simple explanation is that constrained devices don't cope with the hungry resource that a blockchain node requires.

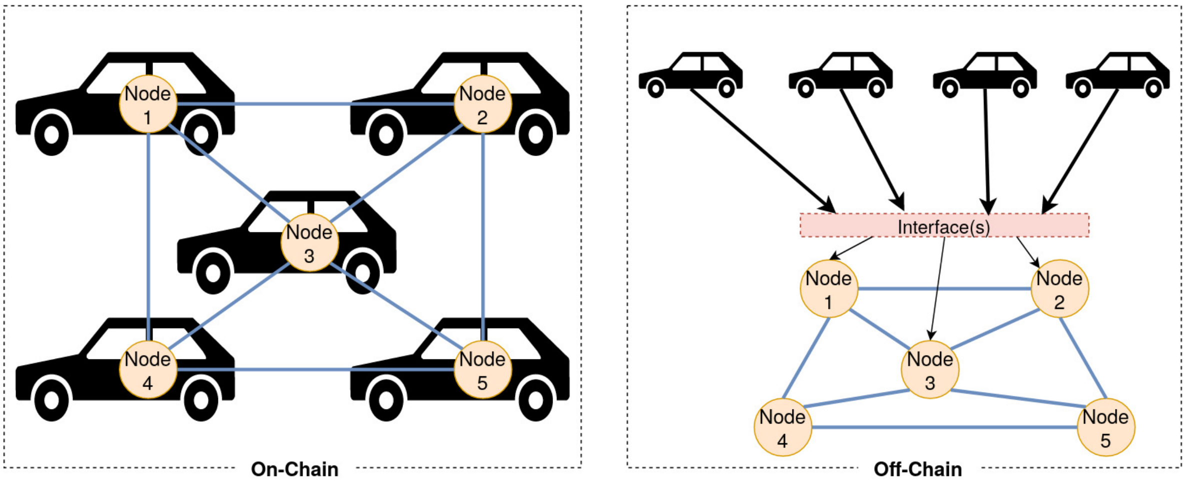


Figure 2.3: On-Chain and Off-Chain solutions

In this report, the practical work will always use the off-chain solution to compare three blockchains. We extract results and information about IoT integration to build a conclusion.

This internship aims to complete previous work and answer the use-case problem with the help of bibliography, practical implementations, and prior results.

Chapter 3

State of the art : Blockchain & IoT

3.1 Blockchain

3.1.1 Structure

A blockchain commonly describes a distributed (peer-to-peer network ruled by a consensus) ledger that records transactions (Fig 3.1). The structure of the blocks is a Merkle tree, where blocks are linked to previous blocks using a hash function to map each other.

The hash of a block is a fixed-size array of bytes resulting from a hash function. The hash function is a mathematical algorithm that maps input data into an arbitrary size array of bytes. The hash algorithm is deterministic, minimizing collision (a collision is when two different input result in the same output), and tends to be a one-way function (easy to compute, very hard to invert). Example: SHA-256 standard [6] hash produces a 32 bytes identifier of the data.

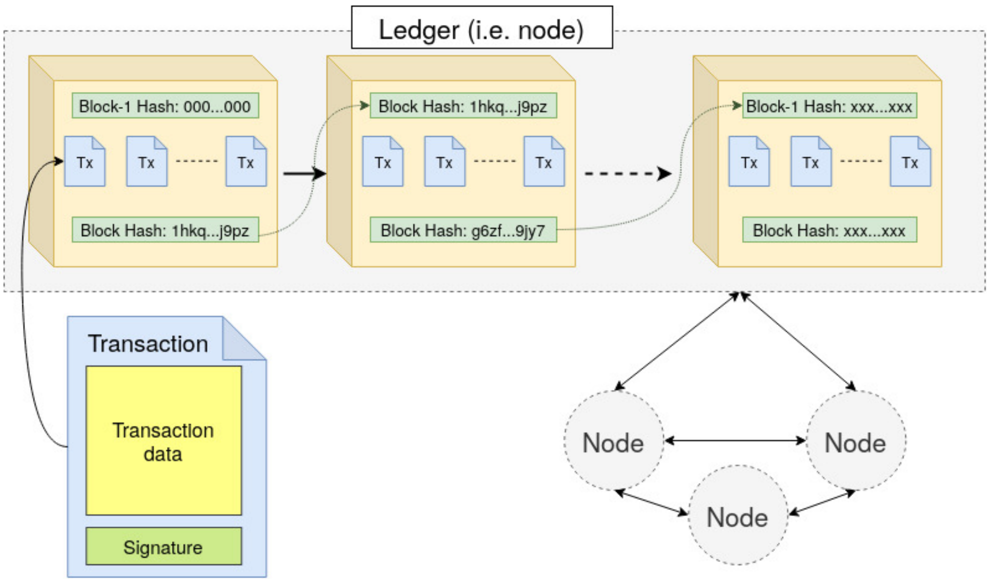


Figure 3.1: Blockchain Structure

Blockchain aims to provide a ledger that is secured cryptographically (detailed in Sect. 3.4), secure against faulty nodes using a consensus, is immutable (i.e. append-only) and distributed (i.e., a peer-to-peer network). Depending on the blockchain type (Sect.3.1.2), the blockchain can be considered trustless, meaning the parties can transact without trusting each other. Details about the nodes structure in practice are in Sect. 4.1.

The first usage of such technology was to create virtual currency (i.e., cryptocurrencies). After the integration of smart contracts, blockchain opened to more than only build new cryptocurrencies. The following applications demonstrate the potential of this technology:

- Healthcare [7]: Authors propose a solution for medical privacy issues using blockchian and Hierarchical Attribute-Based Encryption (HABE). The system manages access and guarantees patient-specific access policies. They propose multiple procedures for managing HABE, such as data storage, grant, and revoke permissions.
- Supply chain: This use-case has gained interest due to the demand. Blockchain features are direct answers to the use-case current issues (record immutability, tractability, etc.). The impact of a blockchain solution for this use-case even made IBM one of the first industrial blockchain solution provider for supply chain.[8]
- Voting [9]: The authors examined the current state of electronic voting and proposed a framework based on blockchain to prevent known electronic voting issues. The framework would prevent problems in the polling process, voting data management, security, and the voting process's authentication.
- Real estate: is a use-case that can answer the real estate record tracking of the properties and the property owners. The author in [10] proposes and implements a blockchain solution to prevent the previous issue and add meaningful tools for a game-theoretic stable-priced market (using smart contracts).
- Business process management
- Record Keeping
- Digital identity

3.1.2 Blockchain type

It is possible to distinguish three types of blockchains: public blockchains, private, and consortium. It defines the blockchain access level from the client's point of view (i.e. blockchain accounts).

When a blockchain is public, all the data on each node is entirely public and accessible to anybody. Also, any account can participate in the consensus. Ethereum, EOS, and Bitcoin are public blockchains.

A private blockchain requires authentication to access a node. It has an additional network security that restricts all communication with the blockchain. By configuring Ethereum and EOS it is possible to build a private blockchain.

It is also possible to combine private and public blockchain within a consortium. A consortium means that the blockchain is maintained by a predefined number of accounts (usually organizations). This type of blockchain is sometimes associated with a private blockchain, but the difference is that a consortium is managed by multiple companies and not by unknown individuals. Therefore, consortium blockchain is a better description of private blockchain used in real-life scenarios involving multiple organizations.

Our use-case fits perfectly with the consortium blockchain. Renault's blockchain is linked to insurance, expertise, sales, and other collaborative services. Practical realizations will be configured as a private blockchain because our goal is to develop the concept and client implementation, not the interoperability between services.

3.1.3 Transactions

Blockchain transactions are the tasks stored in the blockchain ledger records. A transaction contains data describing the task (e.g., a token transfer) and is always signed cryptographically.
A transaction is generally used for smart contract deployments and to interact with it, example: Ethereum and EOS, but NOT Hyperledger Sawtooth.

3.1.4 Security

Independently of the consensus, a DLT has various vulnerabilities and security threats. These threats are archived at the blockchain's core or network level. The most common are :

- The double-spend attack [21]: consists of spending the same cryptocurrency value twice by using the node synchronization latency.
- Cryptographic vulnerabilities: evolution of cryptography security or user lack of knowledge.
  Example: The user lacks awareness of cryptographic weaknesses in a command line to generate a private key: IOTA seed generation using a random function with not enough entropy. A common mistake is to use a seed generator built by a third party, the attacker records or alter the seed to be deterministic.
- Denial of Service (DoS): Network-level attack, saturating the network, making the service unavailable for hours, days, or more.
- Man in the Middle (MitM): an attacker intercept network activity through a relay secretly recording or altering the network data.
- Sybil attacks [10]: A large-scale attack using a large number of accounts to gain influence, obstruct or reshape the blockchain.

3.2 Consensus

3.2.1 Definition and consensus rules

The consensus ensures an agreement between all the nodes (also called peers) in the blockchain network. Behind the term, a consensus is an algorithm. Each algorithm differs from blockchain to blockchain. A good example of "same consensus rule, different algorithm" is Ethereum Proof of Authority (PoA) with *Clique* and *AuRa* different algorithms.

The following consensus algorithms are the most commonly used (there are much more of them) and suitable for practical implementation of consensus rules:

- Proof of Work (PoW) consensus is often used in public blockchains. The algorithm awards miners (i.e. a node participating in the consensus) to create a block (and it wins cryptocurrency) if it is the first node whom can solve a given cryptographic problem. Ethereum [2] and Bitcoin blockchain use this type of consensus. The benefit of this consensus is true trustless blockchain, but the disadvantage is resource hungry nodes with lots of useless energy consumption.

- Proof of Elapsed Time (PoET) consensus is a lottery-like algorithm, more often used in a private blockchain.

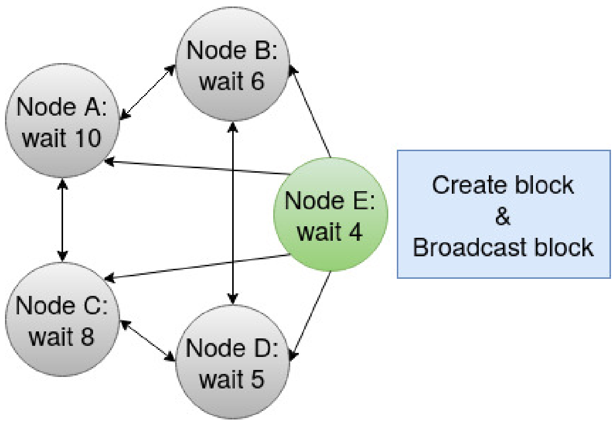


Figure 3.2: Representation of PoET consensus

  A randomly elected peer (using instructions originating CPU trusted environment) distributes a random countdown to its peers. The peer with the smallest time (i.e. lottery winner) is awarded to create the block (Fig. 3.2). In this consensus, the node does not need to compete with other nodes as in PoW. This results in lower consensus energy consumption without lowering trust among nodes. The drawback of PoET is to assume the hardware can create a trusted environment to execute parts of the algorithm instructions.

- Practical Byzantine Fault Tolerance (PBFT): is rather used in private blockchains than in public ones because, in these blockchains, minors are not used for cryptocurrency gaining but for obtaining a secured network. In BFT, if 3/2 of minors agree with the transaction's validity, it would be validated, and it would be taken into a block [11]. This consensus is tagged to work better with small blockchain networks such as a private blockchain with a limited number of nodes. PBFT has a step in the algorithm where nodes send an exponential number of network requests for synchronization.

- Byzantine Fault Tolerance (BFT) is a property of a system, not a consensus. A system that has the BFT property can continue to operate even if some participants are faulty.

- Delegated Proof of Stake (DPoS) consensus : is a consensus that is inspired by representative democracy (Fig. 3.3). The network participant (i.e. in EOS.IO anybody with at least 1 EOS token) can vote for a delegate. The delegates are the participants that are allowed to produce blocks. The delegates have to maintain the system 24-7/365 operational with close to 100% up-time. In EOS.IO there are a total of 21 block producers (top 21 that are most voted).

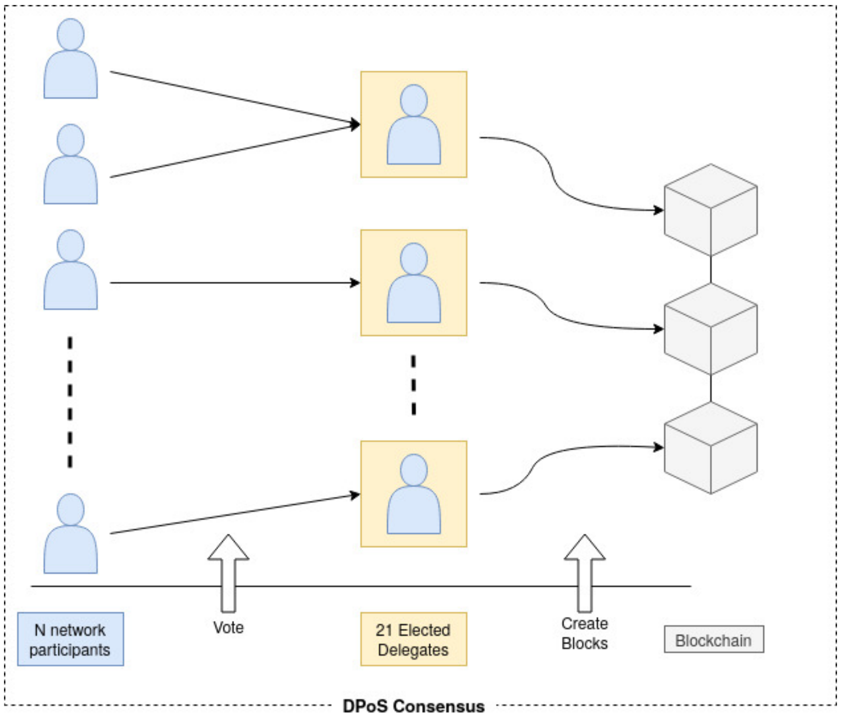


Figure 3.3: Representation of DPoS consensus

  Other then higher transaction speed, the main advantages of such consensus is that it consume less power on the overall network in comparison to PoW. In DPoS only a set of nodes can be part of the consensus, there is no computational power race such as Ethereum. PoW reach consensus after proving that nodes have performed an intensive calculation. This results in lower consensus energy consumption and increase in transaction rate, but accounts have to trust the delegates, making this consensus not trustless and brings it closer to a centralized consensus.

- Proof of Authority (PoA): is a permissioned consensus wherein consensus is maintained by a known group of nodes (called validators). The authorized validators are allowed to validate transactions and add blocks to the blockchain ledger. This consensus advantage is transaction speed and limited almost only by bandwidth. The disadvantage is that it is not suited for non-enterprise applications because the validators need to be trusted.

3.2.2 How secure are consensuses ?

The consensus has different security features depending on the algorithm. The algorithms features impacting security can be summarized with:

- level of decentralization: how many peers of the blockchain participate in the consensus ?
  - decentralized governance or quorum structure ?
  - does it have byzantine fault tolerance property ?
- the faulty peers limit: this limit defines the number of faulty peers or peers with bad intentions. A faulty peer is a node that is corrupted, intentionally modify blocks, intentionally, modify transactions, etc.
- authentication: if authentication is required, how secure is it ?

PoW consensus needs a least 51% of the mining power decentralized [12]. The majority of the mining power has to agree when adding a block. If one organization can reach that limit, it has control over the entire network. Thus it has control over the created blocks and the validated transactions.
With the rise of application specific hardware accelerators, it enabled very high performances to solve the cryptographic algorithm required to reach consensus in PoW. Blockchain consensus participants requires to buy hardware accelerators devices because it is impossible to compete using standard computers. This imbalance in computing power can be considered as a threat to decentralization. An example is Bitcoin: there is currently only three to four organization maintaining 90% of the PoW consensus.

By definition, in PBFT, the blockchain network will be ensured as live and safe if, with n nodes, there are less then $\frac{n-1}{3}$ faulty nodes. The goal of this consensus is to run for enterprise applications, meaning a limited number of peers connected with authentication. The authentication can be archived using different techniques, each with different security.

DPoS consensus security is relative to the delegates and the activity of the blockchain. In DPoS, if the voters are not active and do not participate in the voting process, the consensus risk being corrupted by an organization with sufficient voting power for a malicious or corrupt block producer delegate. Like any democracy, the goal is for electors to vote for the best delegates representing the system (from the point of view of the elector). If the system has no effective governance, a malicious delegate block producer as the power to influence the blockchain in various ways (centralize delegates, propose core changes, reduce up-time, changing blockchain parameters, etc.). A second main disadvantage of DPoS is that the voter influence is proportional to their stakes. Voters with a small amount of tokens have small influence or none compared to prominent stakeholders.

3.3 Smart Contract

3.3.1 Definition

The term "smart contract" (SC) is implemented differently depending on the blockchain but generally refers to the same concept: an agreement between multiple parties. As introduced in Chapter 2, a SC is a business logic that is executed inside of the blockchain. Thus smart contract benefits of the blockchain environment. It can be interpreted as a layer on top of transactions.

The extent to which the smart contract can influence the blockchain (i.e., it's flexibility) depends on the implementation of the smart contract within the blockchain.
A smart contract comparison of Bitcoin, Ethereum, Hyperledger Sawtooth, and EOS shows that each of them has a degree of smart contract flexibility, risks of security/privacy threats, and decentralization specific to the blockchain (See Table 3.1).

The smart contracts can allow to automate custom events of actions inside the blockchain. Depending on the smart contract implementation, it is possible to build a complex business logic. This complexity can allow in some blockchains to implement decentralized applications (dApp). dApps is a layer relying on smart contracts. To build a dApp, the software required is a blockchain, also executing smart contracts, and an interface between the application and the blockchain (commonly an API). More on dApp are in Sect.6.4.1

	Developer SC language	Compiled	Blockchain SC language
Bitcoin	Bitcoin Script	✗	-
Ethereum	Solidity	✓	Bytecode
Sawtooth	(python, JavaScript, C++, ...)	Depends	-
EOS.IO	C++	✓	Web Assembly (Wasm)

Table 3.1: Difference of Smart Contracts between blockchains

3.3.2 Ricardian Contract

The idea of a Ricardian Contract [13] is the definition of how the smart contract should behave and what it should do. Ian Grigg presented this concept along with Gary Howland as the Ricardo payment system in 1999-2000 [14]. He also was the first to introduce proposals and methods to resolve the intersection between smart contracts and Ricardian Contract.

It is "a method to identify and describe issues of financial instruments as contracts" - Ian Grigg.
A Ricardian Contract is a legal form (a document) that provides transparency in the case of something wrong happens. Ricardian Contract, combined with smart contracts (a complex coded language), provides a non-programmer client to understand better what happens.

However, **contracts** is in reality neither a *smart contract* nor a *Ricardian Contract* but an agreement between the parties. The document is only a way to record the agreement. A perfect example is that in some parts of the world, a contract can be verbally legal. I. Grigg pointed out that the Ricardian contract is not a contract but the result of efforts to create a document that dominates the contract as found by the court. He proposed a design to couple smart contracts and legal documents in Fig. 3.4.

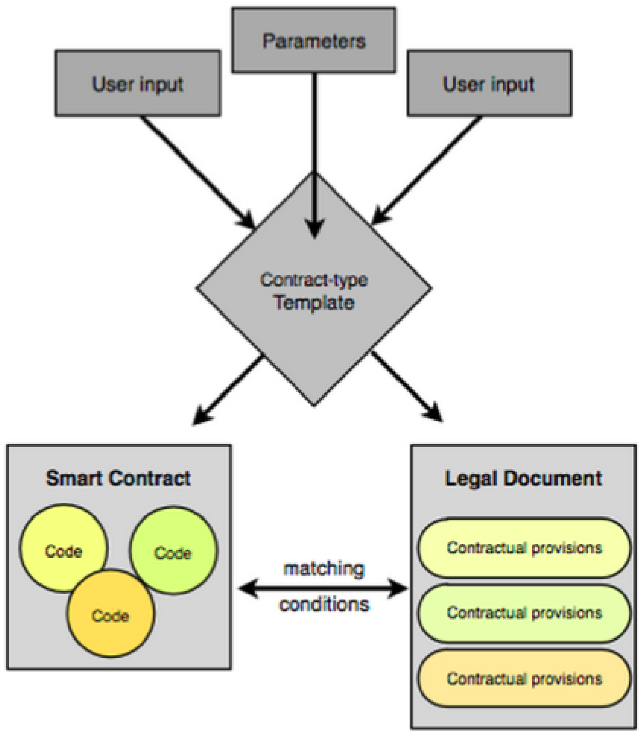


Figure 3.4: Coupling code snippets to clauses and composing upwards

This method has been followed by the EOS.IO developer community to built the Ricardian Template Toolkit [15] in April 2019. We use this framework in our SIM practical work in Sect. 6.4.1.

To write a Ricardian Contract, a developer must know the SC (i.e., what the SC should do). It means the SC developer must be partnered with a lawyer (or legal expert) to build a valid Ricardian Contract. The Ricardian Contract template takes multiple inputs (See Fig. 3.4). In EOS.IO, the Ricardian Contract template, containing text and conditions, takes the transaction as input (containing the SC action data) and produces a formal language text that is understandable by any individual. The results of Sect. 6.4.1 is a demonstration of the Ricardian Contract.

One challenge of the SIM project is the definition of a formal language to specify and verify smart contracts while managing the operations present on the blockchain. Ricardian Contract can answer the verification of SC using a formal language, but it **doesn't define the SC**. Indeed, a SC is a barrier for non-programmer in this transdisciplinary project, and the Ricardian Contract can ease communication and understanding between disciplines.

3.4 Cryptography and blockchain

Cryptography is a pillar to build a blockchain ledger. Blockchain uses cryptography to secure communications between the parties using asymmetric cryptography, signatures and hashes.

Signatures are used to verify the authenticity of blockchain transactions (i.e., the message). Digital signatures are generated using a mathematical scheme. In asymmetric cryptography, the signature is generated using the private key. It is possible to verify the authenticity of the message with its signature and the public key of the individual (corresponding to its private key).

Unlike symmetric cryptography, asymmetric cryptography use two keys (i.e., key pair) to build signature (and encrypt/decrypt data). This allows to create and verify data signatures securely without exchanging the private key.

The standard signature and identification for most blockchains are using Elliptic Curve Cryptography (ECC). There are multiple ECC algorithm to encrypt, decrypt and build digital signatures. To build signatures, blockchain use the Elliptic Curve Digital Signature Algorithm (ECDSA). This signature generation algorithm has multiple parameters. The parameters offer the signature to have a different level of security.

Certicom Research [16] published a paper with a proposal called *Standards for Efficient Cryptography* (SEC), which includes a standard for the ECDSA algorithm. The standard used by almost all blockchain is ECDSA with the **secp256k1** parameters (**SEC P**arameter **256**-bits with **K**oblitz curve) [17].
Using secp256k1 requires the private key is 256 bits (32 bytes), and the generated signature is 512 bits (64 bytes). Depending on the blockchain, the transaction may have different total data sizes (more or less required data field to provide for a valid transaction). We can also notice that all transactions are identified and verified using the public key associated with the private key. The public key size is 512 bits (64 bytes).

Research in [18] successfully build a dedicated hardware accelerator (i.e. ASIC) on FPGA for ECDSA-secp256k1 algorithm. Research specific to cryptography optimization using ASIC is popular but rarely implemented and suitable for blockchain. We discuss more on this subject in Sect. 3.6

By knowing the required transaction fields and the data our use-case needs to send, we can estimate the total transaction's size. This information can be helpful later on to measure the network usage of an IoT device.

Related work on hardware accelerator already exists, and several techniques are developed to have better blockchain peer performances. However, only a handful of studies aims to use blockchain accelerators on IoT devices. This point may be a new research opportunity for future work. It could be an exciting field to have an in-depth study of cryptography on very constrained devices.

3.5 Related work (implementations)

Numerous related works have proposed implementations of DLT associated with IoT devices. The most common practical studies are for energy grid ecosystems, Internet of Medical Things (IoMT) use cases, and Access Management.

3.5.1 Energy grid ecosystem

Yahaya et al. [19] proposed a Home Energy Management (HEM) system and a demurrage mechanism. They aim to reduce energy costs and energy waste following a concept introduced in decentralize peer-to-peer Local Energy Market (LEM). The proposal includes the management of the main energy grid and energy trading: a solution where the local community can use the main grid when the local grid is insufficient to the user's demand. The authors model demonstrate that using their pricing model (implemented on smart contract) combined with the LEM and a scheduling algorithm, it is possible to reduce a lot of electricity cost. The scheduling algorithm used is Critical Peak Price (CPP) and Real-Time Price (RTP) schemes. Table 3.2 summaries the different parameters and cost reduction.

Test #	Using blockchain (P2P pricing model & Demurrage model)	scheduling algorithm	Electricity cost reduction
1	✗	CPP	44.73%
2	✓	✗	65.17%
3	✓	CPP	51.80%
4	✗	RTP	28.55%
5	✓	✗	35.09%
6	✓	RTP	44.37%

Table 3.2: Yahaya et al proposal results

3.5.2 Access management & Healthcare

Medical and healthcare gained interest in blockchain because of the lack of automation, privacy, and security. The features defining DLT are compatible with medical use-cases. This compatibility and need for solutions brought interest.

Ahmed Raza Rajput proposes a perfect example of privacy protection in the medical domain using blockchain: an Emergency Access Control Management System for Personal Health Record [20]. The authors present several papers performing access management on Personal Health Record (PHR) and blockchain access control (and sharing) in general. Usually, patients bring their medical paper-based records to an unfamiliar physician. But there is a problem of emergency access, where the patient is physically incapable to brings his/her record or give access (e.g., in a coma).

The most common personally controlled health records (PCHR) Azure for health[21], Google Health[22], Indivo[23] store patient user's health records on central networked servers or systems outside the medical system. Thus multiple frameworks propose a solution to store and manage personal data access.

The authors proposed and implement a system solving issues related to the current state of personal heath records during emergencies. They use Hyperledger Fabric blockchain and Hyperledger Composer tools to create their framework. The business logic for the smart contract includes Patients, Doctors, and Emergency Doctors. Each participants in the smart contracts have their permissions and connects to the framework using an API (See Fig.3.5).

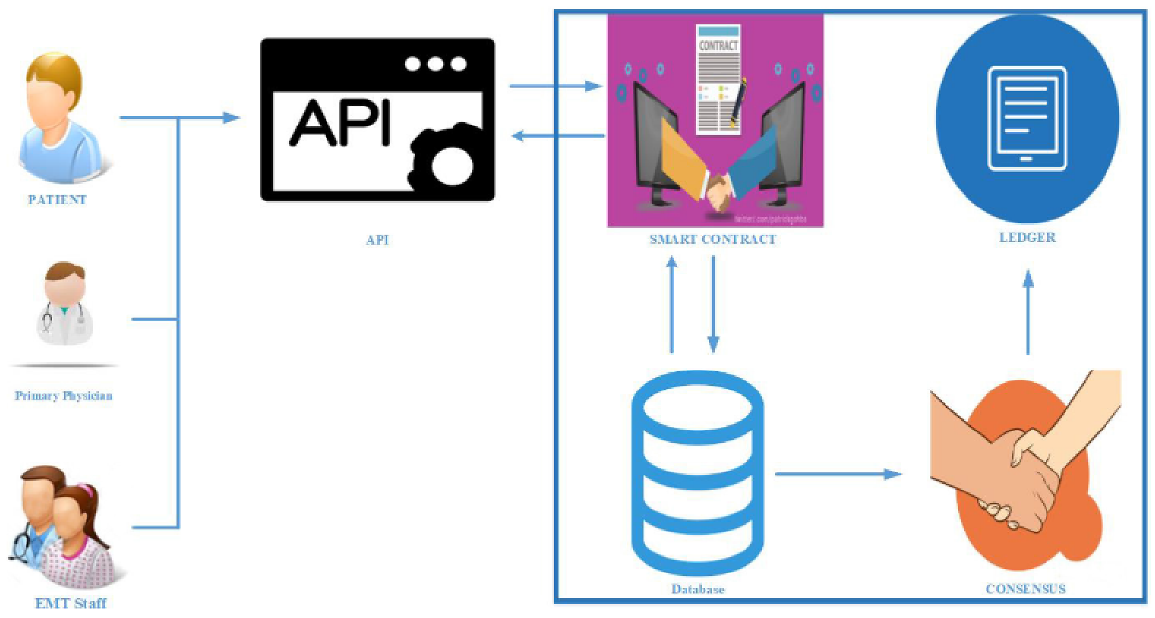


Figure 3.5: Proposed architecture for emergency access for PHR [20]

The authors implement in the smart contract multiple algorithms to register the participants, retrieve information form patients securely, and store patient data. Also after evaluating efficiency, they conclude that the framework is secure, provides privacy for the patient's data, and guarantees efficient access time in case of an emergency. This implementation prove it is possible to improve existing medical procedures with an adequate blockchain implementation.

The SIM project is quite similar on the general point of view of the implementation on the server side of the implementation (off-chain solution with clients that does not contain the blockchain node software and data). However in SIM project we have an IoT with limited resources that has to communicate with the blockchain.

The following related work shows use-case using blockchain and system that doesn't need an IoT device at all (only unique identifiers).

3.5.3 Supply chain

Supply chain in manufacturing systems involves many entities, including people, physical resources, knowledge, processes, and financial contracts. It results in multiple transactions between entities, intending to trade a product from supplier to customer. In a large supply chain system, it is very difficult to have an overall picture of all transactions within the chains.

All entities have their system. Thus, information is typically stored in multiple locations, and some system has permission to access other systems. In such systems, the customers (being the final consumer) usually have partial access to the overall information. In many cases, the supplier does not require to give the information to third parties. Therefore, due to the low level of transparency, transactions' tractability is often based on the trust between the system actors.

DLT can improve transparency and traceability issues within the manufacturing supply chain through immutable records of data, distributed ledger, and controlled user access.

The paper [24] proposes a system to collect, store, and manage key product information of each product throughout its life cycle using blockchain. The authors propose a solution where actors can access and register themselves in the network and how data entry is authorized, validated, and stored. The proposed implementation is represented in Fig.3.6

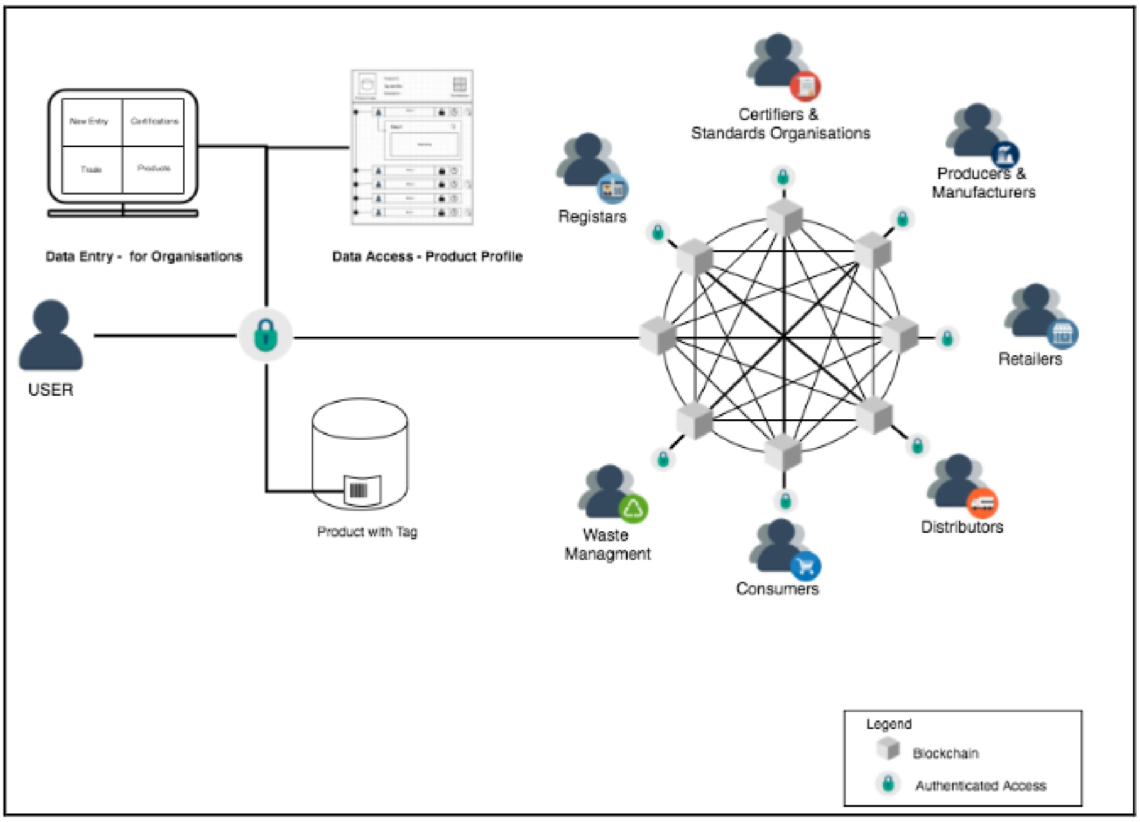


Figure 3.6: Blockchain ready manufacturing supply chain using distributed ledger: Overview of the proposed concept

They propose a solution where the product only has to have a QR-code or RFID (aka a tag) representing a unique digital cryptographic identifier that links the physical product to its virtual identity on the network. Using this method the supply chain has only to embed an unique identifier on the products. In the solution, participants could access or enter data depending on their role (retailer, consumer, etc.).

The authors conclude their proposal by highlighting the advantages: facilitates data collection, traceability, automation. However, such a solution would require a particular IT infrastructure for all actors to participate in the system. They also explain that such a system should make use of smart contracts to automate payments. The SIM project use-case is not similar to a supply chain because it doesn't use IoT devices that interact with the blockchain. However, in Sect.3.6 we present papers that involve RFID that uses hardware accelerators to enable blockchain specific applications to be embedded on the device (cryptographic function implementations). This paper also highlights the interest of blockchain and smart contracts with a consumer's point of view. The actors that will interact with a blockchain system can come from totally different origins. This point of view is often forgotten, but the SIM project also has this project's multidisciplinary side.

3.6 Hardware accelerators

To understand the current state of hardware accelerators used in blockchain we can use Bitcoin as an example. In Bitcoin, minors are the nodes that verify and maintain the consensus. Simply put, the consensus (PoW) is a race to resolve a mathematical problem first (with a degree of difficulty). Winning nodes are rewarded each time the algorithm is solved first.

This race has lead minors up to optimizing the blockchain software using Application-Specific Integrated Circuits (ASIC). When building an ASIC, we can distinguish two goals:

- optimizing speed
- optimizing energy consumption

These ASICs are not useful for an IoT device that doesn't take part in the consensus like PoW. But it has built an interest in hardware accelerators specific for blockchain application.

As we have seen in Section 3.4, signatures are performed with the ECDSA algorithm. This algorithm has been the subject of several studies. The authors in [25] have made a survey about the state of ECDSA algorithm and its application. Two interesting applications that relate to the SIM project are Wireless Sensor Network (WSN) and Radio Frequency Identifier (RFID).

In WSN, they compared the cost of ECDSA and RSA algorithms. RSA (from the inventors **R**ivest–**S**hamir–**A**dleman) is an encryption algorithm (and also has a signing algorithm) using a private-public key (asymmetric cryptography). According to the paper, compared to ECDSA, RSA's algorithm:

- calculations are less computationally hungry
- needs a longer key (3072 bits in RSA is equivalent to 283 bits key in ECDSA) [26]

The authors point out that the same security is achieved with a smaller key size using ECDSA is an advantage. Ten times shorter key size for constrained-source devices means less storage, memory, and computation. Also, authors mention that ECDSA have functional applications on small 8bits microcontroller.

With RFID applications, the authors cite numerous research where ECDSA is the most adapted. One reason is that this algorithm doesn't require storing private/public keys like symmetric cryptography algorithms. Authors state multiple modifications of the ECDSA algorithm (e.g. optimizing the hashing function) and implementing hardware acceleration on RFID devices, decreasing the number of clock cycles and thus energy consumption.

This concludes that hardware accelerators can be used in IoT devices to build blockchain type transactions and reduce consumption of the device. This has also been proven with simulation in [27] and in our paper [28].

Chapter 4

Studied blockchains

This chapter describes the general structure of blockchain-client, and we study three blockchain to reveal their differences.

4.1 Whats is a blockchain node ?

As explained in previous chapter, the clients are off-chain, thus the blockchain nodes are installed on servers. This internship doesn't require to have a in-depth study of this part of the system but it is the heart of the blockchain. Independent of the consensus algorithm, a node - in general - is in charge of processing the incoming transactions of the blockchain. As we have seen, a transaction is processed in multiple steps, commonly:

- transaction sanity check,
- de-serialization,
- transaction execution,
- finally adding the transaction to a block,
- broadcast the block.

A node handles transactions interacting with smart contracts. It schedules the smart contract execution.

The order of execution is important, so all the steps execution must be sequential. Also, we want to have a transaction execution priority. Depending on the blockchain structure, transactions are handled in a transaction pool (priority can be a FIFO, priority to the transaction with most commissions, etc.). It is good to notice that the distributed ledger's goal is to store records sequentially to have a chronological history of the transactions.

4.2 What is a blockchain client ?

4.2.1 Introduction to a blockchain client

The client is responsible of sending transaction to a node. To be able to send a valid transaction, the client requires different information about the blockchain node and about the application. A client also have to do processes, such as :

- Initialize and build the transaction based on the application
- Sign the transaction according to the blockchain standard
- Send the transaction to a node (API)

This process is fairly simple but can be memory demanding or CPU intensive depending on the client resources.

In our case, with an IoT device, the priority is to optimize energy consumption and achieve an efficient execution time. Also, we already have specified that our IoT device will not include a full node (it will be an off-chain node, depicted in Fig.4.1). As we see in Fig. 4.1, the client (IoT) doesn't contain:

- the copy of the blockchain records
- the consensus
- node API endpoint server

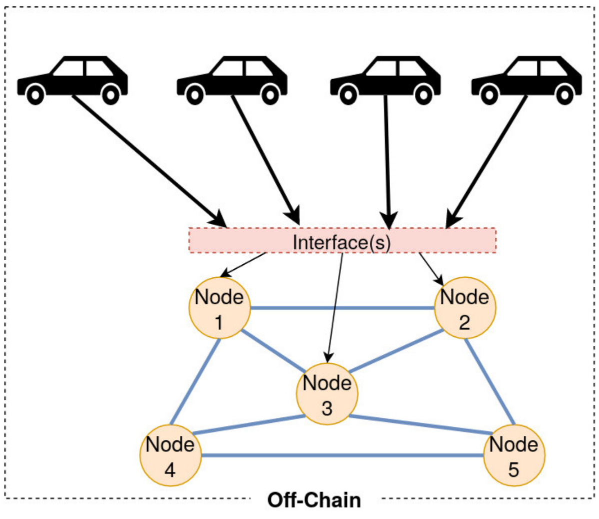


Figure 4.1: Blockchains off-chain solution

It implies that IoT hardware should be focused on improving the client application . This optimization can be archived with ASICs that handles one or more of the blockchain processes in the list above.

4.2.2 Client on an IoT

Firstly, you can observe that the client requires to retrieve and send data to a node, meaning the client needs to have a decent connection to avoid any data loss or failed transaction.

Secondly, the initialization, building and signing the transaction can be CPU and/or memory intensive depending on the application. An everyday computer with GB of storage and memory, multiple GHz of CPU power and unlimited energy could do that in a simple task. On the other hand, for an IoT device with very little storage (or none), memory, computational power and energy it is a true challenge.

As an example presented in our previous work [29], we want to send a hash of all the data retrieved from the car along with the car identity. If we want to have an efficient blockchain client, the board interfacing the blockchain needs to be as low-power as possible (lets say around 100mA-500mA). This defines the resources of the board.

As a base test board, in the entire internship, we used a Raspberry Pi 3B+. This board architecture is composed of : 64GB of storage (Micro SD), a Broadcom BCM2837 (Cortex-A53 64-bit SoC @1.4GHz) and 1GB LPDDR2 SDRAM. With this architecture it is possible to run an entire Linux OS, using on IDLE 500mA and 1A fully loaded. This is of course an excessive amount of hardware resources for the use case, but it allows us to have a base architecture for all tests on different blockchain.

4.3 Sawtooth

4.3.1 Description & Target

Hyperledger Sawtooth is an enterprise blockchain platform, thus it is a permissioned (private) blockchain. Sawtooth was designed to achieve a secured, scalable and modular structure.

Its structure is modular because it contains modular consensus that means that Sawtooth is able to use different type of consensus as PoW, PBFT, and PoET (Proof of Elapsed Time). In addition to the consensus, Sawtooth is modular because, it is possible to implement own modules in a node interacting with the Validator (Sect. 4.3.2). A big advantage of Sawtooth is that the consensus can be changed on running time.

4.3.2 Structure & definitions

Validator : It is the core of the Hyperledger Sawtooth blockchain. It handles blocks, the validation of transactions, the peer to peer connections, the blockchain state and maintain consensus between peers.

Transaction processor (TP) : It is analogous to smart contracts. It is a module added next to the Validator, that handles a specific type of transaction family. A TP can only execute one type of transaction family. These modules can be added in parallel to enable faster execution.

Consensus engine : It is the module that contains the consensus algorithm for the Validator. In Sawtooth, the consensus can be changed dynamically.
REST API : It is a module that communicates with the Validator using the interconnect (ZeroMQ protocol). It is an HTTP interface for the blockchain and thus clients.
Client : It is our IoT device. As depicted in Fig. 4.2, the client does not has the software to participate in the blockchains, the client are thus off-chain.

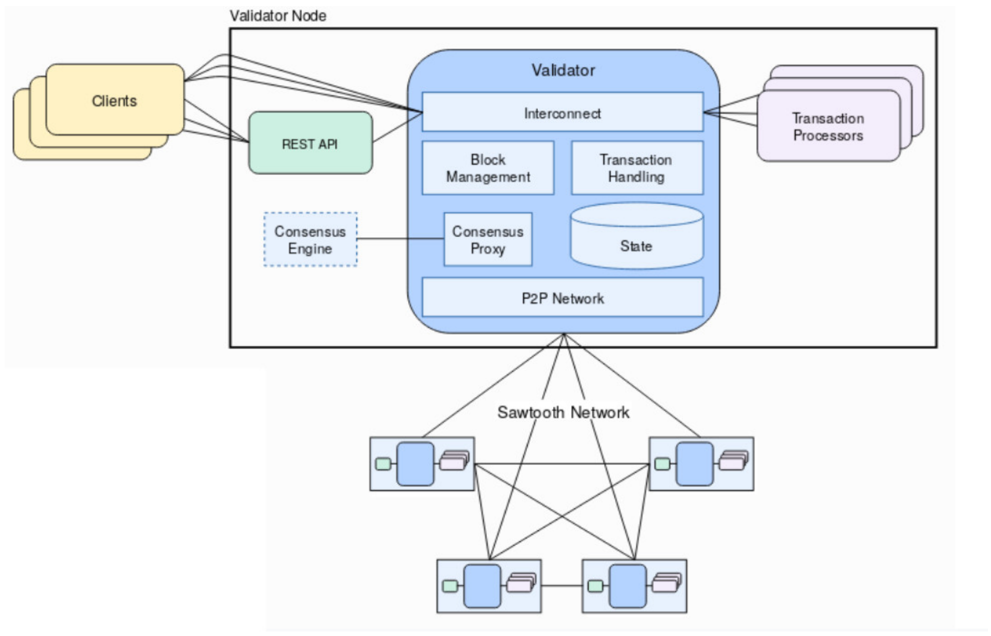


Figure 4.2: Hyperledger Sawtooth structure

Hyperledger Sawtooth has gained interest due to the implementation of PoET compatible with the Intel Software Guard Extensions (SGX) processor. SGX allows Sawtooth's code to run in a protected and trusted execution environment.

SGX is the solution to PoET consensus's central issues: safety and randomness of the leader election process. The consensus needs SGX to create an enclave capable of ensuring safety and randomness, preventing PoW like algorithm usage.

4.3.3 Related work

Author in [30], are analysing the blockchain using Caliper benchmarking tool to test the performance and identify the potential issues. The authors provide the key performance metrics that shows that Hyperledger Sawtooth has:

- An impact of input transaction rate on throughput
- An impact of batch size on throughput
- An impact of input transaction rate on memory usage
- An impact of Throughput on Latency

For the SIM project, related works are limited. Two papers describe similar work using a vehicle use case.

Authors in [31] proposed and implement a Smart Service Book using smart contracts to digitize the vehicle life-cycles. They highlight the features of a blockchain based framework such as transparency and collaboration between involved organizations. The paper describes part decentralized system, where some part of the implementation is centralized.

Authors in [32], built a vehicle framework for forensics (Block4Forensic). They aim to connect vehicles for determining vulnerable parties of a forensic situation. Data is recorded on a Event Data Recorder device, placed in the vehicle. In the implementation Block4Forensic the framework is linked to multiple organizations such as law enforcement, insurance and manufacturer. They propose and implement a solution where the car data is stored in a central system and the blockchain stores data hashes. This paper provide the most related work to the SIM project.
Both papers describe an ecosystem but lacks scalability analysis of their proposals and implementations.

4.3.4 Security

Because Hyperledger Sawtooth is an enterprise blockchain, security depends a lot on the integration and installation.

To have a secure Sawtooth network and like any real-life scenario, it is necessary to add multiple systems on top of the blockchain. Typically, these systems include network layers such as a load balancers, DevOps container-orchestration system, and additional blockchain tools like peer management, etc. These tools and additional software are common to almost all blockchains for real-life installation and won't discuss it in this internship because it is out of scope.

However, we explain security threats involving the blockchain core ideas and system to create a full point of view of the weaknesses and strengths of this technology.

4.4 IOTA

4.4.1 Description & Target

IOTA is one of a kind distributed ledger with a its own cryptocurrency. IOTA relies on a specific structure called the Tangle. Multiple software layers are stacked together to form a system that allows easy transaction processing for embedded systems.
From the start IOTA is designed for embedded systems. The goal is to allow small devices with constrained resources to perform secure transactions on a DLT.

4.4.2 Definitions and structure

The Tangle [33] is the nickname describing IOTA directed acyclic graph (DAG [34]) transaction settlement and data integrity layer. It is the first layer designing the structure that holds IOTA system together. The Tangle algorithm is based on the reputation of nodes: a transaction is validated by validating two previous transaction of two different nodes.

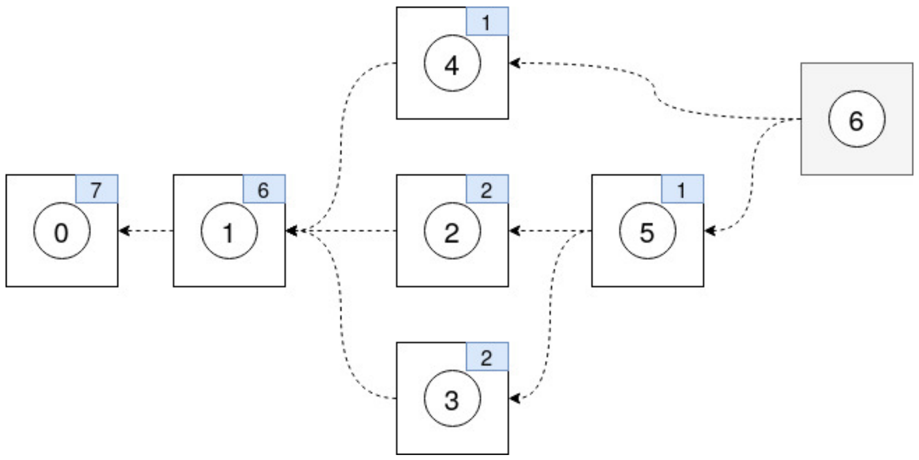


Figure 4.3: Transactions using Directed acyclic graph (DAG)

The DAG is composed of *edges* and *vertices* (Fig. 4.3 shows an example of DAG). The edges are the confirmed links between approved transaction (arrows in Fig. 4.3). The vertices are the approved transactions (squares in Fig. 4.3). The transaction 5 approved transaction 2 and 3. Transaction 6 is a *tip* because no other transaction has approved it yet.

Tip selection (of the two different edge) follows an algorithm that isn't enforced by the network. Tip selection process is a random walk of a subtangle of the ledger and consist of going through the edges with the highest (reputation) rating called : cumulative weight.
Cumulative weight of a transaction is as the own weight of a particular transaction plus the sum of own weights of all transactions that directly or indirectly approve this transaction. In Fig. 4.3, cumulative weight are represented in a blue square in the top-right corner of each edges of the DAG.
Consensus in IOTA is maintained using the Tangle. But it is required also to coordinate the Tangle to "freeze and approve" the state of the DAG transactions at a periodic time. In IOTA the Tangle has the following entities:

- "The Coordinator" (2015- 2021): a central application, run by IOTA Foundation that emits bundles that reference and approve two new random transactions in the ledger. The signed tail transaction in the bundle is called a milestone.

- "Coordicide" (2021-future): a new consensus that won't need the centralised Coordinator. It will be a novel consensus named "Fast Probabilistic Consensus" (FPC).

Practical structure of IOTA: The main entity is called *IRI*. This software is installed on a server (i.e. node). It handles the communication between other IRI nodes and clients. Clients can exchange transaction using the *HTTP API* and can listen to events of the tangle using the *Event API* (depicted in Fig. 4.4).

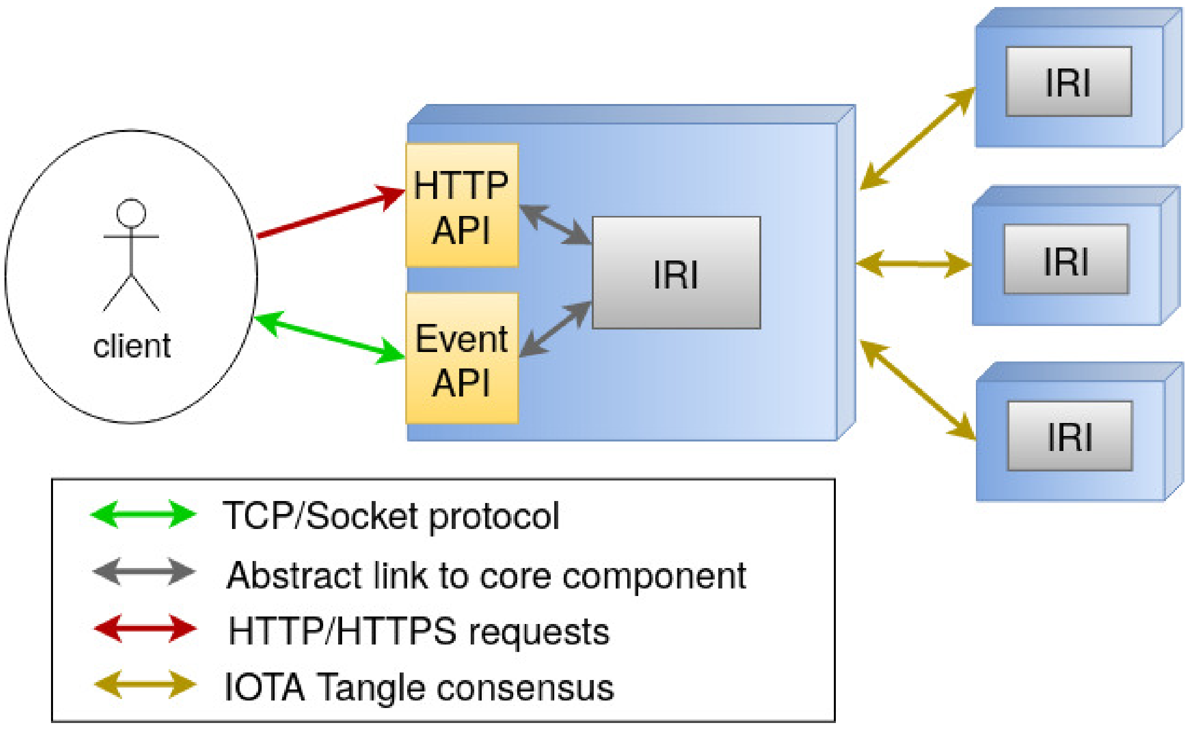


Figure 4.4: Structure of IOTA: Node and client

It is also possible to find other implementation of IRI, such as *Hornet*. It has code improvements and additional tools that helps visualize and control your node (easy configuration, dashboard to visualize peers, explorer, etc.).

4.4.3 Current state of IOTA

The current state of IOTA is complex. Multiple proofs of concepts have been made along with a theoretical study on the structure and consensus. Because of the Tangle's particularity and its need to have enough activity (participants and functional nodes), it was necessary to have a centralized entity ("The Coordinator") to jump-start the blockchain.

In the year 2021, IOTA will have enough participants and functional nodes. IOTA will have a stable, tested version of "Coordicide". Hopefully, this will be followed by the implementation of new layers and software (included in "GoShimmer" prototype project) that will allow the execution of SCs.

4.4.4 IOTA in the context of SIM

SIM project aims to enable SC on IoT devices for a specific use case. However, IOTA's only development on SC is on the "Qubic" project. This is a PoC project built in 2018 has not been maintained since then. It has proven that IOTA can implement SC. Nowadays this project isn't compatible with the current version of IOTA, thus it is impossible to deploy smart contracts on this blockchain.

Thanks to the PoC, the developers have extracted all essential ideas and are now developing an improved SC implementation (very early test module in a "GoShimmer" project). More about this PoC in Sect. 5.1.

We can notice that IOTA is a system whose very core may be very different from more common blockchains. However, it has similar functionality (transactions, a cryptocurrency, and smart contracts in the future).

4.4.5 Related work

In our context, IOTA has two related works:

- Hardware accelerators for Curl hash function and proof-of-work [35]: Authors present hardware accelerators for IOTA cryptocurrency. They implemented their solution on an FPGA and integrated into ARM system-on-chip. They compared the results with official implementations and the authors demonstrate x2100 speed up on the creation of IOTA transactions.

- Data Marketplace: an PoC that allows to exchange, buy and sell data. The coming from sensors can be sold through a marketplace that uses IOTA to handle secure access management. Once data is bough, a user can retrieve secure data streams that are cryptographically verified.

4.4.6 Security

Several security threats have been discovered or are suspected. Without going into details, the following concerns have been exposed:

- IOTA hashing function called Curl-P-27 has flaws: in paper [36], the author present concerns that the main IOTA hash function isn't entirely secure.

- Parasite Chain [37]: is when the Tangle is being corrupted due to a parasite chain. The parasite chain gains reputation by auto verifying its own transactions and gaining more and more influence on the main Tangle chain.

4.5 EOS.IO

4.5.1 Description & Target

EOS.IO appeared in 2017. It is a blockchain that theoretically claims to handle millions of transactions per second. This makes it one of the first blockchain solution fit for industry-leading transaction speed. It targets decentralized applications. Its ecosystem also uses a new kind of model where there is no transaction fee but participants can buy memory (RAM), stake CPU and stake network bandwidth resources.

4.5.2 Structure & definitions

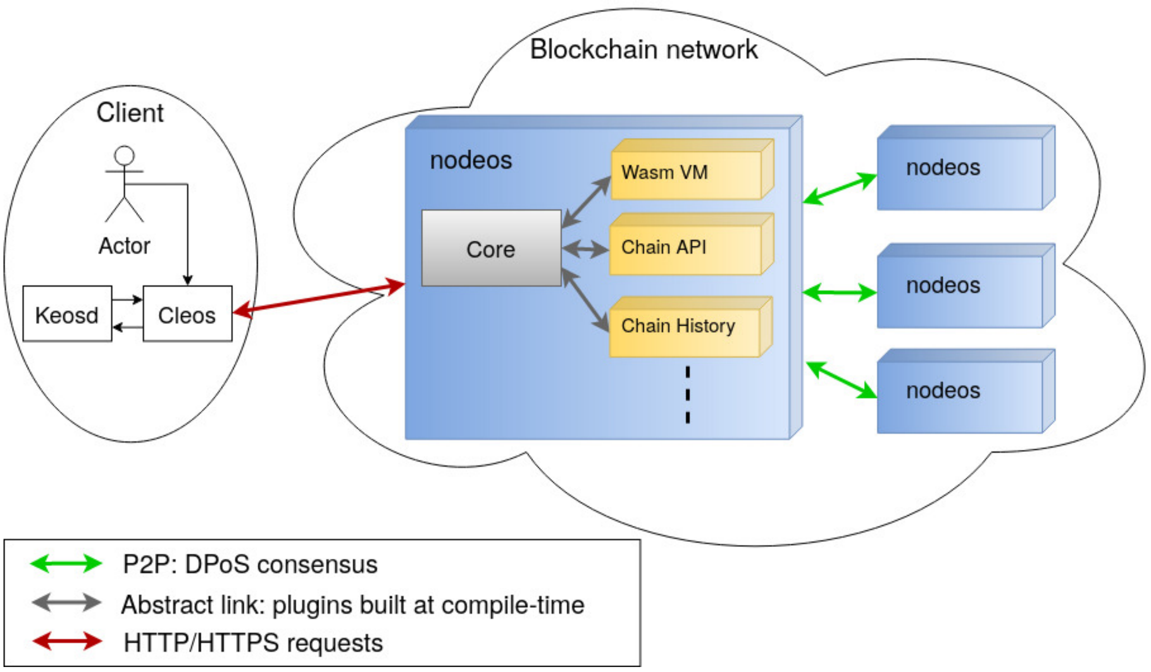


Figure 4.5: EOS.IO general blockchain structure

Nodeos is the core software of EOS.IO blockchain node. Depending on its configuration it can handle smart contracts, validate transactions, and produce and confirm blocks.

Around *Nodeos* are **plugins**. Some plugins are mandatory, such as chain_plugin (process and aggregate chain data on the node), net_plugin (provides an authenticated P2P

protocol to synchronize nodes persistently), and producer_plugin (loads functionality required for a node to produce blocks).
Plugins allow the EOS nodes to be modular. Unlike Hyperledger Sawtooth, EOS nodes plugins are compiled at the same time than the core (i.e., Nodeos). Partitioning software features allows building different blockchain nodes for different purposes. There are two main node setups:

- non-producing node: is a node that is not configured to produce blocks. However, it is connected and synchronized with other peers from an EOS.IO blockchain, exposing one or more services publicly or privately by enabling one or more Nodeos Plugins, except the *producer_plugin*.
- producing node: as its name implies, is a node that is configured to produce blocks in an EOSIO blockchain. This functionality if provided through the *producer_plugin* as well as other Nodeos Plugins.

⚠ An important notice: a block producer has an essential role. It has:

- Consensus: if a block producer is voted, it becomes an entity that participate in the consensus
- Infrastructure and costs: Server upgrades and maintenance to meet the EOS network needs. Also, a block producer takes part in decisions during inflation (increasing number of EOS tokens in the blockchain). EOS.IO uses inflation to fund and control the ecosystem.
- Community Building: Community building and coordination to empower the network. A block producer can use IPFS, a distributed file system (See Sect.6.3.2 for details on IPFS), to store dApp, meaning they have to handle the storage of such system. It makes the community (participant, smart contract developers, etc.) more active and dynamic.
- Product Development: Provide helpful tools for EOS development and usage.

Next, if the goal is to search the blockchain records and data, association of *state_history_plugin* and a plugin called *fill-pg* will allow Nodeos to fill a PostgreSQL database. This database can be searched, queried for particular purposes, and analyzed. This is used for applications and decentralized applications (dApp) that requires to display historical data (as an example, your account balance history).

Decentralized applications (dApp) are applications that use the blockchain to store data, manage identity, automate business logic, etc. A dApp in EOS.IO can be executed from one or multiple platforms and use the blockchain chain API to exchange data and transactions.
In this internship, we built a fully operational dApp for the SIM project as a PoC. As a first step, we aimed to manage the client identify and store arbitrary information (e.g. text, a hash, bytes or anything else) on the blockchain. Details about the implementation is in Sect.6.4.1.

EOS.IO blockchain also has a hierarchical account model, where only an existing account can create an account (i.e., a tree structure). In other words, only an existing account can allocate resources to create a new account.

⚠ It is possible to associate permissions on accounts defining the authorized actions and transactions to other accounts. The actions can be combined with a specific private/public keys pair, limiting the action to be only valid with the defined key pair. In this internship, we used the same key pair for all actions of the accounts. However, in the future, if we implement the SIM project using EOS.IO, the account and permissions will be very useful to create a sophisticated model for identification and transaction execution.

4.5.3 Related work

EOS.IO is currently just two years old. The usage of EOS.IO is primarily for its currency (the "EOS") even if the goal is for the blockchain to allow easy dApp development and implementation. Study and results (on one year data) showed that dApps are being developed but not much used. Also recorded activity on the blockchain shows that the number of active accounts is less than 1% but counts for 90% of transacted funds.[38]

Recent evolution of EOS.IO shows interest in the technology but doesn't involves a lot of use case research or implemented proposals. This is why it is difficult to explore related work of EOS.IO to the SIM project.

Nonetheless, this blockchain has very high potential for our SIM project use case and should be studied. Our implementation in Sect.6.4.1 features multiple very exciting results.

4.5.4 Security

Like any other blockchain, EOS.IO has suffered major security threats that lead to loss of account cryptocurrency. In EOS, attackers used the blockchain resource management to create blockchain Denial of Service (DoS). Denial of Service is when a system is unavailable due to a cyber-attack, in EOS.IO blockchain it will result to blockchain nodes not responding or running properly.

In the paper [39], the authors present two types of DoS by draining EOS resources:

- *SCP* (Smart Contract Provider) *CPU-drain attack*: consuming huge amount of SCP CPU time using only small EOS CPU amounts.
- *SCP RAM-drain attack*: demonstrating a method to consume huge amount / all of SCP RAM. Resulting in a blockage of the SC, and in worst case scenario a required modification of the SC by SCP resulting in data loss.

The authors also proposed and implemented a *RAMsomware attack*, which can fill the RAM a user can occupy to the maximum (2GB). This can then block the user's activity due to the reached limit.

Paper [38] is a in-depth study about EOS.IO blockchain characteristics. The authors analysed all the blockchain data from 2018-06 to 2019-04 by aggregating account creation, account usage, smart contract usage, number of nodes, transfer frequency and quantify, and more. After analysing the blockchain data, they have the following conclusions on EOS.IO:

- Overall increase of transactions (over 1 billion transfers). EOSIO is dominated by a small percentage of accounts. The top 0.47% of accounts constitute 90% of the total transaction volume.
- Bot-like accounts are dominant. Over 30.75% of the accounts (381,008) as bot-like, with over 192 million transactions, and 640 million EOS transferred. These bots are mainly used for malicious and fraudulent purposes including Bonus Hunting, Clicking Fraud, etc.
- Permission misuse issues are overlooked by users. Some account granting their "eosio.code" permissions to other accounts, which could cause serious security issues.
- EOSIO suffers from a number of serious attacks: over 301 suspicious attack accounts, causing over 1.5 million EOS losses. Also multiple fraudulent dApp.

Chapter 5

Smart Contracts

5.1 IOTA

As explained in Section 4.4.3 this blockchain only has the "Qubic" PoC that implements SCs.
The "Qubic" PoC adds two entities to IOTA:

- Qubics: analogous to SC, code that has inputs and outputs a result
- Oracle Machines: are the machines that executes the Qubic code

In a simple way the protocol to use IOTA-Qubic smart contracts works as follows:

1. Send a normal IOTA transaction containing special data (for the smart contract)
2. The Oracle machine detects the transaction with the smart contract (Qubic) input
3. The Oracles machines associated to the Qubic code resolve the Quorum
4. If Quorum is accepted and complete, the Oracle machine creates transactions in IOTA containing the Qubic results allowing the Qubic owner to consume it

Independently to the Tangle, the consensus reached for the SC to be validated is done by a minimum number of agreed Oracles: call the Quorum. In Fig.5.1, you can observe that Qubic is a layer on top of IOTA. Unlike Ethereum, Qubic is not a true on-chain smart contract solution because the blockchain doesn't handle the smart contract, it is rather a separate network with a separate consensus.

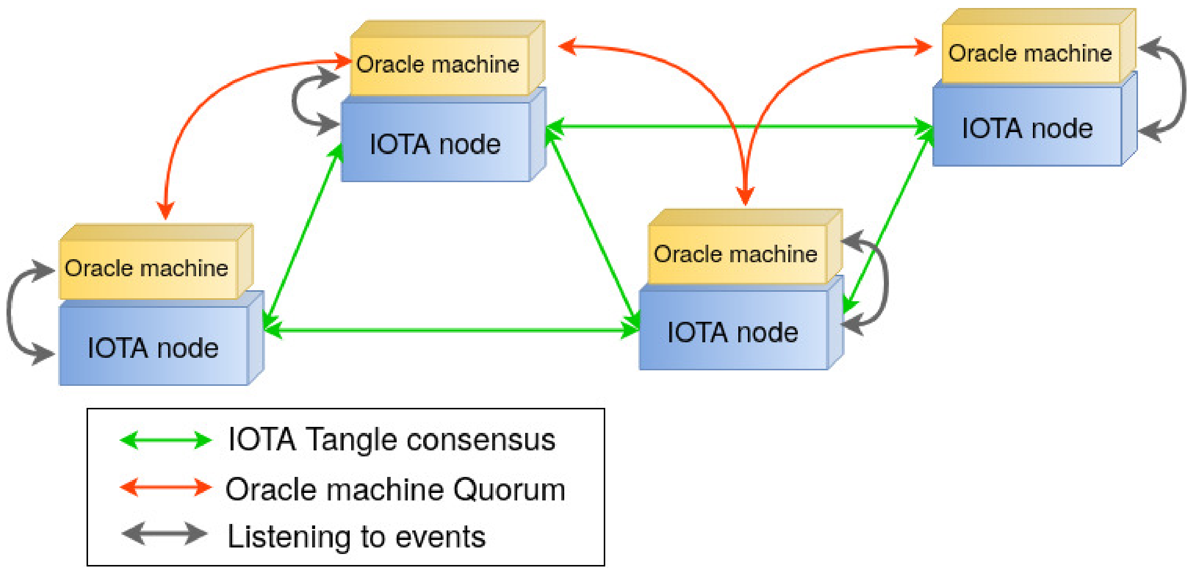


Figure 5.1: Qubic structure in IOTA

The Qubic code is programmed in Abra (a trinary-based language). To generate the compiled Abra tricode, developer uses the Qupla (QUbic Programming LAnguage) language-compiler [40].
As explained by E. Hop in article [41], a member and active developer at IOTA, the Abra language is not entirely specified and complete. The Quapla interpreter and compiler is still in development to implement the Abra specification. E. Hop also aims to use Abra because this language can be implemented easily in hardware.
Several problems can be encountered with this type of SCs:

- The SC consensus, precisely the Quorum. Depending on the expected security of a user, if the number of Oracles machines for a Qubic is too low it can be considerated that the Quorum is ”too” centralized.
- The number of transactions created by the Oracle machines is significantly high. It saturates the IOTA network.
- Previous results of Qubic implementation in 2018 showed that the structure is not practically scalable.

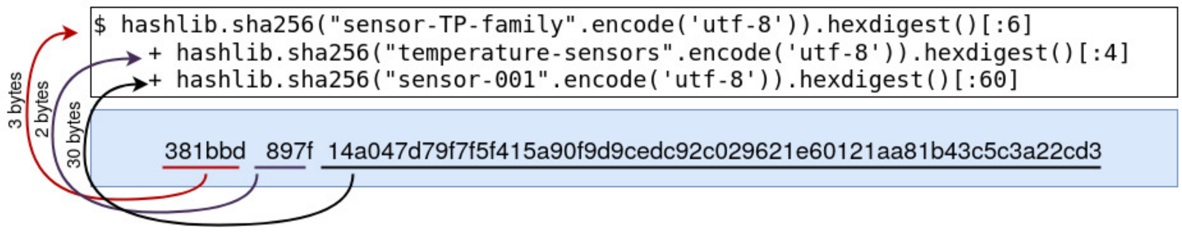
To this day, the Qubic project has not been maintained enough to be usable on the main IOTA network. This leads us not to use IOTA with Qubic for the SIM project. Thus, there are no results and benchmarks for IOTA and IOTA smart contracts.

5.2 Sawtooth

Sawtooth transaction processor (TP) are the smart contracts for this blockchain. The TP is an installed module of the Validator node. Each TP handles one type of transaction family. This module communicates with the Validator using a TCP connection and ZeroMQ messaging mechanism. ZeroMQ has a perfect messaging protocol for distributed or concurrent applications. After the transaction has been verified in the Validator (sanity check, deserialization, signature), it is passed to the corresponding transaction processor with the correct transaction family. The transaction family is analogous to the SC type, for example:

- *settings* TP: Used to define the blockchain configuration of the network
- *intkey* TP: Is a TP to set a key-value inside of the blockchain state. Mostly used for development and testing purposes.

The transaction and TP have their own namespace inside of the blockchain. The namespace defines addresses where the TP can read/write data in the blockchain state. The TP has its own global namespace, and each transaction can read/write data to addresses based on the namespace. In the TP it is possible to add additional checks to limit access for each executed transactions. The namespaces are built from the SHA-256 hash of the transaction family. The addresses are the result of the concatenation of namespace and SHA-256 hash of a text describing the address (See Fig.5.2).



```
$ hashlib.sha256("sensor-TP-family".encode('utf-8')).hexdigest()[:6]
    + hashlib.sha256("temperature-sensors".encode('utf-8')).hexdigest()[:4]
    + hashlib.sha256("sensor-001".encode('utf-8')).hexdigest()[:60]
```

3 bytes
2 bytes
30 bytes
381bbd
897f
14a047d79f7f5f415a90f9d9cedc92c029621e60121aa81b43c5c3a22cd3

Figure 5.2: Hyperledger Sawtooth Transaction Processor namespace and addresses

This structure allows the Validator to have multiple TPs connected to the Validator, and provides parallelism for executing transactions (Fig.5.3).

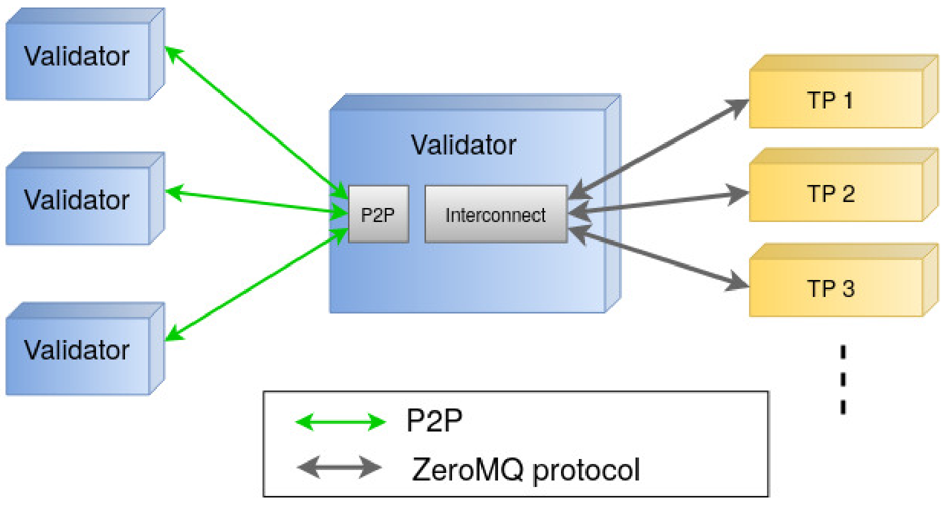


Figure 5.3: Hyperledger Sawtooth Transaction Processor

Thus, Sawtooth SC structure is entirely different from other blockchain. Ethereum and EOS can deploy smart contract on-chain, without modifying the node software configuration. With Sawtooth TP it is required to deploy the TP program directly on the node.

The TP contains event listeners for specific messages of the Validator. The main event is *apply*, where the Validator wants to apply the transaction to the TP. When *apply* is called the TP will execute the implemented business logic. For example in paper [29] the business logic contained a blockchain state check to verify if the incoming transaction in the TP is authorized to store data on the blockchain (represented in Fig. 5.4). In Fig.5.4, each transaction (1) and (2) are required beforehand to execute transactions (3) and (4).

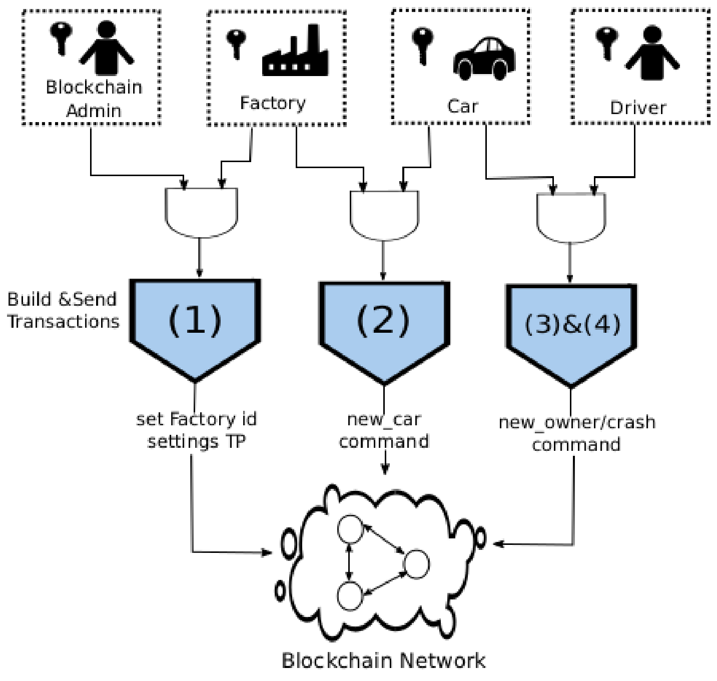


Figure 5.4: Hyperledger Sawtooth Transaction Processor permission business logic

5.3 EOS.IO

EOS.IO [42] designed similar smart contract to Ethereum using a smart contract layer inside the blockchain (on-chain smart contract) core that execute the contract in a virtual machine (VM). SC are written in C++ and compiled in standard WebAssembly (Wasm). EOS.IO smart contracts are executed in the Wasm VM. Visual representation can be found in Fig.5.5.
The EOS.IO EOS Wasm VM characteristics are:

- Deterministic execution
- Time bound execution
- Secure by design
- High performance execution
- Effortless Integration

- Highly Extensible Design

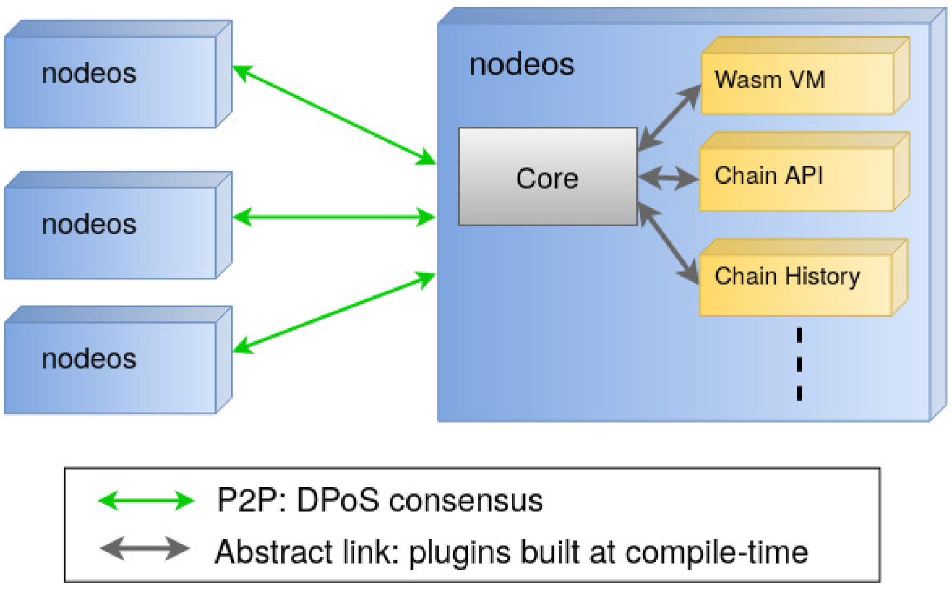


Figure 5.5: EOS.IO smart contract & Wasm VM

These advantages originate from the fact that Wasm was designed to have extremely fast execution speed in-browser applications.
We have to notice that in EOS.IO a Wasm SC file is always associated with an Application Binary Interface (ABI) file (Fig.5.6). This file is used to interface with the Wasm binary and provide important information that describes how to convert user actions (JSON to Binary) and how to convert database state (JSON ⟷Binary).

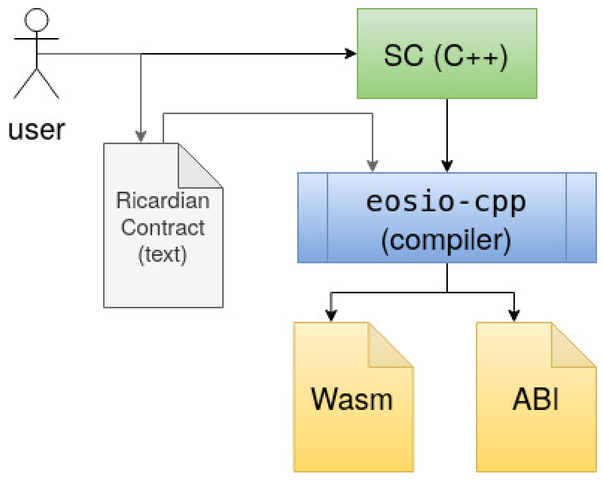


Figure 5.6: EOS.IO SC compiler: Wasm & ABI file

Thanks to the EOS.IO community, an implementation of Ricardian Contract has been added. The Ricardian Contract is written in a plain text file using a template specification to include variables and conditions in the contract document. It is in the ABI file that the Ricardian Contract template is stored.

⚠ An important notice is that ABI can be bypassed when executing transactions. Messages and actions passed to a contract do not have to conform to the ABI. The ABI is a guide, not a gatekeeper.
From this notice, we deduce that when displaying the compiled Ricardian Contract template to a user, the data displayed is **for information purposes only**. <u>It does not take part of the agreement coded in the SC</u>.

Chapter 6

Results

6.1 From theory to practice

The client-side application executes on an IoT device. Thus we need to match the IoT architecture with all required features of the blockchain application.

The first step is to build the blockchain client application. We choose C++: a low-level language with high-level features. After code profiling with a call-graph, it allows us to have a good understanding of the underlying process to send a valid transaction to a blockchain node.

In the following results, with the blockchain Hyperledger Sawtooth and EOS.IO, we had to build our C++ client libraries because none existed before the internship.
The programs and libraries are publicly available online on https://github.com/lucgerrits.

6.2 Code profiling

It is with code profiling that we will find out the most used function on the programs. We use *Valgrind* combined with the *callgrind* tool. These are analysis tools that support multiple architectures: X86/Linux, AMD64/Linux, ARM/Linux, ARM64/Linux, ARM/Android, and others. Thanks to the Valgrind version-3.16.1 (released on 22 June 2020), all the latest architectures are fully supported.

Next, *callgrind* results are visualized with *Kcachegrind*: A data visualization tool that can create a map with each function's program cost (relative to CPU usage). Thanks to the option "relative to parent" it is possible to view the usage of functions relative to the parent function. This functionality improves the visualization because our programs always have unrelated functions to the main function used to load libraries, libraries links, and handle memory.

An example of Kcachegrind visualization after a valgrind callgrind code profiling is represented on Fig.6.1. As you can see, the example shows a *hello_task()* function that is the only function present in the source code in Code 6.1.

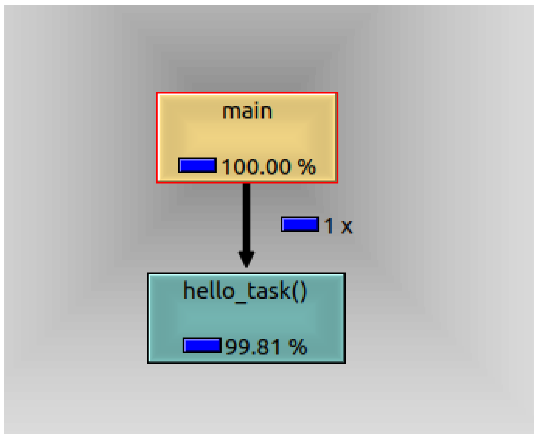


Figure 6.1: Example of Kcachegrind visualization after a valgrind callgrind code profiling

```cpp
#include <iostream>
void hello_task()
{
    //doing something with high cost
    for (int i = 0; i < 10000; i++)
        std::cout << "Dummy task!";
}
int main()
{
    //Example of program
    std::cout << "Hello World!";
    hello_task(); //executing a dummy test function
    std::cout << "Done";
    return 0;
}
```

Code 6.1: Example of program for Kcachegrind visualization

6.3 Hyperledger Sawtooth

6.3.1 Web Application

Thanks to previous work (with R. Kromes and F. Verdier), I have studied, built, and tested Hyperldger Sawtooth. The published paper [28] lead us to multiple conclusions. Hyperldger Sawtooth can be used for the SIM project use case under the right circumstances.

The web application was built in NodeJs, executed on a server, accessing a node directly. It kept a local database updated in real-time by binding a WebSocket to each node events (new validation, new block, etc.).

6.3.2 C++ client library Send Transaction

In Hyperledger Sawtooth the client application has to follow requirements. These requirements and steps are:

1. Build the transaction:
   - (a) Initialize a batch (in Protobuf)
   - (b) Initialize a transaction inside the batch
   - (c) Create the hash (sha-512) of message = message_hash512
   - (d) Set transaction headers:
     - i. Set batch and signer public key
     - ii. Set transaction family name and version
     - iii. Set payload (= JSON message) hash with message_hash512
     - iv. Set transaction inputs/outputs (=blockchain memory addresses we need to read/write)
     - v. Set a nonce (= bytes to prevent execution the transaction multiple times by accident)
   - (e) Set transaction payload
   - (f) Sign (the hash sha-256 of the transaction header) and set the transaction header (= transaction ID identifier)
   - (g) Add transaction ID to batch header
   - (h) Set batch header signer public key
   - (i) Sign (the hash sha-256 of the batch header) and set the batch header
2. Serialize the Protobuf = transaction data for HTTP request to blockchain node
3. Send the transaction data

The signature in Hyperledger Sawtooth is a a standard ECDSA secp256k1 signature. Protobuf is a language-neutral and platform-neutral protocol buffer for serializing structured data. Thanks to this mechanism, we can write the client in a different language and still have a compatible transaction structure.

After I built a C++ client capable of sending transactions to Sawtooth blockchain, we analysed (with R. Kromes) Hyperledger Sawtooth and the client program. The program results and Hyperledger Sawtooth performances were published in two papers [28] and [29].

The first step was to establish a profiling of the program and determine the client program's most intensive functions (that can be optimized) for a later IoT device integration. Profiling showed that hash function sha-256 could be optimized.

Analysing time delays in the program confirmed the Valgrind results (Table 6.1). The study also shows that the ECDSA signature algorithm occupation time is 3.82% (using data smaller than 1 MBytes).

The results lead us to combine a SystemC model (BCM2837) of the Raspberry Pi, with a dedicated hardware accelerator modules for the two hashing functions. The

Data Size	Total exec. time	Time occupation of total time by SHA-512	Time of creation of one SHA-512 Hash
1 MByte	4.495 s	3.46 %	9.382 μs
2 MBytes	8.76 s	3.49 %	9.433 μs
41 MBytes	161.931 s	4.04 %	10.21 μs
Average :		3.66 %	9.675 μs

Table 6.1: Summarized execution time (Data $\geq$ 1 MByte) reported as an average of 10 run

simulation showed a gain of 293 (SHA-512) and 112 (SHA-256). The gain is obtained by comparing the execution time of a CPU and an ASIC.

The last work or Hyperledger Sawtooth [29] proposed and executed **a secure ecosystem specific for the use case**. The proposed ecosystem is entirely decentralized using IPFS [43] as a decentralized file system and an encryption mechanism to secure the use-case data (Fig. 6.2).

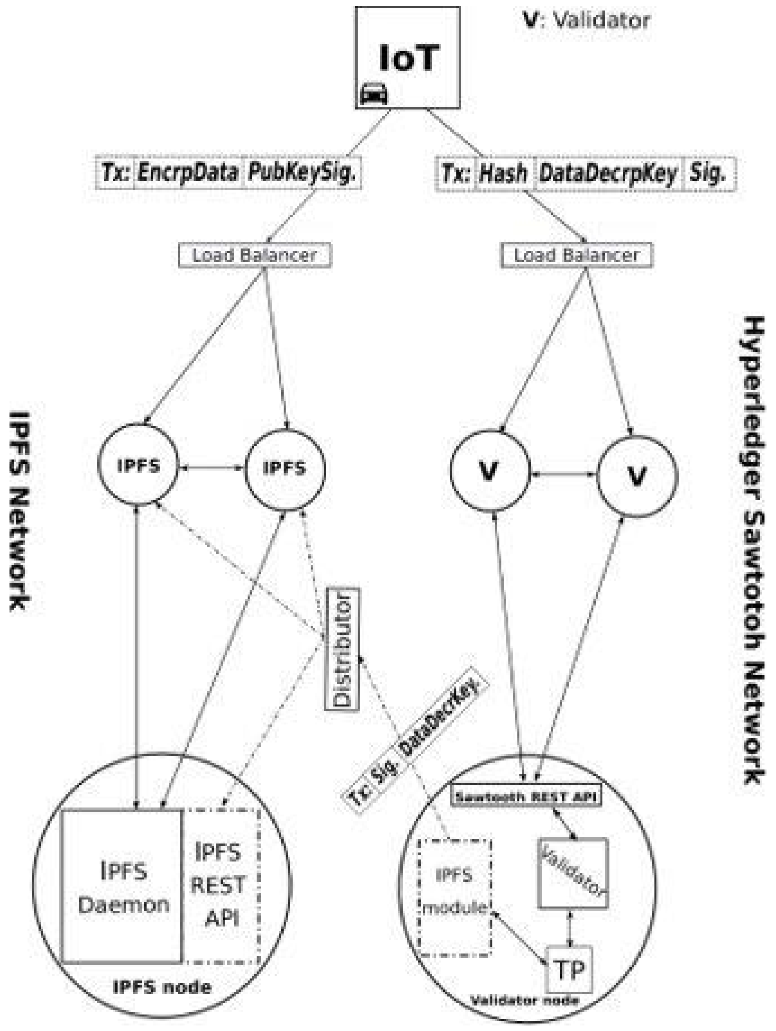
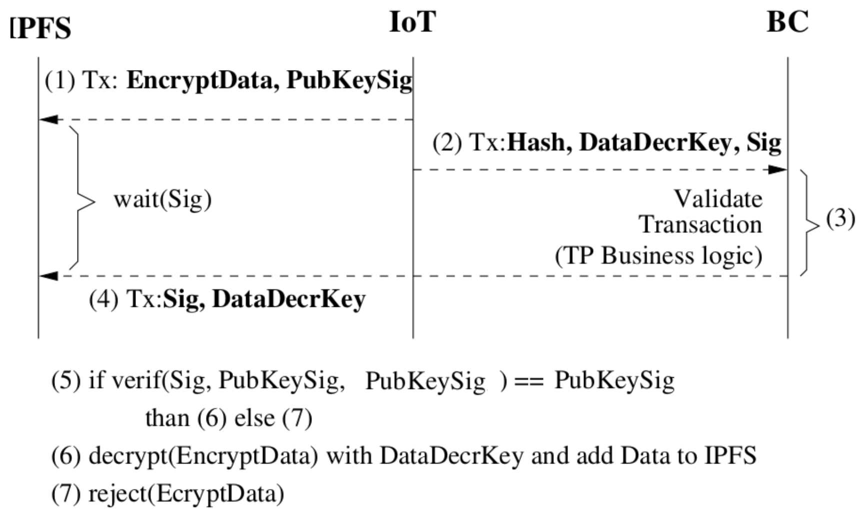


Figure 6.2: Hyperledger Sawtooth ecosystem for the SIM project

The IoT in the ecosystem built a transaction that contains two specific pieces of information:

- the reference to the data in IPFS
- a signature of the data sent to IPFS

The ecosystem stores the raw data of the IoT (car odometer, speed, etc.) in IPFS and sends a transactions, containing the default transaction data plus the two specific information above, to the blockchain.
Inside the blockchain we have a permission mechanism to identify the IoT.
To verify the data origin inside of IPFS we built an *IPFS module* along with a protocol (Fig. 6.3). It is required to have that protocol to prevent unauthorized IoT to send data to IPFS.


IPFS
IoT
BC
(1) Tx: **EncryptData, PubKeySig**
(2) Tx:**Hash, DataDecrKey, Sig**
wait(Sig)
Validate Transaction (TP Business logic)
(3)
(4) Tx:**Sig, DataDecrKey**

(5) if verif(Sig, PubKeySig, PubKeySig) == PubKeySig
than (6) else (7)
(6) decrypt(EncryptData) with DataDecrKey and add Data to IPFS
(7) reject(EcryptData)

Figure 6.3: Hyperledger Sawtooth - IPFS protocol

Along with the ecosystem, we provided detailed results on two consensuses. Each consensus reached a certain threshold that needs to be improved to make the blockchain scalable enough for the use-case. Using PBFT the transaction rate over 10k transaction is 2.7tx/sec.

After the blockchain - file system analysis, we analysed the IoT client program. The client program is the same Sawtooth program as previously, but with the IPFS requirements.
Results on the IoT showed that the client has a prominent SHA-256 hash function (Fig.6.4). Thanks to previous work, we know it can be optimized.

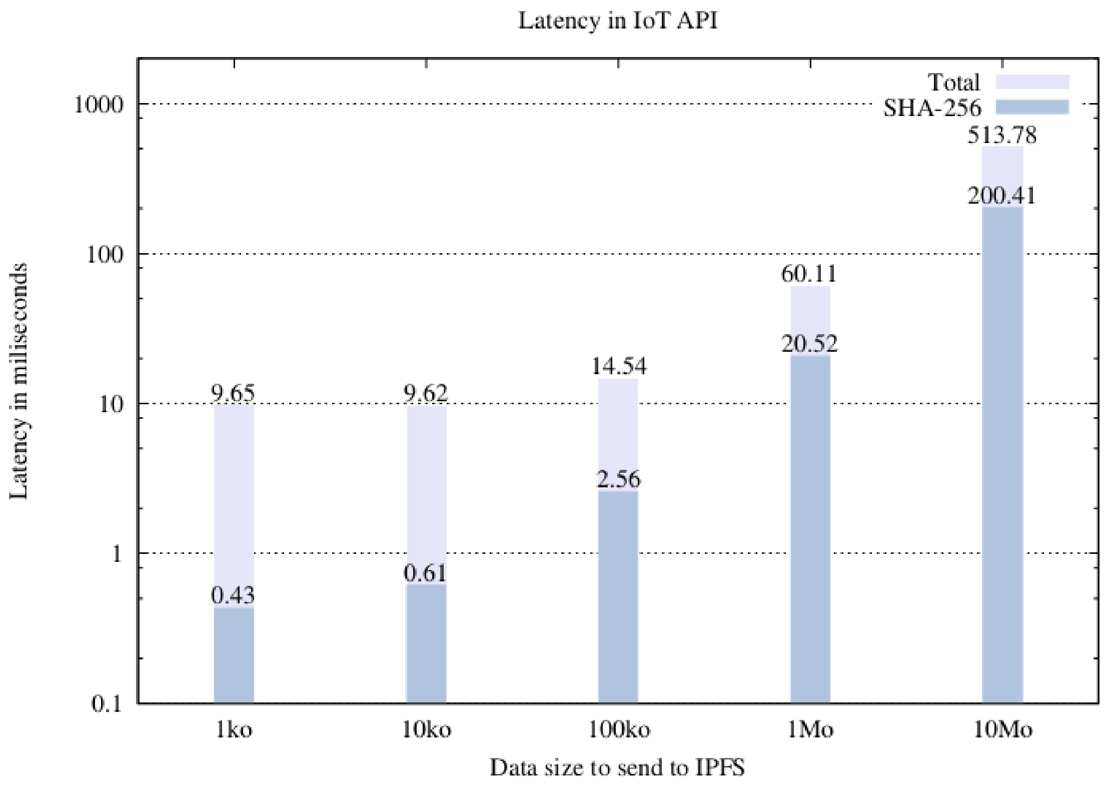


Figure 6.4: Hyperledger Sawtooth - IoT C++ client latency

This conclude that Hyperledger Sawtooth has good potential for the SIM project. It it possible to implement a true decentralized ecosystem for the use-case. Also optimization is possible on the IoT client.

In future works, Hyperledger Sawtooth has to be tested for pushing the consensus limit, by testing more blockchain and network configurations.

6.4 EOS.IO

With EOS.IO the work done was first to build an entire custom decentralized application that sends data to the blockchain. In the SIM project use case, following the work done in [29], the information sent in the blockchain ledger, is hashes (of the user data) and public keys.

6.4.1 Decentralized Application

As a proof of concept, the dApp built is sending transactions that allow some kind of authentication, securely save records of data (strings) and view the data according the authentication structure.

Graph 6.5 shows the practical implementation of dApp within the blockchain system. dApp are the result of the creation of smart contracts describing the application. It is also possible to create a **D**ecentralized **A**utonomous **O**rganization (DAO) on top of the dApp and SCs as an abstract layer. We should notice that dApps have the ability to exist because the blockchain structure allows it. This means the blockchain should have: API to connect to blockchain, sufficient transaction speed, and preferably feeless transactions.

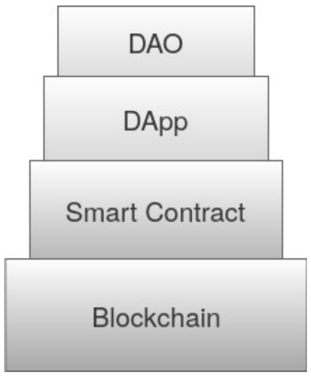


Figure 6.5: dApp & DAO: a layer on top of blockchain and smart contracts

A dApp execution is described in Fig. 6.6, and follows three main steps:

- The dApp is retrieved from a source provided by any system (here an endpoint from internet).
- The dApp is then executed on the client side (we build a WebApp as dApp, thus executed in the browser).
- Finally, all dApp data source and exchange are operated directly with a blockchain node

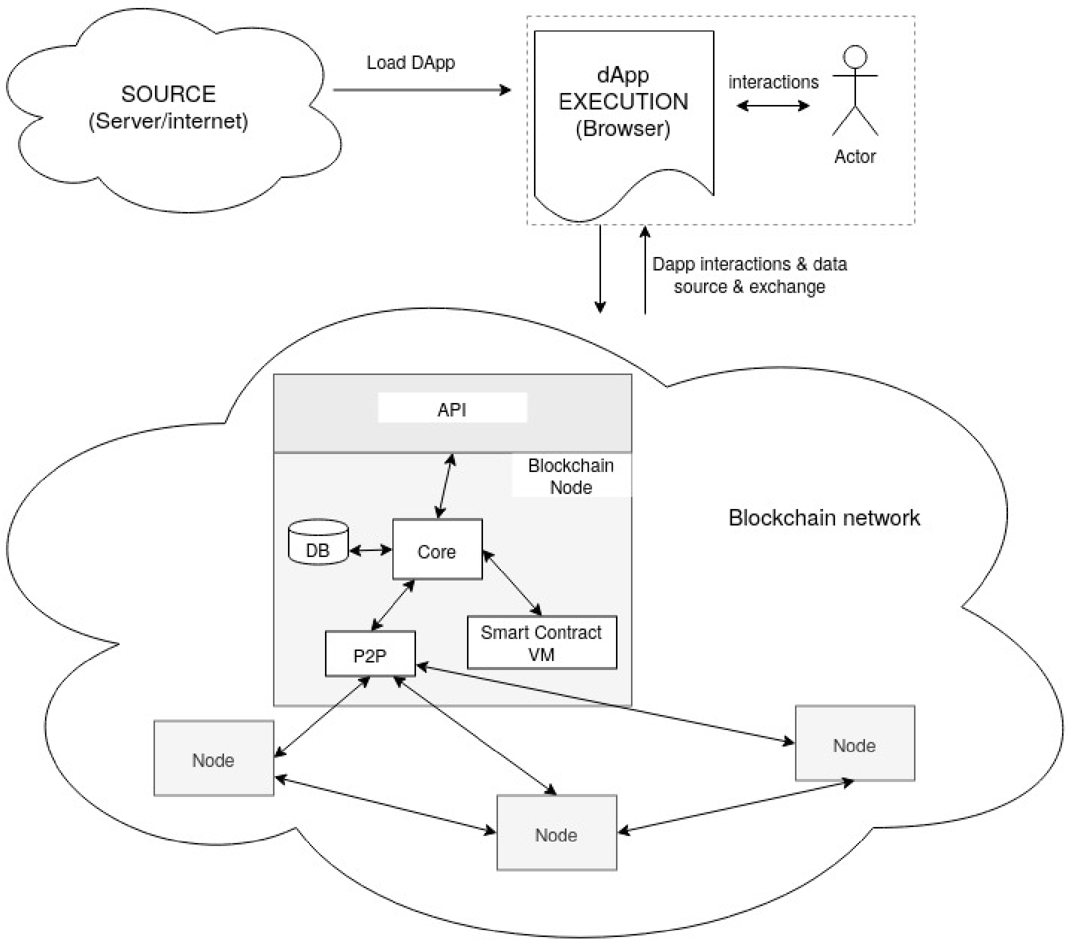


Figure 6.6: dApp integration on blockchains

This dApp has been built in JavaScript, with the *React* [44] Web framework. It includes only one library, *eosjs* [45], maintained by the EOS community.

The dApp built has an interface for viewing sent transactions, viewing the application configuration,

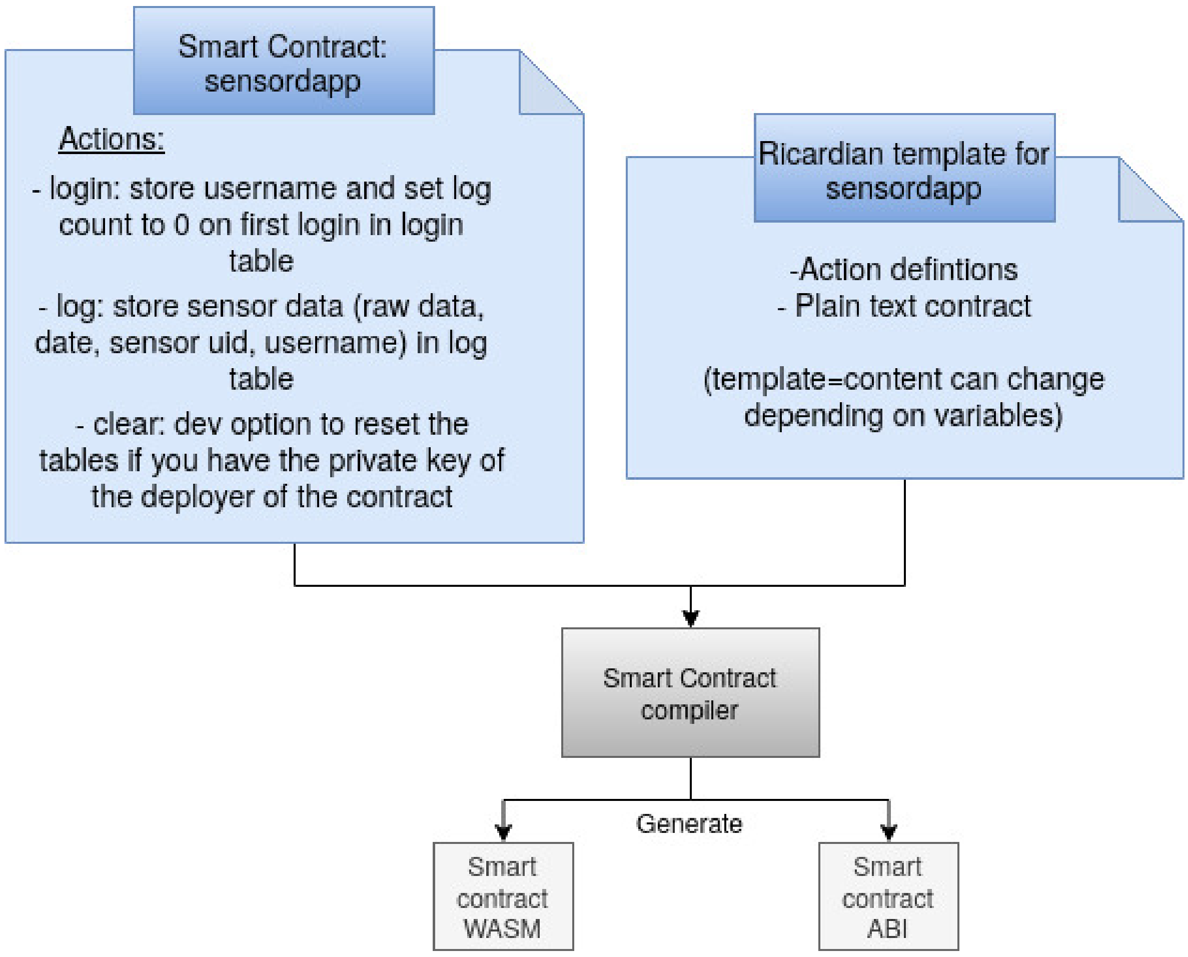

Smart Contract: sensordapp
Actions:
- login: store username and set log count to 0 on first login in login table
- log: store sensor data (raw data, date, sensor uid, username) in log table
- clear: dev option to reset the tables if you have the private key of the deployer of the contract
Ricardian template for sensordapp
-Action defintions
- Plain text contract
(template=content can change depending on variables)
Smart Contract compiler
Generate
Smart contract WASM
Smart contract ABI

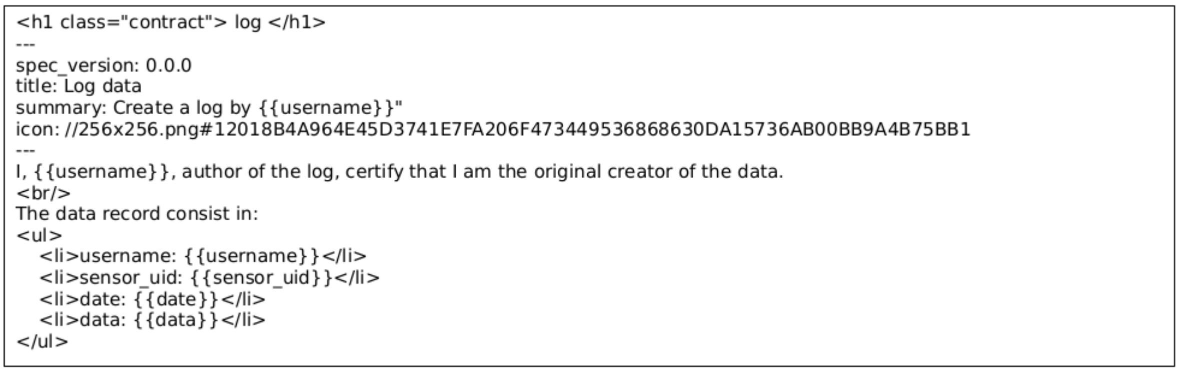
Figure 6.7: EOS dApp smart contract structure

The smart contract used for our dApp (Fig. 6.7) has three actions and a matching Ricardian Contract for each actions. As explained in Sect. 5.3, the smart contract project is compiled and produce a Wasm binary with a ABI file. These two files are then deployed to an EOS node using a smart contract account. This account is the owner of the smart contract.

The authority built into the smart contract prevents the action *log* only available to accounts that have executed the action *login* before. Therefore, it is impossible to send (and thus save) data to the EOS ledger without agreeing to the smart contract *login* action.

In a real life scenario, the login action would have a Ricardian Contract containing the legal information about the client agreement of using the smart contract.

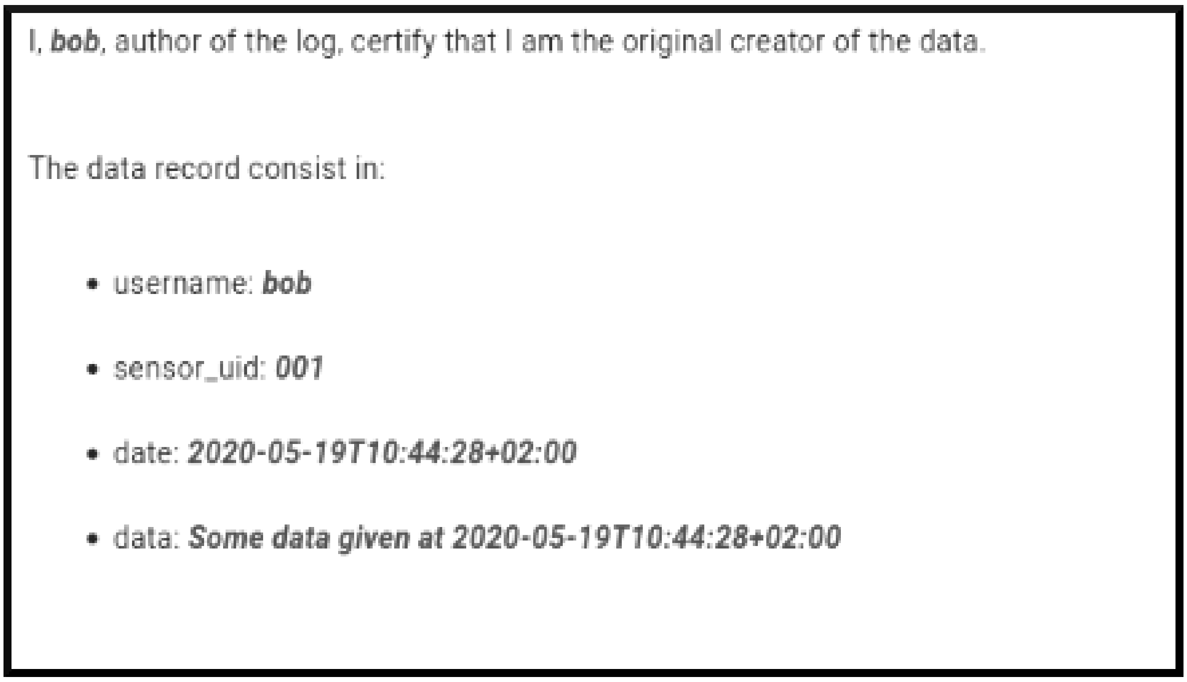
To understand how the practical Ricardian Contract, Fig. 6.8 show a really basic example of template used for the *log* action Ricardian Contract template.



```
<h1 class="contract"> log </h1>
---
spec_version: 0.0.0
title: Log data
summary: Create a log by {{username}}"
icon: //256x256.png#12018B4A964E45D3741E7FA206F473449536868630DA15736AB00BB9A4B75BB1
---
I, {{username}}, author of the log, certify that I am the original creator of the data.
<br/>
The data record consist in:
<ul>
   <li>username: {{username}}</li>
   <li>sensor_uid: {{sensor_uid}}</li>
   <li>date: {{date}}</li>
   <li>data: {{data}}</li>
</ul>
```

Figure 6.8: *log* action Ricardian Contract template

As you can see in the template 6.8, variables can be inserted using **{{**`<VARIABLE>`**}}**. The variables are provides from the transaction action data. The output of the template when given input looks like Fig. 6.9.



I, ***bob***, author of the log, certify that I am the original creator of the data.

The data record consist in:

- username: ***bob***
- sensor_uid: ***001***
- date: ***2020-05-19T10:44:28+02:00***
- data: ***Some data given at 2020-05-19T10:44:28+02:00***

Figure 6.9: EOS.IO dApp: Ricardian Contract *log* action after template executed

The user path in the dApp works as follows:

1. Get the app from IPFS: One HTTP request containing the entire app
2. Login (Fig. 6.10) (send transaction *login*) using the user private key (private key is only being used by the app to sign all transactions)
3. View dashboard (Fig. 6.11) if login (retrieve data from blockchain node)
4. If open page "Log Sensor data"
   (a) Set field data: fill input boxes with the info the user wants to send to the blockchain smart contract (Fig. 6.12)

(b) Open the Ricardian Contract: This will open the formal language contract with the expected output of the template based on the user inputs. (Fig. 6.13)

(c) If contract accepted: the app sends the transaction with the smart contract data

5. If open page "View Logs": The app retrieve all the logs stored in the blockchain sent by the user

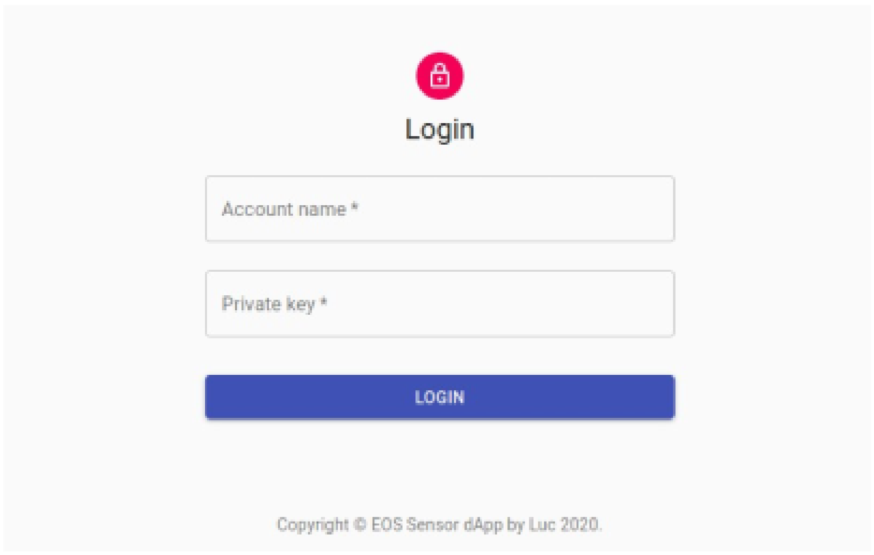
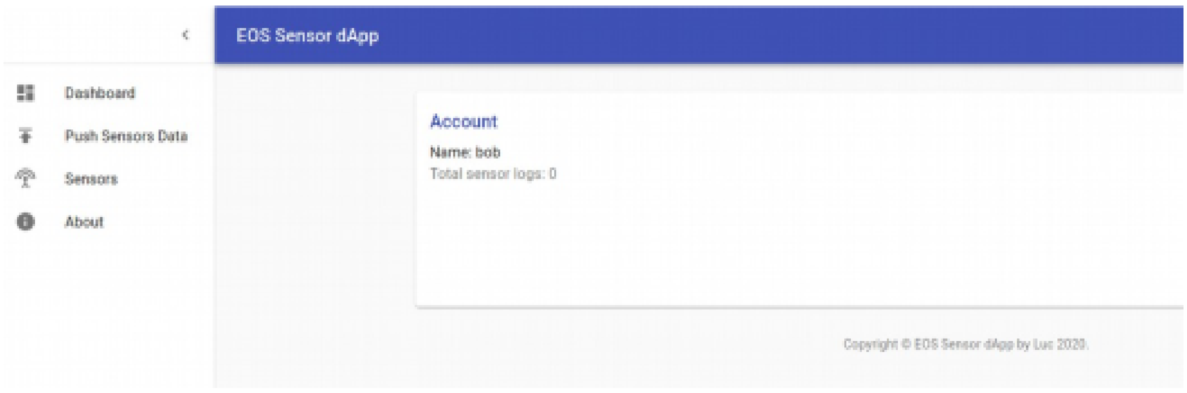
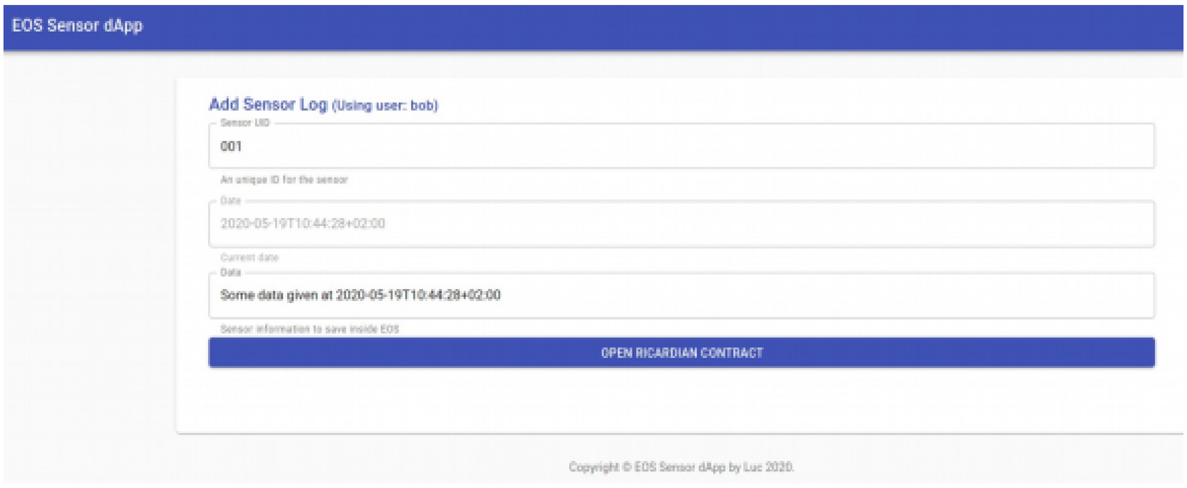


Figure 6.10: EOS.IO dApp: Login



Name: bob
Total sensor logs: 0

Figure 6.11: EOS.IO dApp: View dashboard


EOS Sensor dApp

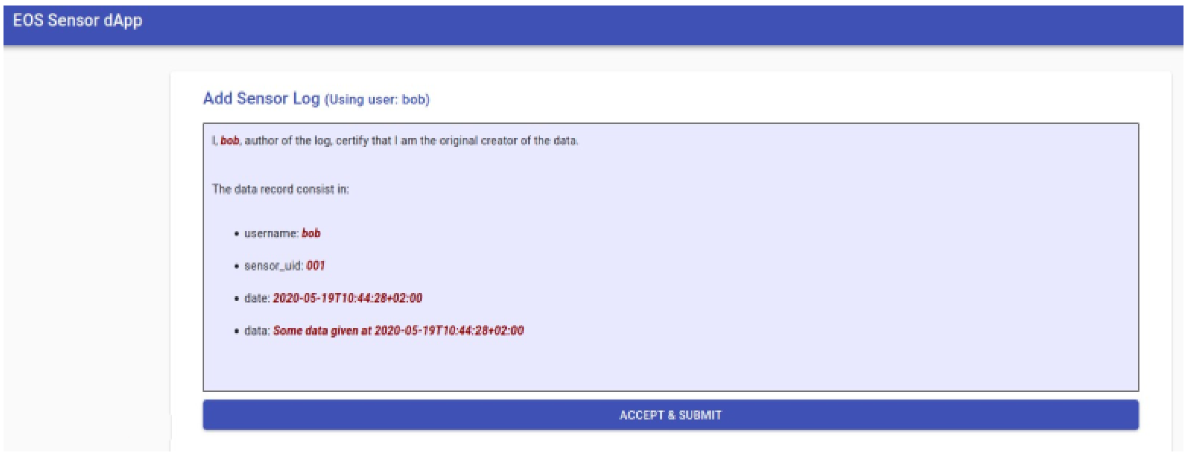
Add Sensor Log (Using user: bob)

Sensor UID: 001
An unique ID for the sensor

Date: 2020-05-19T10:44:28+02:00
Current date

Data: Some data given at 2020-05-19T10:44:28+02:00
Sensor information to save inside EOS

Figure 6.12: EOS.IO dApp: Set data fields


EOS Sensor dApp

Add Sensor Log (Using user: bob)

I, ***bob***, author of the log, certify that I am the original creator of the data.

The data record consist in:

- username: ***bob***
- sensor_uid: ***001***
- date: ***2020-05-19T10:44:28+02:00***
- data: ***Some data given at 2020-05-19T10:44:28+02:00***

Figure 6.13: EOS.IO dApp: Open Ricardian Contract

6.4.2 C++ client Send Transaction

In EOS.IO, the client application needs to follow these steps :

1. Initialize variables and objects (JSON object, cryptography object, etc.)
2. Retrieve node information:
   (a) Get the node last irreversible block number ID: is the identifier of the last immutable block that has been pushed to the blockchain ledger
   (b) Get the node chain ID: an identifier for the EOS node
   (c) Get the last irreversible ref block prefix and build expiration based on block timestamp
   (d) Get the transaction smart contract ABI (and cache it)
3. Build the transaction:
   (a) Transform the transaction action (a JSON string object describing the inputs for the smart contract action, e.g. *log* action in Sect. 6.4.1) into a binary format
   (b) Pack the transaction with all previous data (transaction JSON to binary)
   (c) Built a signature with the transaction binary
   (d) Append signature to the transaction
4. Send the transaction

EOS.IO client application needs to make three API calls to a node to retrieve all the first step information. It is important because IoT devices can have a limited internet connection. Table 6.2 summarizes the data transferred into/out of the program. Of course the data transmitted depends on the data included in the transaction, in our case we use an smart contract that set one field on the blockchain record. One field contains 32 bytes.

Number of tests	Transmission (Bytes)	Reception (Bytes)
100	342	3362

Table 6.2: Data transmission and reception of the EOS.IO C++ client

The result above in Table 6.2 shows that the IoT has to send 342 bytes that are the bytes of the transaction being push and field bytes needed to be retrieve the last irreversible block. The IoT also has to retrieve 3362 bytes needed to retrieve the node information, the smart contract ABI, and the last irreversible block.

The C++ program follows these same steps. Step 1., 2., 3. and 4. are functions that encapsulate the sub-steps (we will call them sub-functions). They are then used for easy program profiling. The profiling results are visualized on Fig. 6.14.

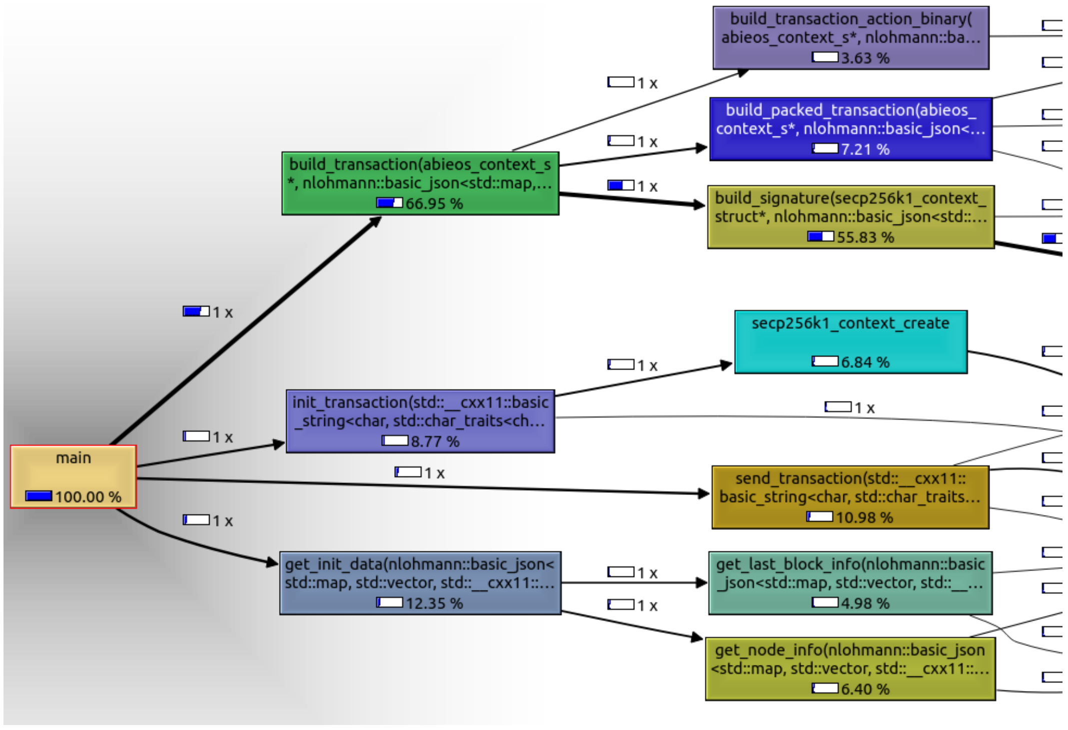

build_transaction_action_binary(abieos_context_s*, nlohmann::ba...
3.63 %
build_packed_transaction(abieos_ context_s*, nlohmann::basic_json<...
7.21 %
build_signature(secp256k1_context_ struct*, nlohmann::basic_json<std::...
55.83 %
build_transaction(abieos_context_s *, nlohmann::basic_json<std::map,...
66.95 %
secp256k1_context_create
6.84 %
init_transaction(std::__cxx11::basic _string<char, std::char_traits<ch...
8.77 %
main
100.00 %
send_transaction(std::__cxx11:: basic_string<char, std::char_traits...
10.98 %
get_init_data(nlohmann::basic_json< std::map, std::vector, std::__cxx11::...
12.35 %
get_last_block_info(nlohmann::basic _json<std::map, std::vector, std::__...
4.98 %
get_node_info(nlohmann::basic_json <std::map, std::vector, std::__cxx11::...
6.40 %
1 x

Figure 6.14: EOS.IO C++ client profiling: main function

As you can see, building the transaction (step 3) cost is 66% higher. Meaning the most used functions are in this step. Relative to the main function, the sign function (*secp256k1_ecdsa_sign_recoverable()*) costs 53.39% of the entire program. It is confirmed by selecting step 3 build transaction as the parent of the graph. We can observe that the sign function is again the most used (Fig. 6.15). Also the step 3.(c) (sub-function "build_signature()") is used at more 95% by the *secp256k1_ecdsa_sign_recoverable()* function.

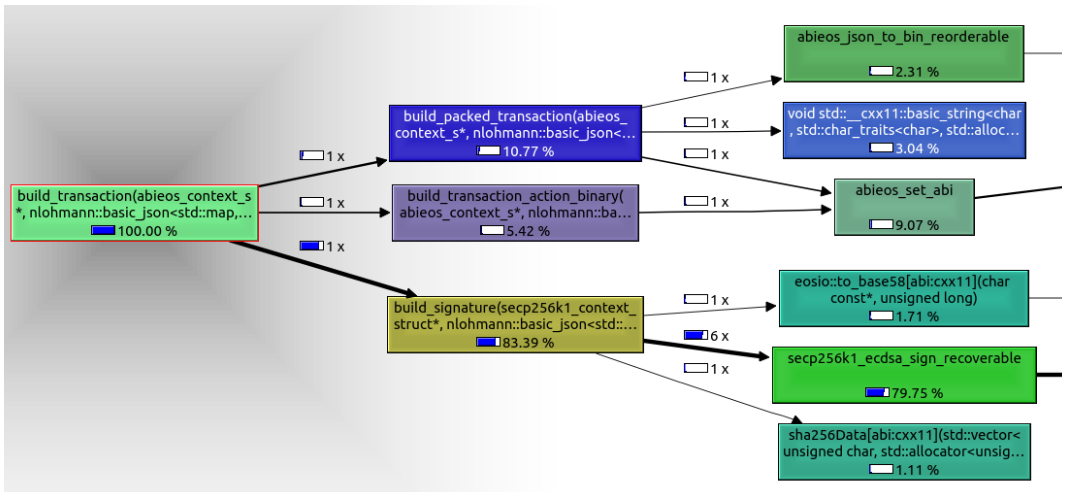

abieos_json_to_bin_reorderable
2.31 %
build_packed_transaction(abieos_ context_s*, nlohmann::basic_json<...
10.77 %
void std::__cxx11::basic_string<char , std::char_traits<char>, std::alloc...
3.04 %
build_transaction(abieos_context_s *, nlohmann::basic_json<std::map,...
100.00 %
build_transaction_action_binary(abieos_context_s*, nlohmann::ba...
5.42 %
abieos_set_abi
9.07 %
eosio::to_base58[abi:cxx11](char const*, unsigned long)
1.71 %
build_signature(secp256k1_context_ struct*, nlohmann::basic_json<std::...
83.39 %
secp256k1_ecdsa_sign_recoverable
79.75 %
sha256Data[abi:cxx11](std::vector< unsigned char, std::allocator<unsig...
1.11 %
1 x
6 x

Figure 6.15: EOS.IO C++ client profiling: build transaction step

This concludes that to accelerate and optimize the client application of the EOS.IO blockchain we can use a dedicated hardware accelerator IP for the *secp256k1_ecdsa _sign_recoverable()* function.

⚠ An important notice on the C++ program is that the signature generation cost is not deterministic. The results shown previously are only results with best case scenario of the signature generation.

To generate a valid signature, in EOS.IO, it is required to have a canonical signature (See in Code 6.2). An ECDSA secp256k1 signature can have four different, perfectly valid forms (called *Signature Malleability* [46]). To prevent the same mistake that Bitcoin has, EOS.IO must have only one valid, canonical form.
In order to be canonical, our program has a while loop that iterates a maximum four times with the condition "is canonical". This condition means that the *build_transaction()* function cost is minimum when the first iteration is true and a maximum when the function is false three times.
The reason for having a maximum of four iteration is that the *secp256k1_ecdsa _sign _recoverable()* function can only have a recover id going from 0 to 3.

```cpp
//Sign the data following EOS standard:
int recover_id;
int loops = 0;
do {
  // make new signature while canonical is false
  CHECK(SECP256K1_API::secp256k1_ecdsa_sign_recoverable(ctx,
      &recoverable_signature,
      signature_feed_sha256_bytes,
      priv_key_bytes,
      NULL,
      &loops
      ) == 1);
  CHECK(SECP256K1_API::
   secp256k1_ecdsa_recoverable_signature_serialize_compact(
         ctx,
         recoverable_signature_serilized,
         &recover_id,
         &recoverable_signature
         ) == 1);
  // std::cout << "Test canonical " << loops << " rec id: " <<
   recover_id << std::endl;
  loops++;
} while (is_canonical(recoverable_signature_serilized) == false);
```

Code 6.2: EOS.IO dApp: Signature generation function

Next, we can also observe with the visualization tool, that **sha-256 uses 13% of the total program cost**. This function has been successfully simulated with an accelerator hardware model in System-C in the Hyperledger Sawtooth implementation and paper (See Sect. 6.3.2).

In future works, we have to analyze the best and worst-case scenario to understand the signature generation's impact on the entire program. This will be followed by an analysis of the ECDSA function execution time in general and the execution time of the sub-functions alone. The paper [18] uses FPGA to accelerate the arithmetic verification of ECDSA signature. More in-depth study about ECDSA hardware implementation for signature generation could lead us to optimize the IoT C++ client. Based on these results, it will be possible to conclude on a new hardware accelerator model for ECDSA.

Chapter 7

Conclusion and perspectives

This six-month internship allowed me to have an in-depth study of the practical implementations of decentralized technology. The aim is to complete an overview of multiple DLT software in the context of the SIM project. Thanks to previous and complementary work, the main challenges in this internship were answered. I studied two blockchains and built specific software whose purpose is to be integrated on constrained devices. After analyzing the results, I conclude that integrating the software on the embedded device can be optimized by using hardware accelerators. However, more development is necessary to complete my results, such as execution time analysis and also hardware accelerator simulation of secp256k1 most used functions (in our C++ client).
Also, as the report presents, IOTA software is at a turning point in time. Only after the 2021 upgrade, this technology will be able to meet our context requirements. Indeed, IOTA is missing a stable integration of smart contracts in their software. However, they presented multiple PoC (such as Qubic and IOTA Data Marketplace), demonstrating that the SIM project should stay up to date with IOTA upgrades.

Blockchain technology still has a lot of unanswered questions. Thanks to this internship, I will follow the research on this topic as a thesis subject, specifically the role and implementation of IoT architecture within DLT technology. Again, the challenge is to integrate blockchain requirements on constrained devices that interface with the world around us. This thesis, *an exploration of IoT architecture for dApp and DAO blockchain*, will take part in several phases. After additional research on decentralized applications and decentralized autonomous organizations, the thesis will take part in practical applications. Practical applications target only use-cases involving everyday objects and will require an in-depth study of embedded hardware architectures. One or multiple case-studies will allow the thesis to build case specific hardware or hardware architectures models.

I personally believe DLT technology has still undiscovered potential, and the SIM project perfectly shows the unlimited applications for everyday life. A excellent analogy to explain the state of blockchain: “It is like trying to build electronic mail (e-mail) without any mail protocol and only IP/TCP at your disposition.”.

It is good to notice non-technical obstacles such as making individuals aware of the technology. We cannot build technology just because we can do it. It also has to have a purpose for the end-user.

Bibliography

[1] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] V. Buterin *et al.*, "Ethereum white paper, 2014," *URL https://github.com/ethereum/wiki/wiki/White-Paper*, 2013.

[3] N. Szabo, "The idea of smart contracts," *Nick Szabo's Papers and Concise Tutorials*, vol. 6, 1997.

[4] F. Verdier, P. De Filippi, F. Mallet, P. Collet, L. Arena, A. Attour, M. Ballator, M. Chessa, A. Festré, P. Guitton-Ouhamou, R. Bernhard, and B. Miramond, "Smart IoT for Mobility: Automating of Mobility Value Chain through the Adoption of Smart Contracts within IoT Platforms," 17th Driving Simulation & Virtual Reality Conference (DSC 2018), Sep. 2018, poster. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01903049

[5] G. Ishmaev, "The ethical limits of blockchain-enabled markets for private iot data," *Philosophy & Technology*, pp. 1–22, 2019. [Online]. Available: https://link.springer.com/content/pdf/10.1007/s13347-019-00361-y.pdf

[6] Wouter Penard and Tim van Werkhoven, *Cryptography in Context.* Gerard Tel, 2001, ch. 1: On the Secure Hash Algorithm family.

[7] S. Ghaffaripour and A. Miri, "Application of blockchain to patient-centric access control in medical data management systems," in *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, Oct 2019, pp. 0190–0196.

[8] IBM, "Ibm blockchain supply chain solutions," 2020. [Online]. Available: https://www.ibm.com/blockchain/industries/supply-chain

[9] B. Shahzad and J. Crowcroft, "Trustworthy electronic voting using adjusted blockchain technology," *IEEE Access*, vol. 7, pp. 24 477–24 488, 2019.

[10] S. Zhang and J. Lee, "Double-spending with a sybil attack in the bitcoin decentralized network," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 10, pp. 5715–5722, 2019.

[11] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, and P. Verissimo, "Efficient byzantine fault-tolerance," *IEEE Transactions on Computers*, vol. 62, no. 1, pp. 16–30, 2013.

[12] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security*, N. Christin and R. Safavi-Naini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 436–454.

[13] Ian Grigg, "The ricardian contract," 2000. [Online]. Available: https://iang.org/papers/ricardian_contract.html

[14] ——, "Financial cryptography in 7 layers," 1999. [Online]. Available: https://iang.org/papers/fc7.html

[15] EOS.IO, "Ricardian template toolkit," 2020. [Online]. Available: https://eos.io/build-on-eosio/ricardian-template-toolkit/

[16] Certicom Research, "Standards for efficient cryptography," 2010. [Online]. Available: https://www.secg.org/sec2-v2.pdf

[17] ——, "Ecdsa secp256k1 parameters," 2010. [Online]. Available: https://www.secg.org/sec2-v2.pdf

[18] S. Tachibana, S. Araki, S. Kajihara, S. Azuchi, Y. Nakajo, and H. Shoda, "Fpga implementation of ecdsa for blockchain," in *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, 2019, pp. 1–2.

[19] A. S. Yahaya, N. Javaid, F. A. Alzahrani, A. Rehman, I. Ullah, A. Shahid, and M. Shafiq, "Blockchain based sustainable local energy trading considering home energy management and demurrage mechanism," *Sustainability*, vol. 12, no. 8, p. 3385, Apr 2020. [Online]. Available: http://dx.doi.org/10.3390/su12083385

[20] A. R. Rajput, Q. Li, M. Taleby Ahvanooey, and I. Masood, "Eacms: Emergency access control management system for personal health record based on blockchain," *IEEE Access*, vol. 7, pp. 84 304–84 317, 2019.

[21] Microsoft Azure, "Azure for health," 2020. [Online]. Available: https://azure.microsoft.com/en-us/industries/healthcare

[22] Google, "Google health," 2020. [Online]. Available: https://health.google.com/health/

[23] K. D. Mandl, W. W. Simons, W. C. Crawford, and J. M. Abbett, "Indivo: a personally controlled health record for health information exchange and communication." BMC Medical Informatics and Decision Making, 2007. [Online]. Available: https://doi.org/10.1186/1472-6947-7-25

[24] S. A. Abeyratne and R. Monfared, "Blockchain ready manufacturing supply chain using distributed ledger," Jan 2016. [Online]. Available: https://hdl.handle.net/2134/22625

[25] M. Al-Zubaidie, Z. Zhang, and J. Zhang, "Efficient and secure ECDSA algorithm and its applications: A survey," *CoRR*, vol. abs/1902.10313, 2019. [Online]. Available: http://arxiv.org/abs/1902.10313

[26] N. Jansma and B. Arrendondo, "Performance comparison of elliptic curve and rsa digital signatures," *nicj. net/files*, 2004.

[27] R. Kromes and F. Verdier, "Iot devices hardware modeling for executing blockchain and smart contracts applications," in *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, 2019, pp. 1–6.

[28] R. Kromes, L. Gerrits, and F. Verdier, "Adaptation of an embedded architecture to run Hyperledger Sawtooth Application," in *for IEEE IEMCON 2019*, Canada, France, Oct. 2019, p. 7. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02288894

[29] L. Gerrits, R. Kromes, and F. Verdier, "A True Decentralized Implementation Based on IoT and Blockchain: a Vehicle Accident Use Case," in *COINS 2020 - IEEE International Conference on Omni-layer Intelligent Systems*, IEEE, Ed., Barcelone, Spain, Aug. 2020, p. 6. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02648403

[30] B. Ampel, M. Patton, and H. Chen, "Performance modeling of hyperledger sawtooth blockchain," in *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2019, pp. 59–61.

[31] K. L. Brousmiche, T. Heno, C. Poulain, A. Dalmieres, and E. Ben Hamida, "Digitizing, securing and sharing vehicles life-cycle over a consortium blockchain: Lessons learned," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, pp. 1–5.

[32] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, and S. Uluagac, "Block4forensic: An integrated lightweight blockchain framework for forensics applications of connected vehicles," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 50–57, 2018.

[33] S. Popov, "The tangle," *cit. on*, p. 131, 2016. [Online]. Available: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf

[34] K. Thulasiraman and M. N. Swamy, *Graphs: theory and algorithms.* John Wiley & Sons, 1992.

[35] I. Korotkyi and S. Sachov, "Hardware accelerators for iota cryptocurrency," in *2019 IEEE 39th International Conference on Electronics and Nanotechnology (ELNANO)*, 2019, pp. 832–837.

[36] M. Colavita and G. Tanzer, "A cryptanalysis of iota's curl hash function," 2018.

[37] A. Penzkofer, B. Kusmierz, A. Capossele, W. Sanders, and O. Saa, "Parasite chain detection in the iota protocol," 2020. [Online]. Available: https://arxiv.org/abs/2004.13409

[38] Y. Huang, H. Wang, L. Wu, G. Tyson, X. Luo, R. Zhang, X. Liu, G. Huang, and X. Jiang, "Characterizing eosio blockchain," *arXiv preprint arXiv:2002.05369*, 2020.

[39] S. Lee, D. Kim, D. Kim, S. Son, and Y. Kim, "Who spent my {EOS}? on the (in) security of resource management of eos. io," in *13th {USENIX} Workshop on Offensive Technologies ({WOOT} 19)*, 2019.

[40] IOTA, "Qubic programming language," 2020. [Online]. Available: https://github.com/iotaledger/qupla

[41] Eric Hop, "The state of qubic and the future of smart contracts on iota," 2020. [Online]. Available: https://blog.iota.org/the-state-of-qubic-63ffb097da3f

[42] EOS.IO, "Eos.io technical white paper," 2020. [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md

[43] Juan Benet, "Ipfs - content addressed, versioned, p2p file system," 2020. [Online]. Available: https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[44] Facebook Open Source, "React," 2020. [Online]. Available: https://reactjs.org/

[45] EOS.IO, "eosjs," 2020. [Online]. Available: https://github.com/EOSIO/eosjs

[46] Bitcoin, "Transaction malleability," 2020. [Online]. Available: https://en.bitcoin.it/wiki/Transaction_malleability