Universidade de Aveiro Departamento de Engenharia Mecânica
2020

Diogo Filipe Duarte Costa

Proposal of a Traceability 4.0 System in Renault CACIA Using Blockchain Technologies

Proposta de um Sistema de Rastreabilidade 4.0 na Renault CACIA Usando Tecnologias Blockchain

Universidade de Aveiro Departamento de Engenharia Mecânica
2020

Diogo Filipe Duarte Costa

Proposal of a Traceability 4.0 System in Renault CACIA Using Blockchain Technologies

Proposta de um Sistema de Rastreabilidade 4.0 na Renault CACIA Usando Tecnologias Blockchain

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de José Paulo Santos, Professor Auxiliar, e de Armando Nolasco Pinto, Professor Associado com Agregação do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e de Engº Miguel Teixeira, Supervisor na Renault CACIA.

Este trabalho teve o apoio financeiro dos projetos UIDB/00481/2020 e UIDP/00481/2020 - FCT - Fundação para Ciência e Tecnologia; e CENTRO-01-0145-FEDER-022083 - Programa Operacional Regional do Centro (Centro2020), no âmbito do Acordo de Parceria Portugal 2020, através do Fundo Europeu de Desenvolvimento Regional.

o júri / the jury

presidente / president	**Prof. Doutor Victor Fernando Santos Neto** Professor Auxiliar em Regime Laboral da *Universidade de Aveiro*
	Prof. Doutor Paulo Alexandre Carreira Mateus Professor Catedrático do *Instituto Superior Técnico de Lisboa*
	Prof. Doutor José Paulo Santos Professor Auxiliar da *Universidade de Aveiro* (orientador)

agradecimentos / acknowledgements

Ao Prof. Doutor José Paulo Santos e ao Prof. Doutor Armando Nolasco Pinto, agradeço as suas orientações, disponibilidade e apoio ao longo de toda a minha dissertação. Ao Engenheiro Miguel Teixeira da Renault CACIA, agradeço a sua amizade ao longo de todo o meu trabalho, e a sua ajuda, fundamental nas etapas iniciais do meu estágio e dissertação. Nota especial de agradecimento, aos meus pais e à Izabele.

keywords Blockchain; Traceability; Industry 4.0; IIoT; Cryptography

abstract Increasing popularity of cryptocurrencies lead to the rapid development of blockchain-based distributed ledger keeping systems. Industrial applications for the new technologies were soon found, enhanced by needs of security, trust and transparency along the value-chain in traceability of products, parts and processes, particularly when transactions are being maintained through several different entities. Traceability solutions are also increasingly being built upon IoT or IIoT platforms. Although this allows for highly modular and flexible architectures, severe security deficiencies make these designs inapplicable to most real-life production lines. Lack of research and practical advances in the creation of modular and easily deployable frameworks motivated development of the project underlying this dissertation.

palavras-chave Blockchain; Rastreabilidade; Indústria 4.0; IIoT; Criptografia

resumo

O aumento da popularidade de *criptomoedas* levou ao rápido crescimento e desenvolvimento de sistemas baseados em *blockchain* de registos distribuídos. Aplicações industriais para estas novas tecnologias foram rapidamente encontradas, exacerbadas pelas necessidades de segurança, confiança e transparência ao longo da cadeia de valores na rastreabilidade de produtos, peças e processos, em particular quando as transações são mantidas através de várias entidades díspares. Soluções de traçabilidade são crescentemente construídas tendo por base plataformas de dispositivos IoT ou IIoT. Embora isto leve à criação de arquiteturas extremamente modulares e flexíveis, graves deficiências de segurança tornam estas inaplicáveis a um largo número de linhas de produção. A falta de avanços práticos nesta área, motivou o desenvolvimento do projeto subjacente a esta dissertação.

Contents

Intentionally blank page.

List of Tables

Intentionally blank page.

List of Figures

Intentionally blank page.

Listings

Intentionally blank page.

Acronyms

AES Advanced Encryption Standard. 15, 50

API Application Programming Interface. 39, 65

BBC British Broadcasting Company. 85

BFT Byzantine Fault-Tolerant. 24

CA Certification Authority. 20, 54

CBC Cipher Block Chaining. 16

CBM Cloud-Based Manufacturing. 11, 29

CIA Confidentiality, Integrity and Security. 17

CLI Command Line Interface. 86

CSS Cascading Style Sheets. 65

DH Diffie-Hellman. 20

DNS Domain Name System. 55

DPoS Delegated Proof of Stake. 24

DSA Digital Signature Algorithm. 20

DSS Digital Signature Standard. 20

ECB Electronic Codebook. 16

ECDHE Elliptic Curve Diffie Hellman Ephemeral. 56

ECDSA Elliptic Curve Digital Signature Algorithm. 77

EEPROM Electrically Erasable Programmable Read-Only Memory. 48

GPIO General Purpose Input/Output. 50, 85

HMAC Hashed Message Authentication Code. 18

HMI Human Machine Interface. 40

HTML Hypertext Markup Language. 65

HTTP Hypertext Transfer Protocol. 21

HTTPS Hypertext Transfer Protocol Secured. 38, 39

I^{2}C Inter-Integrated Circuit. 13, 48

IBM International Business Machines. xvii, 34

IDE Integrated Development Environment. 86

IIC Industrial Internet Consortium. 12

IIoT Industrial Internet of Things. 1

IoT Internet of Things. 2

IPC Industrial PC. 12

ISO International Organization for Standardization. xviii

ITU International Telecommunications Union. xviii, 20

IV Initialization Vector. 16, 54

KDC Key-Distribution Center. 20

LDR Light Dependent Resistor. 49

MAC Message Authentication Code. 18

MCU Microcontroller. 13

MD5 Message Digest Algorithm 5. 15

MSP Membership Service Provider. 34

NIST National Institute of Standards and Technology. 15

NSA National Security Agency. 15

OS Operating System. 55

PBFT Practical Byzantine Fault Tolerance. 24

PLC Programmable Logic Controller. 12

PoA Proof of Authority. 24

PoS Proof of Stake. 23

PoW Proof of Work. 22

PWM Pulse-Width Modulation. 48

RAM Random Access Memory. 85

REPL Read, Evaluate, Print, Loop. 86

RFID Radio-Frequency Identification. 45

RIPEMD RACE Integrity Primitives Evaluation Message Digest. 15

ROM Read-Only Memory. 49

RTC Real Time Clock. 49

SAP Systems, Applications and Products. 34

SBC Single Board Computer. 29

SCM Supply Chain Management. 11

SCP Statistical Process Control. 31

SHA Secure Hash Algorithm. 15

SPI Serial Peripheral Interface. 13, 48

SQL Structured Query Language. 42

SRAM Static Random Access Memory. 48, 49

SSH Secure Shell. 21

SSL Secure Sockets Layer. 20

TCP Transmission Control Protocol. 21

TCP/IP Transmission Control Protocol/Internet Protocol. 21

TLS Transport Layer Security. 20

UART Universal Asynchronous Receiver/Transmitter. 48

URL Uniform Resource Locator. 21

USB Universal Serial Bus. 13, 49

UUID Universally Unique Identifier. 53

VDOP Variable Displacement Oil Pump. 5

WSGI Web Server Gateway Interface. 77

YAML YAML Ain't Markup Language. 105

Intentionally blank page.

Glossary

Blockchain Fully-distributed software network aiming to keep records immutable and secure though the application of encryption techniques along with linkage of the appended data by means of hash-values. 1

Blockchain System A Blockchain System is comprised by a network of machines (called nodes); a blockchain data structure that is replicate all across the network; and a network protocol that defines basic rules for operation [20]. 24

Cipher A decription and/or encription algorithm. 16

Ciphertext Message after encryption; unreadable unless decrypted. 16

Cryptography The science of ensuring that message contents are kept secured. 2

Digital Signatures Authentication codes that allow to determine the provenance of a given message. 17

Digital Twin As defined by International Business Machines (IBM), a digital twin is "*the virtual representation of a physical object or system across its life-cycle. It uses real-time data and other sources to enable learning, reasoning, and dynamically recalibrating for improved decision making*" [47]. 14

Distributed Ledger "*A distributed ledger is an append-only store of transactions which is distributed across many machines*" [20]. 21

Edge Computing Computational work or data processing that occurs in the *edge layer*. Part of the computational workload is done next to low-level hardware (e.g., sensors) in order to relieve pressure from the central data centres and the cloud, improving overall efficiency and transmission speeds. 2

IIoT Enterprise focused variant of IoT-based technologies. An IoT device can be defined as the integration of embedded computing hardware with network capabilities. 1

Industry 4.0 Term given to the fourth industrial revolution, marked by the rise of automation and data exchange within manufacturing technologies and processes. 1

JSON Lightweight data-interchange format that is easily understood by humans and easily parsed by machines. Completely language independent, although based on conventions of the C-family languages [44]. 54

Nonce Contraction of *n*umber used *once*. Found in many cryptography protocols, the nonce value generally does not have to be either a secret or unpredictable, but it does need to be unique. 22

Plaintext Message before encryption or after decryption; text in a readable format. 16

Python Interpreted, high-level, programming language featuring a comprehensive standard library inclunding many of the tools applied in creating blockchain-styled networks. 2

RSA The most common public-key algorithm, created by Rivest, Shamir, and Adleman. 20, 21

Smart Contract The attributed creator of the Smart Contract, Nick Szabo, defines it as a "*Computerized transaction protocol that executes terms of a contract*" [46]. In other words, smart contracts are programs first being deployed as data in the blockchain and then executed each time a transaction occurs. For a smart contract to hold any power, it's code must be determinist and immutable [20]. 26

Smart Manufacturing Term that describes the modernization of current manufacturing process, in light of the principles brought forward by Industry 4.0. 12

Supply Chain Network of organizations and activities that supply enterprises with goods and services [5]. 1

Traceability According to the International Organization for Standardization (ISO) 9000: *"Traceability is the ability to trace the history, application, use and location of an item or its characteristics through recorded identification data"*. 1

X.509 Recommendation devised by International Telecommunications Union (ITU) and generally accepted by the Internet that defines the structure in which certificates are built. 20

Chapter 1

Introduction

1.1 Context

Stringent demands have been placed all throughout manufacturing processes, with greater product liability, rising quality standards and more complex customer requirements, particularly in fields that carry real harm risk to end users, such is the case of the automotive industry. Auto manufacturing is truly global, with parts being sourced and assembled from multiple entities worldwide, resulting in an intricate Supply Chain that requires constant monitoring to assure quality needs. However, the Fourth Industrial Revolution (Industry 4.0) set a higher priority towards Traceability, supporting the idea that every part of the supply chain must be monitored and controlled at all times by employing methodologies that work systematically, requiring minimal to none human interference.

Outdated solutions that rely heavily in manual or partially manual tracking techniques proved themselves to be inadequate for the extremely competitive automotive sector, as the unreliable and slow influx of data can not be promptly accessible. The operation of tracking a singular event within the value chain is a time and resource consuming action that is heavily hampered by information skewing or even absence along the intermediary steps taken in storing it, creating uncertainty regarding veracity. The concept of a traceability system that applies a network consisting of a central data storage server aided by Industrial Internet of Things (IIoT) devices then appears as a proposed cost-effective alternative to ensure real time data acquisition whilst it still being readily available through all participants of the supply chain. Nonetheless, these IIoT based platforms deploy highly centralized architectures, suffering from numerous technical limitations, and ultimately compromising the security of data being acquired by the means of a cyber-attack or single points of failure [1]. Alternatively, several processes are still based off and monitored by legacy devices that have to be adapted to achieve compliance with these newfound technologies.

In this dissertation, it is proposed a template architecture that allows not only for full integration of the IIoT technologies found in Renault CACIA's facilities, but also interfacing with older legacy devices; whilst working alongside existing enterprise traceability systems. With trust and credibility at the forefront of concerns, the solution must, therefore, be based on an auditable and transparent system. A Blockchain based, immutable distributed-ledger is proposed, avoiding issues typically associated with centralized architectures, as, for instance, single points of failure. Trust is created as historical data is maintained throughout all the network's participants, whether they might be different business units within a manufacturing facility or a distinct corporation all together. Transactions in the network are secured by Cryptography, and must be approved by all adhering members through a previously agreed upon consensus algorithm or method [2].

Credibility then becomes a cost-saving measure as data being kept on-chain can be quickly and accurately verified against the internal ledgers distributed across multiple peers, assuring that only truthful information is being used in subsequent operation management processes such as root defect cause analysis.

A master thesis by D.F.Rocha [3] was used as an entry point on the subject due to the author's previous efforts in integrating Renault CACIAS's production lines with Internet of Things (IoT) platforms.

1.2 Methodology and Expected Results

This dissertation aims for the creation of an IIoT and legacy device based solution, integrating blockchain technologies, allowing for deployment of a traceability system fully capable of tracking products during each stage of the manufacturing process, since the arrival of raw materials to the completed products' shipping. The final platform must be capable of assuring real-time information with digital certification allowing for rapid verification of data and transactions by not only internal (i.e., Renault CACIA employees), but also external entities. As any traceability solution that strives to be competitive in the marketplace, final outcome must ensure a low-latency, private and secure inflow of data, while keeping development and integration costs reduced.

This will be achieved by first analysing existing technologies and advances in the field, and assessing whether or not they are viable for final implementation. All software development will be based on high-level, general purpose programming languages, for example, Python for web-serving or JavaScript/AJAX for web interfaces. To keep development and deployment costs low, hardware used for testing will be based on powerful IoT devices such as ESP32 or ESP8266, applying Edge Computing concepts when needed (e.g., data encryption happening *on edge*).

Three main objectives can be pinpointed as mandatory characteristics for the success of the proposed architecture:

1. Data must be immutable and securely kept, guarded against cyber-attacks.
2. Allow for a high throughput and swift data access, as to not become a bottleneck to currently implemented manufacturing and traceability systems.
3. Underpinning framework must be lightweight and modular, assuring compatibility with most current enterprise systems, technologies and layouts, for rapid deployment and guaranteeing a *hot-swappable* architecture.

1.3 Document Organization

This document is divided into six distinct chapters, the first of which is this introduction. Chapter 2, Renault CACIA, provides an overview of the manufacturing facilities found at Renault CACIA, and a more detailed explanation of the Variable Displacement Oil Pumps and their production lines, for which the proposed architecture was initially devised for.

Chapter 3, Blockchain Concepts and Supporting Technologies, will look into Industry 4.0 and associated concepts, cryptographic and network security notions, followed by a deep analysis of what are blockchain and distributed ledger technologies, finalizing with studying other authors' work done in the field and existing commercial solutions.

Chapter 4, Proposed Blockchain Framework, displays the conceptual stacked-layer architecture, laying ground-work for the final implementation, and explaining the reasoning behind the chosen design path.

Chapter 5, Blockchain Framework Implementation, explains how the architecture was conceived, used technologies and hardware, programming methodology and deployment instructions.

The last chapter, 6 - Testing and Conclusions, contains security and performance testing with final concluding notes.

Intentionally blank page.

Chapter 2

Renault CACIA

2.1 Introduction to Renault CACIA

Founded in September of 1981 as Companhia Aveirense de Componentes para a Indústria Automóvel (C.A.C.I.A.), Renault CACIA is one of 40 plants throughout 16 countries that comprise Groupe Renault's industrial facilities [4]. Initially producing gearboxes and other mechanical components for a wide range of automotive manufacturers, as of 2001, Renault CACIA produces exclusively for the Renault-Nissan-Mitsubishi Alliance. The following figure, 2.1, depicts an aerial view of the Renault CACIA manufacturing plant.

Figure 2.1: Aerial View of the Renault CACIA Plant. Sourced from: [4]

Currently, Renault CACIA produces gearboxes as well as other engine components, namely, oil pumps and differentials. The Variable Displacement Oil Pump (VDOP) production line will be used as proving ground for the solution developed in this thesis, thus meriting an overview of its operational procedures, given in section 2.2.

2.2 The VDOP Production Line

Oil pumps are a pivotal mechanical component in any automobile, being responsible for the constant feed of oil to the engine and several other critical elements in the assembly. Variable displacement oil pumps, figure 2.2, control the amount of work performed by the pump, by matching pressure and volume to conditions such as engine temperature, load or speed.

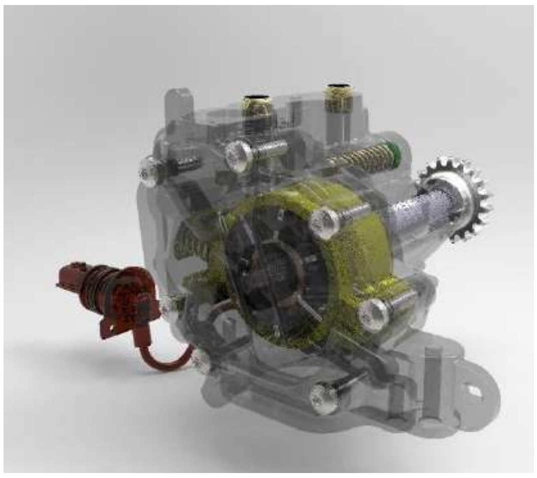


Figure 2.2: Variable Displacement Oil Pump.

These oil pumps are constituted by a multitude of different components, figure 2.3, being assembled across two production lines, Line 1 and Line 2. These lines are located at the Mechanical Components building at Renault CACIA.

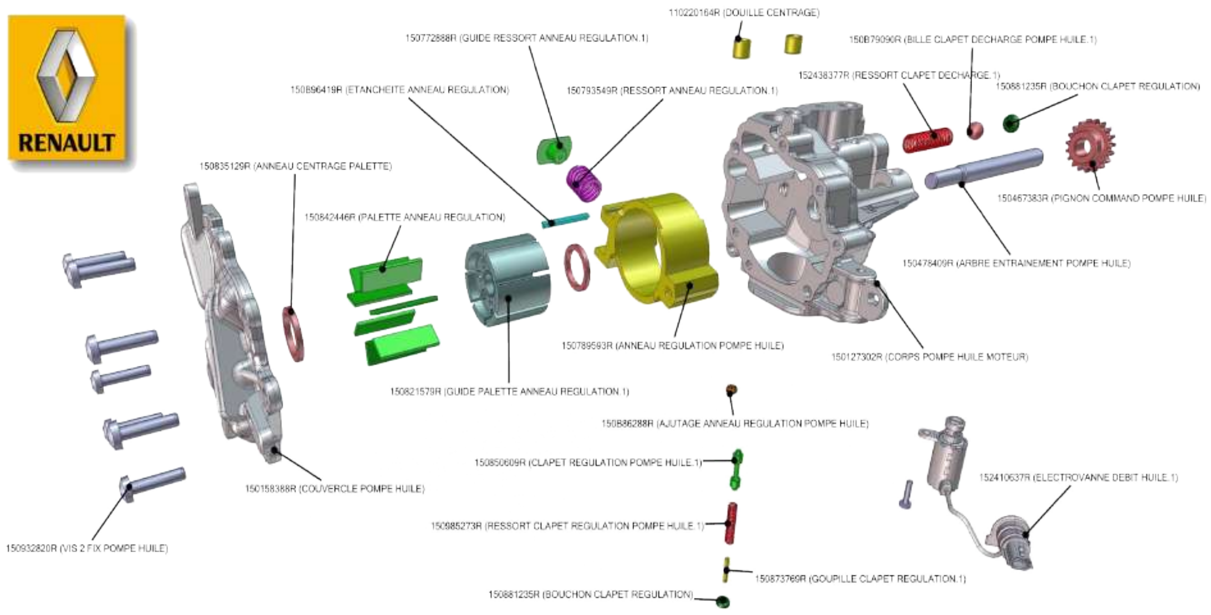

RENAULT
110220164R (DOUILLE CENTRAGE)
150772888R (GUIDE RESSORT ANNEAU REGULATION.1)
150B79090R (BILLE CLAPET DECHARGE POMPE HUILE.1)
152438377R (RESSORT CLAPET DECHARGE.1)
150881235R (BOUCHON CLAPET REGULATION)
150896419R (ETANCHEITE ANNEAU REGULATION)
150793549R (RESSORT ANNEAU REGULATION.1)
150835129R (ANNEAU CENTRAGE PALETTE)
150842446R (PALETTE ANNEAU REGULATION)
150467383R (PIGNON COMMAND POMPE HUILE)
150478409R (ARBRE ENTRAINEMENT POMPE HUILE)
150789593R (ANNEAU REGULATION POMPE HUILE)
150127302R (CORPS POMPE HUILE MOTEUR)
150821579R (GUIDE PALETTE ANNEAU REGULATION.1)
150B86288R (AJUTAGE ANNEAU REGULATION POMPE HUILE)
150850609R (CLAPET REGULATION POMPE HUILE.1)
152410637R (ELECTROVANNE DEBIT HUILE.1)
150158388R (COUVERCLE POMPE HUILE)
150985273R (RESSORT CLAPET REGULATION POMPE HUILE.1)
150932820R (VIS 2 FIX POMPE HUILE)
150873769R (GOUPILLE CLAPET REGULATION.1)
150881235R (BOUCHON CLAPET REGULATION)

Figure 2.3: Exploded View of a Variable Displacement Oil Pump.

The production lines have a combined yearly throughput of 1 616 112 pumps, with each line accounting for roughly half of the sum amount. Line 2 shows a greater level of automated processes, compared to Line 1. However, the processes themselves, apart from minor differences, are fundamentally the same. Figure 2.4 provides an overview of

VDOP production. Operations relative to Line 1 are shown highlighted in yellow, and to Line 2 highlighted in green. Components that make up the pump's housing are fed into the production line in the areas pointed by the arrows marked *Corpo* and *Tampa* - Body and Cover, respectively.

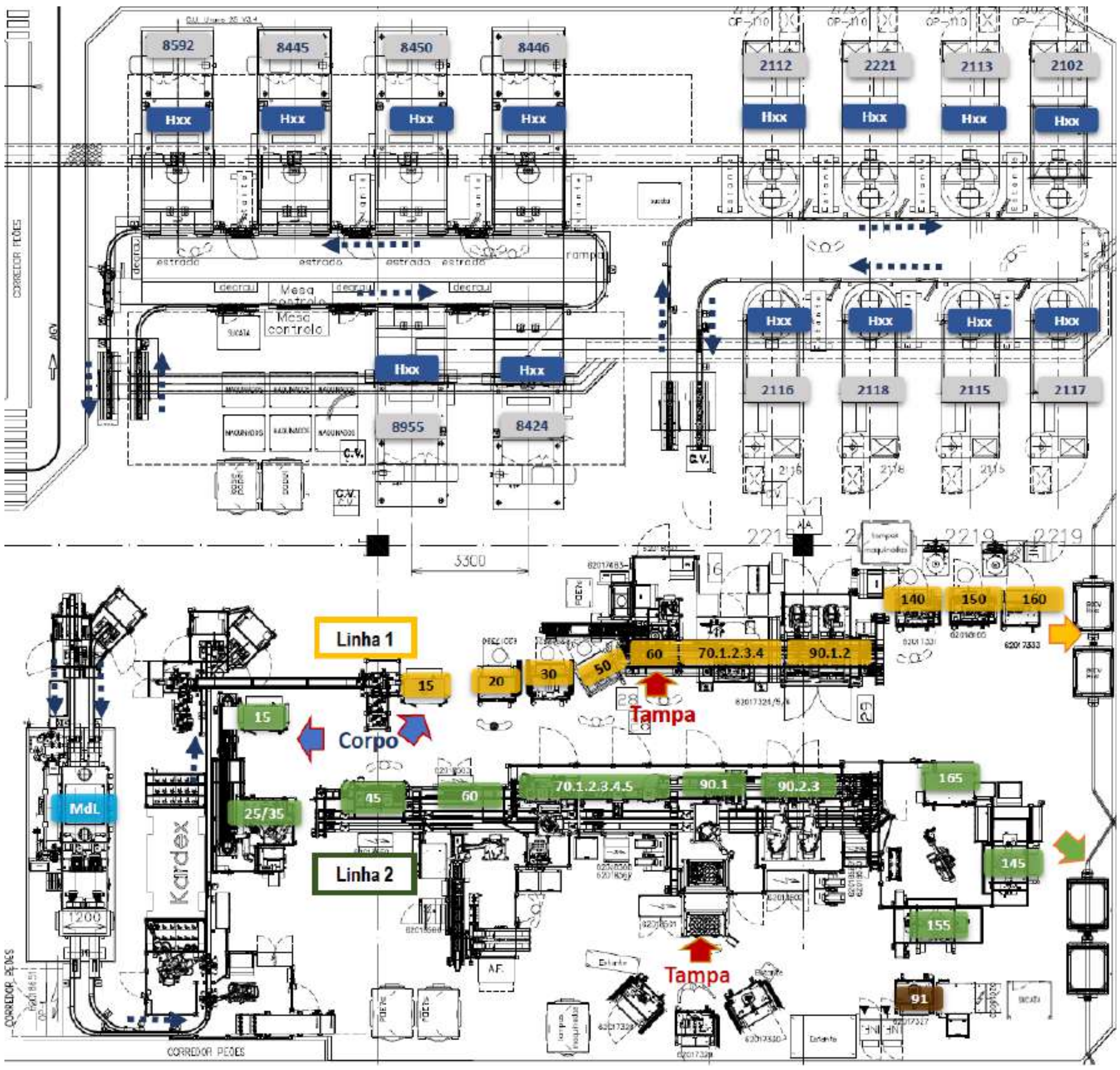


Figure 2.4: Overview of the Variable Displacement Oil Pump Production Line.

A description of each operation is present in figure 2.5. This diagram provides a more accurate recollection of the operations that take place at Line 2, although most of it is directly translatable into Line 1, apart from a few added manual tasks, due to its less automated nature.

Operation 91, highlighted in brown in figure 2.4, is a *Retouche* station, meaning a re-working station for parts with defects that are deemed fixable and can be retouched and corrected. Three operations are absent from the figure 2.4: OP 110, 120 and 130. Operations 110 and 120 involve the machining procedure of the pump's body and cover. OP 130 is a washing and drying process of the machined parts.

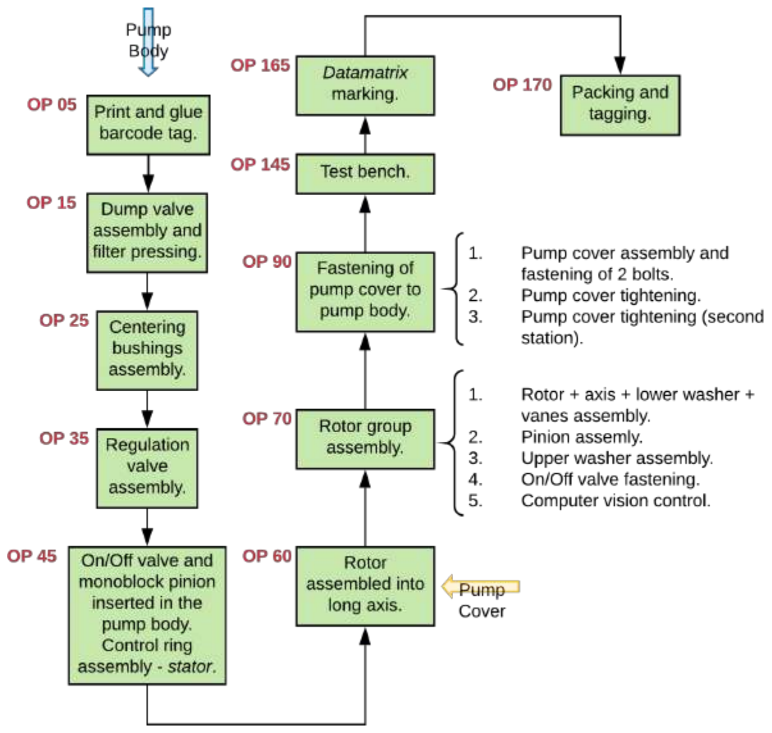

Pump Body
OP 05
Print and glue barcode tag.
OP 15
Dump valve assembly and filter pressing.
OP 25
Centering bushings assembly.
OP 35
Regulation valve assembly.
OP 45
On/Off valve and monoblock pinion inserted in the pump body. Control ring assembly - *stator*.
OP 60
Rotor assembled into long axis.
Pump Cover
OP 70
Rotor group assembly.
1. Rotor + axis + lower washer + vanes assembly.
2. Pinion assemly.
3. Upper washer assembly.
4. On/Off valve fastening.
5. Computer vision control.
OP 90
Fastening of pump cover to pump body.
1. Pump cover assembly and fastening of 2 bolts.
2. Pump cover tightening.
3. Pump cover tightening (second station).
OP 145
Test bench.
OP 165
Datamatrix marking.
OP 170
Packing and tagging.

Figure 2.5: Characterization of Operations at VDOP line.

A wide range of suppliers are responsible for providing the production line with POEs (Externally Worked Parts) and raw materials. These suppliers include ALPEN, BONTAZ, PIERBURG, SIMO, SOFRASTOCK and SONAFI.

2.3 Traceability System at the VDOP Line

When a defective oil pump is returned, both Quality Services and Logistics Services at Renault CACIA must be capable of identifying other oil pumps likely to possess the same type of defect, as well as the engine to which they were later coupled. To achieve this, each batch of completed parts (VDOPs) is accompanied by a GALIA tag, figure 2.6. Likewise, whenever a POE batch is received, it is marked with a GALIA tag. This tag identifies the part reference, barcode, supplier, batch and number of parts within the batch.

The implemented traceability system at the VDOP line is carried out on a part-by-part basis, meaning that it is possible to access the production parameters of each and every part produced. Firstly, the operator at OP 110 is responsible for registering the GALIA tag present at the POE batch. A barcode - figure 2.7 - is then glued onto the pump body and pump cover. This will allow for each individual pump to be traced along the production line.

During each subsequent operation, the barcode is read at the start of the station. After all production processes are completed, and the pump passes quality requirements at the test bench, it is marked with a *Datamatrix* tag. This tag contains the completed part's reference, year of fabrication and part number. Parts produced in Line 1 have

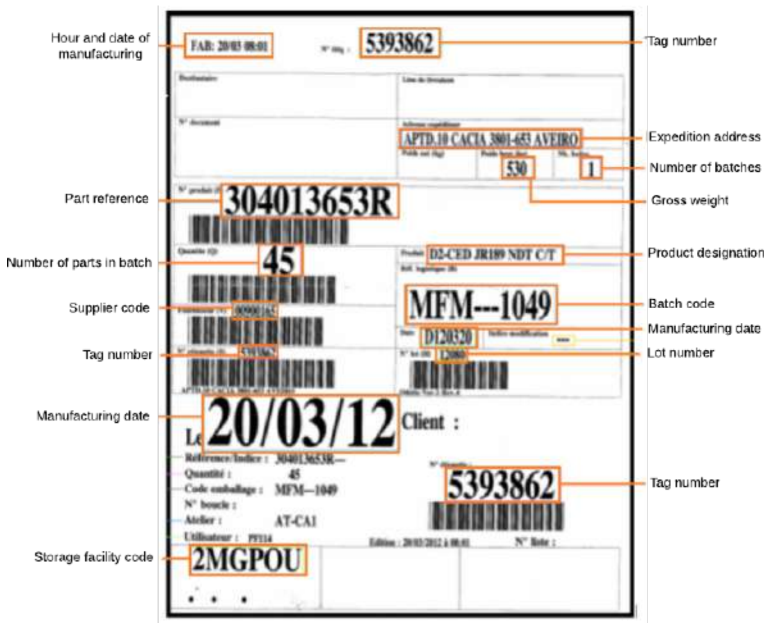


Figure 2.6: Example of a GALIA Tag for Finished Products. Adapted from: [3].

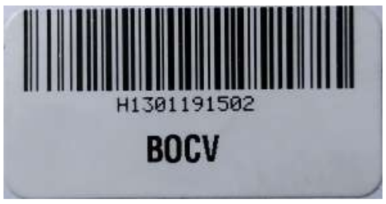


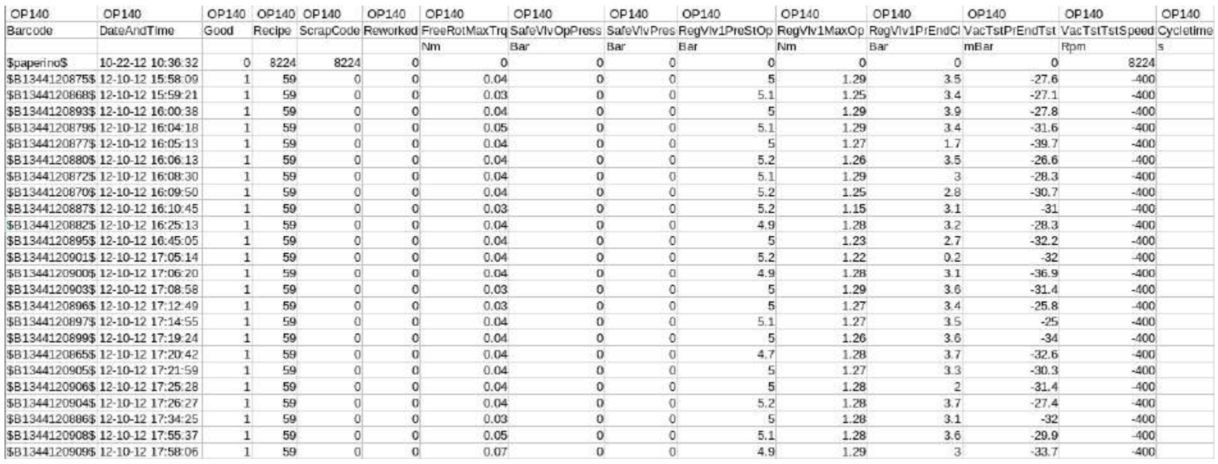
Figure 2.7: Barcode Tag Used in the VDOP Production Line.

a part number raging from 0001 to 4999, while parts produced in Line 2 have a part number raging from 5000 to 9999. In OP 170 a new GALIA tag is emitted after each container is packed. Packing occurs for each container produced in the shift, and not at the end of the shift. All the tracing data is stored in a central server, with a redundant hard-drive responsible for data back-up.

2.3.1 Gathered Data at VDOP Line

The type of data gathered will depend on the kind of process conducted in a given manufacturing cell. For example, OP140 - analogous to OP145 of diagram 2.5 - collects information regarding the time and date in which the process was initiated and time taken to completion; if the part passed quality control or not; respective scrap code if part is not accepted; if the part was previously reworked; and several pressure and torque readings, corresponding to each one of the different tests conducted. Each one of these database entries is associated with each part's barcode tag, used for tracing in intermediate production stages. Figure 2.8, shows part of the used data tables.

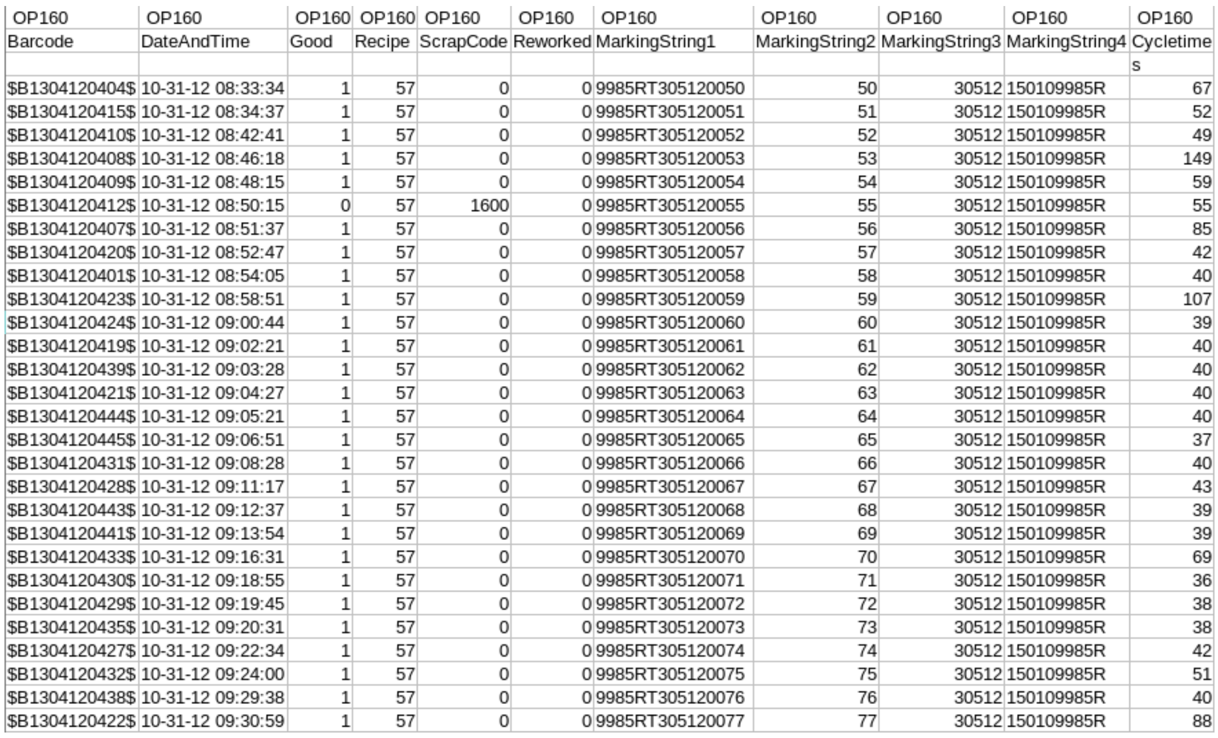
A different example is found at OP160 - analogous to OP165 of diagram 2.5 - in which *DataMatrix* marking of the finished product is done. The timestamp, scrap code, reworking notice, quality control status and cycle-time metrics are maintained from the



OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140	OP140
Barcode	DateAndTime	Good	Recipe	ScrapCode	Reworked	FreeRotMaxTrq	SafeVlvOpPress	SafeVlvPres	RegVlv1PreStOp	RegVlv1MaxOp	RegVlv1PrEndCl	VacTstPrEndTst	VacTstTstSpeed	Cycletime
						Nm	Bar	Bar	Bar	Nm	Bar	mBar	Rpm	s
$paperino$	10-22-12 10:36:32	0	8224	8224	0	0	0	0	0	0	0	0	8224	
$B1344120875$	12-10-12 15:58:09	1	59	0	0	0.04	0	0	5	1.29	3.5	-27.6	-400	
$B1344120868$	12-10-12 15:59:21	1	59	0	0	0.03	0	0	5.1	1.25	3.4	-27.1	-400	
$B1344120893$	12-10-12 16:00:38	1	59	0	0	0.04	0	0	5	1.29	3.9	-27.8	-400	
$B1344120879$	12-10-12 16:04:18	1	59	0	0	0.05	0	0	5.1	1.29	3.4	-31.6	-400	
$B1344120877$	12-10-12 16:05:13	1	59	0	0	0.04	0	0	5	1.27	1.7	-39.7	-400	
$B1344120880$	12-10-12 16:06:13	1	59	0	0	0.04	0	0	5.2	1.26	3.5	-26.6	-400	
$B1344120872$	12-10-12 16:08:30	1	59	0	0	0.04	0	0	5.1	1.29	3	-28.3	-400	
$B1344120870$	12-10-12 16:09:50	1	59	0	0	0.04	0	0	5.2	1.25	2.8	-30.7	-400	
$B1344120887$	12-10-12 16:10:45	1	59	0	0	0.03	0	0	5.2	1.15	3.1	-31	-400	
$B1344120882$	12-10-12 16:25:13	1	59	0	0	0.04	0	0	4.9	1.28	3.2	-28.3	-400	
$B1344120895$	12-10-12 16:45:05	1	59	0	0	0.04	0	0	5	1.23	2.7	-32.2	-400	
$B1344120901$	12-10-12 17:05:14	1	59	0	0	0.04	0	0	5.2	1.22	0.2	-32	-400	
$B1344120900$	12-10-12 17:06:20	1	59	0	0	0.04	0	0	4.9	1.28	3.1	-36.9	-400	
$B1344120903$	12-10-12 17:08:58	1	59	0	0	0.03	0	0	5	1.29	3.6	-31.4	-400	
$B1344120896$	12-10-12 17:12:49	1	59	0	0	0.03	0	0	5	1.27	3.4	-25.8	-400	
$B1344120897$	12-10-12 17:14:55	1	59	0	0	0.04	0	0	5.1	1.27	3.5	-25	-400	
$B1344120899$	12-10-12 17:19:24	1	59	0	0	0.04	0	0	5	1.26	3.6	-34	-400	
$B1344120865$	12-10-12 17:20:42	1	59	0	0	0.04	0	0	4.7	1.28	3.7	-32.6	-400	
$B1344120905$	12-10-12 17:21:59	1	59	0	0	0.04	0	0	5	1.27	3.3	-30.3	-400	
$B1344120906$	12-10-12 17:25:28	1	59	0	0	0.04	0	0	5	1.28	2	-31.4	-400	
$B1344120904$	12-10-12 17:26:27	1	59	0	0	0.04	0	0	5.2	1.28	3.7	-27.4	-400	
$B1344120886$	12-10-12 17:34:25	1	59	0	0	0.03	0	0	5	1.28	3.1	-32	-400	
$B1344120908$	12-10-12 17:55:37	1	59	0	0	0.05	0	0	5.1	1.28	3.6	-29.9	-400	
$B1344120909$	12-10-12 17:58:06	1	59	0	0	0.07	0	0	4.9	1.29	3	-33.7	-400	

Figure 2.8: Partial Example of a Data Table for OP140 at the Renault CACIA VDOP Production Line.

previous operations, however, in this table, the intermediate barcode tag is associated with the product's final *DataMatrix* marking string. Figure 2.9, shows the used data tables at this operation.



OP160	OP160	OP160	OP160	OP160	OP160	OP160	OP160	OP160	OP160	OP160
Barcode	DateAndTime	Good	Recipe	ScrapCode	Reworked	MarkingString1	MarkingString2	MarkingString3	MarkingString4	Cycletime
										s
$B1304120404$	10-31-12 08:33:34	1	57	0	0	9985RT305120050	50	30512	150109985R	67
$B1304120415$	10-31-12 08:34:37	1	57	0	0	9985RT305120051	51	30512	150109985R	52
$B1304120410$	10-31-12 08:42:41	1	57	0	0	9985RT305120052	52	30512	150109985R	49
$B1304120408$	10-31-12 08:46:18	1	57	0	0	9985RT305120053	53	30512	150109985R	149
$B1304120409$	10-31-12 08:48:15	1	57	0	0	9985RT305120054	54	30512	150109985R	59
$B1304120412$	10-31-12 08:50:15	0	57	1600	0	9985RT305120055	55	30512	150109985R	55
$B1304120407$	10-31-12 08:51:37	1	57	0	0	9985RT305120056	56	30512	150109985R	85
$B1304120420$	10-31-12 08:52:47	1	57	0	0	9985RT305120057	57	30512	150109985R	42
$B1304120401$	10-31-12 08:54:05	1	57	0	0	9985RT305120058	58	30512	150109985R	40
$B1304120423$	10-31-12 08:58:51	1	57	0	0	9985RT305120059	59	30512	150109985R	107
$B1304120424$	10-31-12 09:00:44	1	57	0	0	9985RT305120060	60	30512	150109985R	39
$B1304120419$	10-31-12 09:02:21	1	57	0	0	9985RT305120061	61	30512	150109985R	40
$B1304120439$	10-31-12 09:03:28	1	57	0	0	9985RT305120062	62	30512	150109985R	40
$B1304120421$	10-31-12 09:04:27	1	57	0	0	9985RT305120063	63	30512	150109985R	40
$B1304120444$	10-31-12 09:05:21	1	57	0	0	9985RT305120064	64	30512	150109985R	40
$B1304120445$	10-31-12 09:06:51	1	57	0	0	9985RT305120065	65	30512	150109985R	37
$B1304120431$	10-31-12 09:08:28	1	57	0	0	9985RT305120066	66	30512	150109985R	40
$B1304120428$	10-31-12 09:11:17	1	57	0	0	9985RT305120067	67	30512	150109985R	43
$B1304120443$	10-31-12 09:12:37	1	57	0	0	9985RT305120068	68	30512	150109985R	39
$B1304120441$	10-31-12 09:13:54	1	57	0	0	9985RT305120069	69	30512	150109985R	39
$B1304120433$	10-31-12 09:16:31	1	57	0	0	9985RT305120070	70	30512	150109985R	69
$B1304120430$	10-31-12 09:18:55	1	57	0	0	9985RT305120071	71	30512	150109985R	36
$B1304120429$	10-31-12 09:19:45	1	57	0	0	9985RT305120072	72	30512	150109985R	38
$B1304120435$	10-31-12 09:20:31	1	57	0	0	9985RT305120073	73	30512	150109985R	38
$B1304120427$	10-31-12 09:22:34	1	57	0	0	9985RT305120074	74	30512	150109985R	42
$B1304120432$	10-31-12 09:24:00	1	57	0	0	9985RT305120075	75	30512	150109985R	51
$B1304120438$	10-31-12 09:29:38	1	57	0	0	9985RT305120076	76	30512	150109985R	40
$B1304120422$	10-31-12 09:30:59	1	57	0	0	9985RT305120077	77	30512	150109985R	88

Figure 2.9: Example of a Data Table for OP160 at the Renault CACIA VDOP Production Line.

Chapter 3

Blockchain Concepts and Supporting Technologies

3.1 Supply Chain

A *Supply Chain* is a global network of organizations, people, information, resources and activities that supply a firm with goods and services [5] in order to fulfill customers' needs. Through the functions of marketing, operations and management, value is created for the customer, however seldom do firms create this value by themselves, relying on a variety of suppliers who provide everything from raw materials to accounting services. The primary goal of Supply Chain Management (SCM) is to ensure the most efficient use of the supply chain network's resources whilst fulfilling customer demands.

3.2 Traceability

Traceability is the capability of participants to trace products throughout the supply chain by means of either the product and/or container identifiers in a forward and/or backward direction [6]. Complete traceability allows for the entirety of the product's history to be known, from its inception to retail.

3.3 Industry 4.0

Industrial IoT created the never seen before possibility to merge the traditional operational technology found in most manufacturing plants, with enterprise grade information technology systems. Connected industrial processes, large volume real-time data monitoring and analysis coupled with complex analytical and event processing algorithms allowed for the first appearance of flexible production systems. This marks the beginning of Industry 4.0, presenting itself as an opportunity for more efficient manufacturing process, higher integration of the supply chain, preventative maintenance of the shop-floor and added flexibility to suppress ever-changing customer demands [7].

Industry 4.0 is a concept brought forth by the German Academy of Science and Engineering, standing for the integration of industrial manufacturing and information technology based on cyber-physical systems, IoT and Cloud-Based Manufacturing (CBM) [7]. One of the major goals of Industry 4.0 is to enable decentralized decision making and to allow for self-organizing production systems. Unlike any previous industrial revolution, all principals defended by Industry 4.0 are spread through every phase of the product life-cycle as well as the entirety of value-chain, from the company to suppliers and even final costumers. The following sections discuss some of the major technological trends considered instrumental in shaping the fourth Industrial Revolution [8].

3.3.1 Big Data and Analytics

As the amount of gathered data increases, so does the need to process and organize it in a coherent manner. Data is generated in one of the multiple phases of the production process, with correlations being analysed, allowing to identify faults and streamlining the manufacturing process. Big data and analytics is based upon the six Cs:

Connection Connection of several sensors and networks together.

Cloud Computing

Cyber Cyber-physical models.

Content/Context

Community Collaboration and sharing of data between multiple stakeholders.

Customization

3.3.2 Internet of Things

Internet of Things, Industrial Internet of Things or simply *Things*, enable the connection of sensors, actuators, and devices to network allowing for the collection, exchange and analysis of generated information [9]. In spite of the numerous advantages, converting a traditional manufacturing process to a Smart Manufacturing process is not a straightforward operation as several implementation challenges exist. For instance, ageing machinery existing in current manufacturing operations might lack networking capabilities, being refereed to as *brownfield*. Programming these existing machines to perform additional data transport might not possible or simply deemed too costly.

Ideally, only up-to-date Programmable Logic Controller (PLC) and Industrial PC (IPC) would be present in the shop-floor, forming a *greenfield* solution, but this is even more unlikely as industrial equipment is expected to endure over a decade in use, and, as such, at least some brownfield will always be present. A possible solution would be a *Gateway Aggregator*: use of external sensors or an aggregation point inside the control loop, connected to a gateway (e.g., an IoT module). This permits data extraction from the real-time system, without the need to reprogram or retro-fit the PLC/IPC.

Another limiting factor is the proven lack of security associated with IoT devices, with many exploits being already of note [1] [10]. This is one of the major concerns of the Industrial Internet Consortium (IIC). Furthermore, very few IoT devices are designed and engineered to such standard as to ensure reliability and fault tolerance for a continuous manufacturing application. New network standards, such as LoRaWAN and NB-IOT are being created for the deployment of low powered networks based upon small embedded devices. Other standards developed allow for the communication between IoT peers, such as the MQTT standard. Furthermore, advances have been made towards interconnectivity of these devices with data storage systems (e.g., traditional remote server based or cloud services based databases). Coupled with new data analysis tools, like Eclipse BIRT, Apache Hadoop, Apache Spark or Apache Storm, the real-time processing and monitoring of the immense amount of data being gathered be the IoT is possible.

An IoT architecture is an aggregation of multiple IoT devices, using some sort of gateway and communicating through a network to a common enterprise back-end server. Analogous to what is seen in web infrastructures, dominated by the LAMP (Linux - Apache HTTP Server - MySQL - PHP) stack, three different software stacks can be identified as crucial to any IoT architecture [9].

Stack for Constrained Devices

An IoT platform first starts with the IoT devices themselves. Usually very constrained in terms of power, computational prowess and size, Things are normally based on Microcontroller (MCU) architectures. As such, many of these run *bare-metal* or with simple real-time operating systems.

Things need to interact with the physical world in order to gather data (through GPIOs for instance), a software layer that provides hardware abstraction is needed. Communication support is required for networking. Several different protocols can be present, both wireless (Bluetooth, MQTT) and wired (serial connections like Universal Serial Bus (USB), RS232, Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C)). Lastly, as deployment of Things is usually done in large numbers, remote management of each unit is fundamental for system maintenance.

Stack for Gateways

Either a physical or a software gateway manages connectivity between IoT devices and each other and to external networks. Usually featuring general purpose operating systems (e.g., Linux), IoT gateways have the ability to run application code, sometimes supporting high-level general use languages like Java, Python or Node.js. This makes these devices suitable for data processing *at the edge* - Edge Computing - with sometimes even having some type of storage capability. In essence, gateways are the backbone of the architecture managing data, remotely managing the network and supporting the communications, whilst needing to ensure reliability, security and confidentiality.

Stack for Cloud Platforms

IoT cloud platforms operate from cloud infrastructures, being the most high-level layer of the stack. Besides needing to ensure connection to IoT gateways, this stack needs to be able to support the large volume of data - *Big Data* - created by the IoT. Analytics, user interfaces and web front-end might be integrated onto cloud platforms.

Deployment of these architectures must feature loosely coupled, modular and platform-independent stacks. Any given type of hardware or cloud service should be compatible with the architecture, allowing for the horizontal and vertical scaling of the platform, aligning it with enterprise needs at any given time. IoT is still one of the major drivers of Industry 4.0 allowing for implementations of highly flexible production systems approaching ideal scenarios of *lot size 1.*

3.3.3 Autonomous Robotics

Although robotics is commonplace in industry for several decades, robots are subject to improvement and evolution towards increasing self-sufficiency and autonomy. Departing from just being tools used by humans, robots are quickly becoming integral work units that function besides humans [8].

3.3.4 Simulation

Industry 4.0 allows for the diminishing of product development time, whilst increasing production optimization. Employment of simulation modelling and testing by the creation of a Digital Twin replaces trial and error experimentation in process testing.

3.3.5 Additive Manufacturing

Increasingly higher demand for more customized products produced in very small batches, along with a need for the reduction of prototyping times has popularized addictive manufacturing technologies. Addictive manufacturing is characterized by the creation of three-dimensional objects by the addiction of layer-upon-layer of material.

Used materials are commonly of metallic or polymeric origin, and with each layer's thickness typically in the order of the tenths of millimetre. One currently very popular example is 3D printing.

3.4 Cryptography Notions

3.4.1 Hashing Functions

As defined by Menezes et al. [11], a hash function is "*a computationally efficient* (one-way) *function mapping binary strings of arbitrary length to binary strings of some fixed length, called hash-values*". In other words, hash functions allow the generation of fixed-length *hash-values* or *digests* that are representative of any given input string, figure 3.1. Furthermore, for a hash function to be useful in a cryptographic context, it must have the following basic properties [11]:

Computationally Efficient Hash functions' mathematical labour must be completed in shorts amounts of time.

Deterministic For any given input, if the same input is given twice, the same output has to be obtained from the hash function.

Pre-Image Resistant Given a specific hash value y, it has to be computationally infeasible to find an input (pre-image) x such that $h(x)=y$.

Collision Resistant Has to be computationally infeasible to find two different inputs that return the same output (i.e., two colliding inputs x and y such that $h(x)=h(y)$).

There are several solutions for hashing functions, being the most relevant:

1. Secure Hash Algorithm (SHA)
2. RACE Integrity Primitives Evaluation Message Digest (RIPEMD)
3. Message Digest Algorithm 5 (MD5)
4. BLAKE2

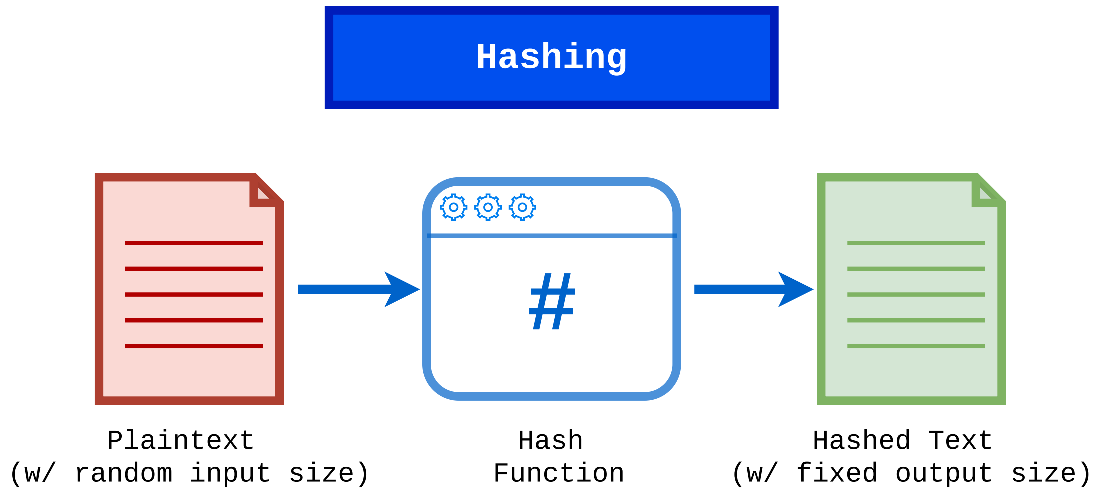


Figure 3.1: Exemplification of the Hashing Process.

Currently, the most broadly used method in a blockchain context is a subset of SHA-2, SHA-256, where the extension to the "SHA" name reefers to the output value in bits. In this case, a SHA-256 generated hash-value will consist of 256 bits (32-bytes).

Hash functions are commonly used in conjunction with digital signatures. The hash functions are applied to a large message and only the resulting hash-value is signed and not the entirety of the message. Receiving parties will then hash the incoming payload and verify that the received signature is correct for the hash-value. This saves tremendous amounts of time compared to signing the data directly, which would typically involve the splitting of data into appropriately sized blocks and signing each one individually. Other applications for hash values include virus protection in software distribution, in which the hash-value of input data (i.e., software) is verified for equality with the original hash-value in order to ensure that tampering did not occur.

SHA-2 Family of Hash Functions

Designed by the National Security Agency (NSA), and later standardized by the National Institute of Standards and Technology (NIST), the first version of a Secure Hash Algorithm was introduce as what is now known as SHA-0, with a later revision being called SHA-1. Based on the MD5 algorithm, SHA-1 was eventually considered unsafe as it is possible to find collisions due to the relatively small 160-bit result size [12]. In 2001, a new draft standard containing three new hash functions was published, being updated in 2004 to include a fourth hash function. These four hash functions are collectively referred to as the SHA-2 family of hash functions, having 224-, 256-, 384-, and 512-bit outputs. They were specifically designed for use with the key sizes found in Advanced

Encryption Standard (AES). Table 3.1 compares the maximum allowed message size for each of the SHA family's hash functions, as well as the block and output digest sizes.

Hash Function	Message Size [bits]	Block Size [bits]	Word Size [bytes]	Digest Size [bits]
SHA-224	$<2^{64}$	512	32	224
SHA-256	$<2^{64}$	512	32	256
SHA-384	$<2^{128}$	1024	64	384
SHA-512	$<2^{128}$	1024	64	512

Table 3.1: Summary of SHA Hash Functions.

3.4.2 Symmetric vs. Asymmetric Encryption

Encryption schemes usually fall within two major categories: *symmetric-key* and *asymmetric-key.* Each provide different advantages and disadvantages over the other.

Symmetric Encryption

Considering that a message is encrypted with a key, A, and then decrypted using another key, B, if it is computationally easy to determine A from B and vice-versa, then it is said to be symmetric-key encryption [11]. In most practical scenarios, both keys are indeed the same (i.e., $A{=}B$), hence the term *symmetric.*

Most symmetric-key encryption techniques are based on *block-ciphers*, in which the message is broken up into strings (called *blocks*) of fixed length, with each block being encrypted one at a time. The current generation of block ciphers has a block size of 128 bits (16 bytes) [12]. In the very likely scenario that the message to encrypt is not a multiple of 16, padding is used as to satisfy length requirements, whilst still being reversible in order to determine the original message. Multiple block long messages require the use of a *block Cipher mode* to establish how the various blocks of data are going to be encrypted. Two of the most common methods used are Electronic Codebook (ECB) and Cipher Block Chaining (CBC).

ECB is the simplest method with each block of data being encrypted separately. Serious weaknesses are found in this method, seeing that for two equal blocks of data, the Ciphertext blocks will also be identical. For highly repetitive payloads, information leakage risks are high. Alternatively, CBC is presented, being one of the most widely used block cipher modes. The drawbacks found in the ECB method are avoided by a *XOR* operation of each Plaintext block with the previous ciphered block. This will create "randomized" blocks of data, that unlike what is seen in ECB mode, will always be different, even with each block's payload being the same. However, there is still the question of which value to use for the first operation, when no previous block exists. This value is called an Initialization Vector (IV).

The most pressing disadvantage found in symmetric encryption methods is referred to as the *key distribution problem.* As all parties within a communication channel have to know the set of encryption/decryption transformations used (i.e., the key used to cipher the data), the issue of an efficient method to agree upon and exchange keys securely arouses.

Asymmetric Encryption

Asymmetric or *Public Key* encryption starts by a party, A, generating a pair of keys: a private key and a public key. The public key is then published over the network, being universally accessible to everyone. A different party, B, may now use the publicly-available public key of A to encrypt the data he wishes to send. Finally, A will decrypt the data by using its own private key. This method solves the distribution of keys problem, as different keys are used for encryption and decryption with one of them always being kept private. However, this method is highly inefficient and expensive.

Most real-world applications rely on a combination of symmetric and asymmetric encryption, using public keys to establish the first contact with the other party and define secret keys, that will in turn encrypt the actual payload between them. This allows to take advantage of the flexibility of public keys and the efficiency of symmetric keys. *Digital Signatures* are the public key counterpart of message authentication codes [12]. Their working shares much of the same philosophy as asymmetric encryption, given that a party might sign a message using its private key, for another party to verify the signature in the message using the corresponding public key and thus assuring the provenance of data. For further explanation of digital signatures for message authentication see section 3.5.2.

3.5 Network Security

Every network has to fulfil three different requirements - Confidentiality, Integrity and Security (CIA) - in order to be classified as fully secured [13]:

Confidentiality Confidentiality is the protection or concealment of information, not only applied to the storage of information, but also during transmission.

Integrity Information suffers constant alterations with integrity meaning that they can only be carried out by authorized entities and through authorized mechanisms. Integrity violations might not only be results of malicious acts; interruptions in the system may also cause unwanted changes in information.

Availability Information is only useful if readily available to the authorized entities, as such unavailability of information is as harmful to an organization as the lack of confidentiality and integrity.

3.5.1 Attacks on the Network

Using the taxonomy proposed by Forouzan [13], attacks can be divided into three groups related to the security goals, as seen in figure 3.2.

Attacks Threatening Confidentiality

Snooping Unauthorized access to or interception of data. To prevent snooping practices, encryption techniques are applied, making the data unintelligible.

Traffic Analysis Monitoring of online traffic might allow attackers to gain information on the network, even if data is encrypted. As seen in section 3.4.2, use of ECB still allows for data leakage in highly repetitive payloads.

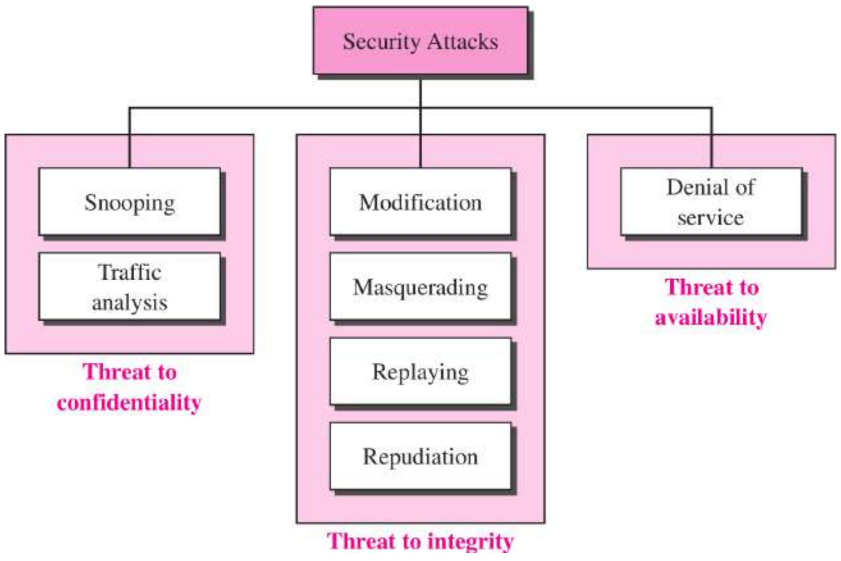


Figure 3.2: Taxonomy of Attacks With Relation to Security Goals. Sourced from: [13]

Attacks Threatening Integrity

Modification Modification or deletion of data after being intercepted.

Masquerading The attacker impersonates someone else, fooling the victim.

Replaying An intercepted message is re-sent by the attacker.

Repudiation Either the sender or the receiver of the communication denies the contents of the message sent, or denies that the message was sent altogether.

Attacks Threatening Availability

Denial of Service One of the most common attacks, it may lead to a system slow down or even complete interruption. Several exploits can be used to achieve this attack, such as sending several dummy requests to the server causing a crash due to heavy-load or by intercepting and deleting a data packet.

3.5.2 Message Authentication

Two common methods to authenticate messages are through the use of a Message Authentication Code (MAC) or a digital signature.

Message Authentication Code

The following figure, 3.3, depicts the working of MAC. A MAC is created by concatenating a secret key together with the message [13] - *h(K+M)*. The MAC is then sent to the recipient through an insecure channel who will separate the message from the MAC, and recalculate the MAC value. If both MAC values match, the message is considered authentic. Nested MAC, often referred to as Hashed Message Authentication

Code (HMAC), is a more complex implementation of traditional MAC, standardized by NIST.

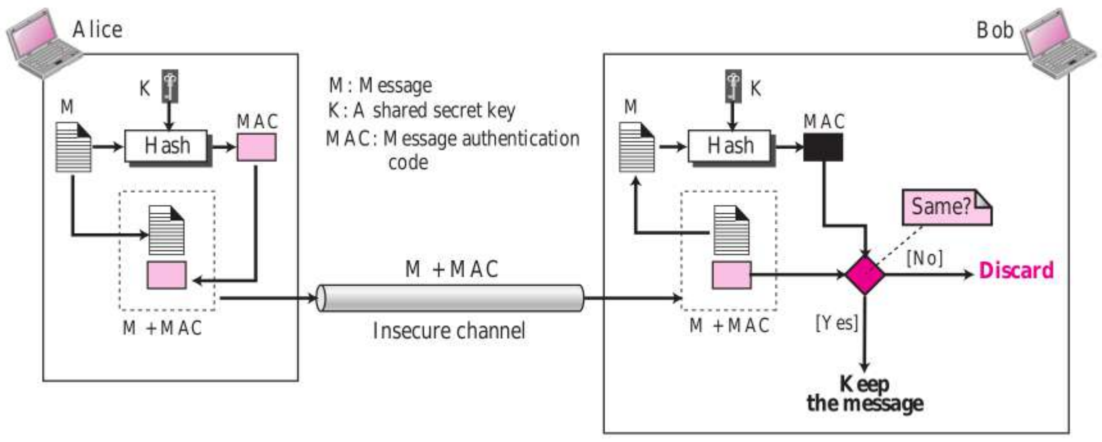


Figure 3.3: Message Authentication Code. Sourced from: [13]

Digital Signature

Contrary to MAC, which uses a variation of what can be considered symmetric encryption, digital signatures rely on a private-public key pair [13]. Many analogies can be made with digital and traditional pen-in-paper signatures, although there are fundamental differences. For instance, a conventional signature is included in a document, whilst a digital signature is usually sent separately. The process of signing messages by digital signatures - as depicted in figure 3.4 - starts by the sender using a *signing algorithm* to sign the message. Both the message and the signature are then sent to the recipient who will apply the *verifying algorithm* to the combination [13]. In case the result is true, the message is accepted as authentic.

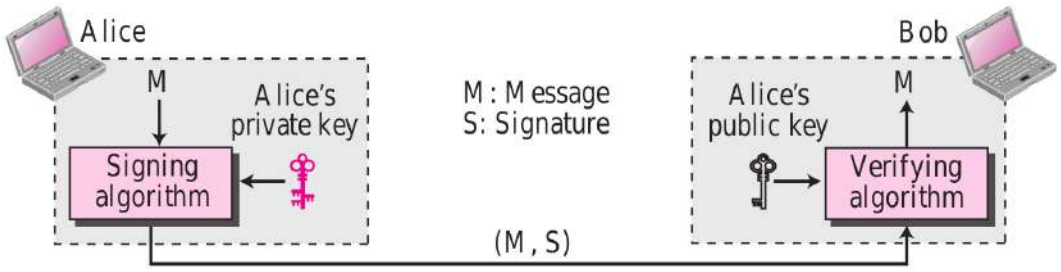


Figure 3.4: Digital Signature Process. Sourced from: [13]

While an entity's private key must be kept secret at all times, the public key sees wide distribution. There exists a clear distinction between asymmetric cryptosystems and digital signatures, as in the latter the sender's key pair is used and in the former it is the receiver's key pair. Due to inefficiencies in asymmetric-key cryptosystems dealing with long messages, typically only the digest of the message is signed.

One of the more common digital signature schemes is RSA, which uses the private/public key pair of the sender, with the sender using their own private key to sign documents and the receiver using the sender's public key for authentication. In broad terms, first both the sender and receiver agree upon a hash function to generate digests from a message, in such a way that: $D = h(m)$. The sender will then sign the resulting digest - $S = D^d mod\ n$ - and send it to the receiver who will then use the sender's public exponent to retrieve the digest - $D' = S^e mod\ n$. By re-applying the hashing algorithm to the message, the receiver can then obtain $D = h(M)$. If both resulting digests, D and D', are equal, the message is accepted as authentication was established. One of the major advantages of RSA is its ability to both be used for message signing and message encryption [12], as both operations use the same computations.

Alternatively, the Digital Signature Standard (DSS) was proposed as a more secure (yet more complicated) digital signature only scheme, by NIST. DSS uses Digital Signature Algorithm (DSA). Nevertheless, both RSA and DSS meet acceptable requirements for application in protocols such as Transport Layer Security (TLS) [14].

Recently it has been up for debate the security of public key cryptography against quantum computer attacks, with claims even reaching that RSA will be broken by a 1/7 chance by 2026 and 1/2 chance by 2031 [15]. Current quantum-resistant schemes have several limitations when compared with traditional alternatives - such as RSA - due to larger public-keys, larger signatures, or slower runtime [16]. Notwithstanding, even if large-scale quantum computing never exists, research into post-quantum cryptography is still valuable, offering protection against (non-quantum) mathematical breakthroughs that might jeopardize current scheme's security [16].

3.5.3 Key Management

Key management is an essential aspect, regardless of the type of encryption being used. For instance, in symmetric encryption, key distribution is one of its downsides due to the difficulty of communicating keys between peers, seeing that insecure channels cannot be used. Two possible solutions are either using a trusted third party to manage keys within a network, referred to as a Key-Distribution Center (KDC); or through an agreement such as the Diffie-Hellman (DH) protocol.

As for asymmetric-encryption, public keys are distributed freely across the network, but it is still required to establish its provenance, as a malicious entity could generate a public key claiming any identity. The common way to distribute public keys is through the creation of *public-key certificates* [13]. A Certification Authority (CA), usually a nationally or globally recognized organization, binds a public key to an entity, issuing a certificate. This is done by proving one's identity to the CA - using conventional identifying elements such as passports - who will then sign a certificate with its private key. As the CA's public key is well-known, it is possible to verify that the certificate used is genuine. Looking to standardize the structure of a certificate, the ITU created the *X.509* format, deterring public key fraud.

3.5.4 Transport Layer Security

Netscape developed the Secure Sockets Layer (SSL) protocol in 1994 [13] with four target objectives: cryptographic security, interoperability, extensibility, and relative efficiency [17]. Following two later revisions, SSL eventually evolved into the TLS protocol, with

SSL now being considered deprecated. Nevertheless, the term SSL/TLS is still often used. If both a client and server are capable of running SSL/TLS programs, the client may use the Uniform Resource Locator (URL) *https://...* rather than the usual *http://...* to allow Hypertext Transfer Protocol (HTTP) messages to be encapsulated in SSL/TLS packets. These protocols operate at the transport layer and above other existing protocols such as Transmission Control Protocol (TCP), offering an abstraction of secure sockets on top of existing Transmission Control Protocol/Internet Protocol (TCP/IP) sockets.

Although use cases of TLS are plentiful, including Web servers, Secure Shell (SSH) connections, and secure email servers, most websites delivering general purpose public information choose to operate without it, as TLS introduces substantial overhead due to computationally expensive RSA operations [17]. TLS has four protocols to accomplish secure connections:

Handshake Protocol Negotiates the cipher suite used to authenticate both the server and, optionally, the client. If required, it is exchanged information for building cryptographic secrets. Most TLS implementations do not certify the client.

ChangeCipherSpec Protocol Special message exchanged during the Handshake Protocol that enables use of previously negotiated cipher suite and cryptographic secrets.

Alert Protocol Used to report errors and abnormal conditions.

Record Protocol Carries messages from the other three protocols as well as from the application layer itself. The message is fragmented and optionally compressed. NIST recommends that all compression methods in TLS be disabled to protect against attacks using compression-based side channels [14].

3.6 Blockchain

3.6.1 Blockchain Architecture

A Blockchain, first proposed by Nakamoto [18] in 2008 for use in cryptocurrency Bitcoin, is based on a software network fully-distributed by it's numerous comprising peers. Each of the network's peers is allowed to track, verify and create cryptographically protected data to add to their shared ledger. In essence, this Distributed Ledger becomes a database containing all of the data gathered by all of the chain's nodes.

From a programming standpoint, it is possible to represent a blockchain as a data structure where entries (blocks) are stored and linked in sequential order [19]. In this fashion, the block immediately previous to the current block is called a parent block. All the blocks in the chain possess a parent block, with the exception being the very first entry in the chain, the genesis block, from which all the blockchain subsequently derives from. The following figure 3.5, exemplifies the basic structure of a blockchain.

The linkage between each one of the blocks is assured by storing the hash value of the previous block. Additionally, a network protocol is required to define a mode of operation for the network. This will encompass rights and responsibilities of each participant as well as providing means of communication, verification, validation and consensus [20]. Other factors such as incentive mechanisms might also be established by this protocol.

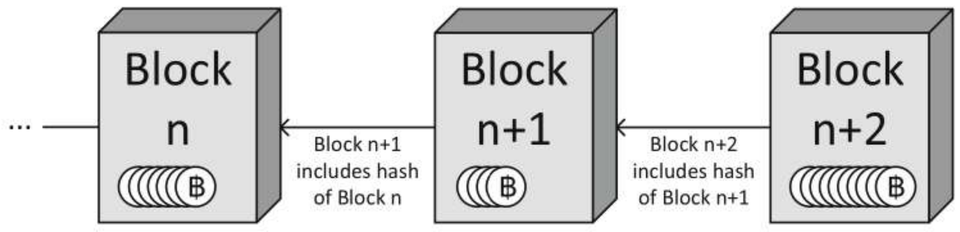

...
Block n
Block n+1 includes hash of Block n
Block n+1
Block n+2 includes hash of Block n+1
Block n+2

Figure 3.5: Basic Structure of a Blockchain, from: [20]

3.6.2 Architecture of the Block

A block is composed by the block header and the block body or payload. The block header is used to convey information about the contents of the block and in the case of a typical blockchain such as Bitcoin, it includes [21]:

Block Version Determines the set of block validation rules to follow.

Parent Block Hash 256-bit SHA256 generated hash value of the previous block to ensure linkage.

Merkle Tree Root Hash Calculated hash value of all the transactions contained within the block.

Timestamp Current timestamp, given by seconds since Epoch (Unix Time).

nBits Sets the new target threshold bellow which any new block hash will have to be in order for it to be validated.

Nonce 4-byte field, which usually starts with 0 and increases for every hash calculation. Used in networks that enforce Proof Of Work algorithms.

The following figure, 3.6, illustrates the described block structure.

The block body will consist of the transactions that occurred over the network and a transaction counter. In the case of Bitcoin, the number of transactions that can be stored in any given block will depend on the size of the block as well as the size of the transactions themselves [22].

3.6.3 Miners and Consensus Algorithms

In order to ensure that only valid transactions are authenticated, and seeing that a decentralized blockchain can not rely on a central authority for validation, consensus algorithms need to be implemented in order to maintain the integrity and security of the distributed system. *Consensus* is one of the long standing hardships of distributed computing, as it requires the formulation of agreement among a distributed number of processes [23].

The first consensus algorithm to be created was Proof of Work (PoW), designed by Nakamoto for implementation on Bitcoin, as a way to overcome the Byzantine Generals

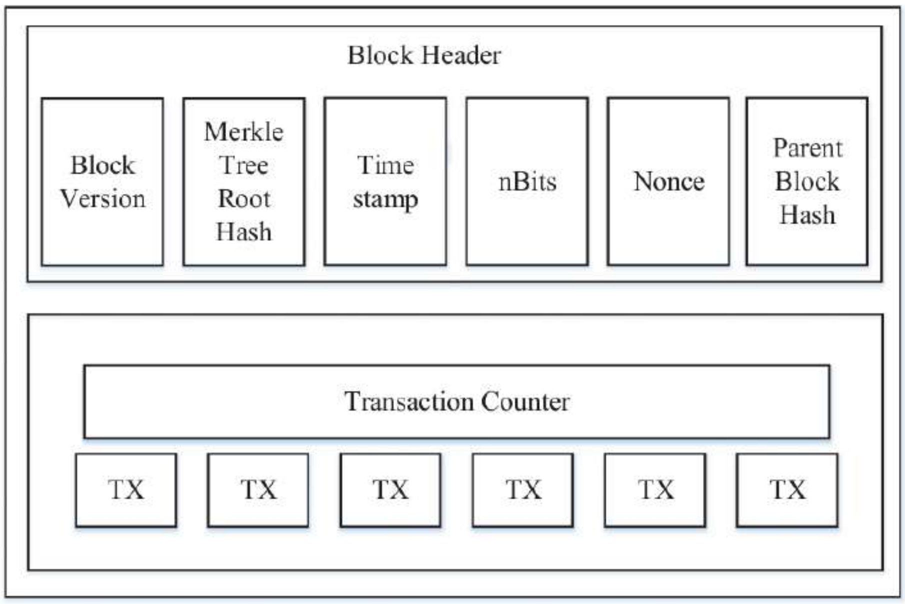


Figure 3.6: Block Structure, from: [22]

Problem (Lamport et al., 1982) [24]. Lamport et al. created an analogy between multi-processes computer systems and a group of Byzantine army generals camped with their troops around an enemy city. Communication between the generals is assured only by the means of a messenger, who must agree upon a common attack plan. The problem is to find an algorithm that allows for loyal generals to reach an agreement, impervious of traitors that might exist trying to confuse others. Table 3.2 shows a comparison between several consensus algorithms.

Proof of Work

The Proof of Work algorithm requires that each node in the network calculates through numerous attempts the new hash value of the block header. This is achieved by iterating the nonce value of the block header, until the final hash value is bellow a certain threshold. When the target value is achieved, every other node in the network must mutually confirm the correctness of the values, thus achieving consensus. The nodes calculating the hash values are called miners and procedures such as PoW is refereed to as mining. Although effective, this consensus algorithm receives criticism for being inefficient as a large amount of the computational power is wasted causing high energy costs, leading to the appearance of alternatives such as Proof of Stake (PoS).

Proof of Stake

The PoS consensus algorithm validates blocks according to the stake of each participant on the basis that whoever controls the biggest portions of currency are the least likely to attack the network. The blockchain is secured typically by pseudo-random elections that consider the node's wealth and the coins age (how long were the coins kept), but variations of this method are plentiful. Seeing that the selection is based on account balance, it is possible for a network that employs this type of algorithm to be dominated by the single richest node in the network, nevertheless, several popular cryptocurrencies are considering transitioning to this type of consensus algorithm and away from PoW, such as Ethereum [21].

Delegated Proof of Stake

Delegated Proof of Stake (DPoS) differs from both PoW and PoS in the sense that miners work together in forging new blocks, instead of competing for completion [25]. Validation of blocks is done by a delegate, elected by the stake holders of the network.

Proof of Authority

Compared to other consensus algorithms, Proof of Authority (PoA) is more efficient, supporting a higher level of performance, by virtue of lighter message exchange. PoA algorithms rely on a set of trusted nodes, called *authorities*, identified by a unique *id*. It is assumed that a majority, at least 51 %, of the authorities acting on the network are honest. Consensus in PoA is achieved through a *mining rotation* schema, which allows to fairly distributed the mining load across all participants of the network. As authorities act as sources of truth for other nodes, a central point of failure is introduced, approximating the system to a centralized one. Nevertheless, PoA is one of the all-round best performers for private blockchain networks, granting a high block output.

Practical Byzantine Fault Tolerance

Byzantine Fault-Tolerant (BFT) protocols are able to tolerate subverted nodes trying to hinder the achievement of agreement in a network [23]. One such example is Practical Byzantine Fault Tolerance (PBFT), a replication algorithm [21], in which three rounds of message exchange occur before consensus is set. Hyperledger Fabric employs this consensus algorithm, handling up to a third of malicious byzantine nodes acting on the network [21].

Characteristic	**PoW**	**PoS**	**PoA**	**DPoS**	**PBFT**
Node Identity	Open	Open	Permissioned	Open	Permissioned
Energy Efficiency	No	Partial	Yes	Partial	Yes
Tolerated Power of Adversary	<25% compute power	<51% stake	<51% malicious nodes	<51% validators	<33.3% of faulty replicas

Table 3.2: Comparison between consensus algorithms.

3.6.4 Types of Blockchain

Although multiple definitions are found, a Blockchain System is usually categorized in three different types: *Public* or *Permissionless* Blockchains, *Private* or *Permissioned* Blockchains and *Consortium* (hybrid) chains. Table 3.3 gives a comparison between different types of blockchains.

Public Blockchain

A Public Blockchain has an open network with anyone being allowed to join and fully participate by reading and writing to the shared ledger. This type of network is fully decentralized and there is no governing or sovereign entity that can tamper the data once it has been validated and appended to the blockchain. As the adhering nodes are anonymous, the blockchain is maintained by delivering economic incentives for good behaviour, as seen in cryptocurrencies like Bitcoin or Ethereum. This ensures correct operation of the network and the rejection of invalid transactions. A Public Blockchain is highly inefficient, due to the need to propagate data across a large number of nodes and due to the need to implement some sort of consensus algorithm like the ones previously refereed.

Private Blockchain

Private Blockchains are fully controlled by an organization which determines the entities that are allowed to join the network, as well as limiting their power within it. This concept of blockchain is interesting in business or enterprise applications in which the participants are identified but not necessarily fully trusting of each other.

Nodes exist in a much smaller number and are distributed locally making transaction times faster. It is also possible to add new nodes to the network to comply with current demands greatly improving the solutions scalability. A private chain is not much different to other kinds of distributed storage mechanisms [26].

Consortium Blockchain

The consortium or hybrid chain is a partially distributed (multicentre) blockchain [26]. Each generation of block is determined by pre-selected nodes. Through appropriate consensus agreements, several organizations can participate to build a joint blockchain for common goals. One of the most prominent examples is the Hyperledger Project, that will be explained in greater detail in Section 3.8.

Characteristic	**Public Blockchain**	**Private Blockchain**	**Consortium Blockchain**
Read/Write Permissions	Public	Public or restricted	Public or restricted
Immutability	Nearly impossible to tamper	Can be tampered	Can be tampered
Efficiency	Low	High	High
Centralised	No	Yes	Partial

Table 3.3: Comparison between public, private and consortium blockchains.

3.6.5 Characteristics of Blockchain [26]

Decentralization

There does not exist a central authority to tamper with the stored data. All information is immediately spread out through all the adhering nodes without third-party intervention.

Detrusting

A blockchain can perform even within an untrustworthy environment. Seeing that each block is stored in all of the participating nodes, every peer can verify the ongoing transactions. By applying hash functions - Section 3.4.1 - to link blocks of data together and through consensus algorithms, the need for mutual-trust relationships does not exist.

Transparency

Due to the blockchain's decentralized and distributed nature, all transactions that occur over the network are transparent to every node.

Immutability

By using hash-values to link data-blocks together along with timestamps, it is possible to keep a clear record and lineage of data. Traceability in these circumstances is facilitated and any change made to the network is detected immediately.

Anonymity

In typical blockchains, anonymity is assured by the use of asymmetric encryption keys that allow for data provenance to be determined, without revealing the participants identity. Further explanation on asymmetric encryption in Section 3.4.2.

Credibility

Credibility in the blockchain is established by applying consensus algorithms. These allow for conditions of complete anonymity whilst increasing the security and credibility of the transaction.

3.6.6 Smart Contracts

A Smart Contract is a portion of code that is able to run within a blockchain, containing a series of clauses that determine how the system is to perform should a given condition be met, essentially enhancing the blockchain's ability for data manipulation and asset managing. Most smart contracts are produced using specific programming languages such as Solidity, which might hinder smart contract use as it increases the technical difficulty of implementing a blockchain solution.

The Bitcoin blockchain only allows for simplified versions of smart contracts, while for instance Ethereum uses a *Touring Complete* language that is able to perform the same tasks as any other general purpose programming language [20]. This unlocks potential for a blockchain to become a general computational platform.

3.7 Previous Scholar Work

3.7.1 Blockchain Based Architecture Proposed by Hang et al.

One of the most prominent solutions found was proposed by Hang et al. [1], in which figure, 3.7, shows the conceptual solution. A large number of IoT devices, data storages, user devices and servers are linked together around a peer-to-peer blockchain network. The authors propose that data storage residing within the blockchain store environmental data as well as each of the device owners' profile. The most alluring factor of this architecture is organization by layers, allowing for a high modularity of the solution. It is possible to remove and add layers at any given time. Four major layers are presented and can be consulted in figure, 3.8.

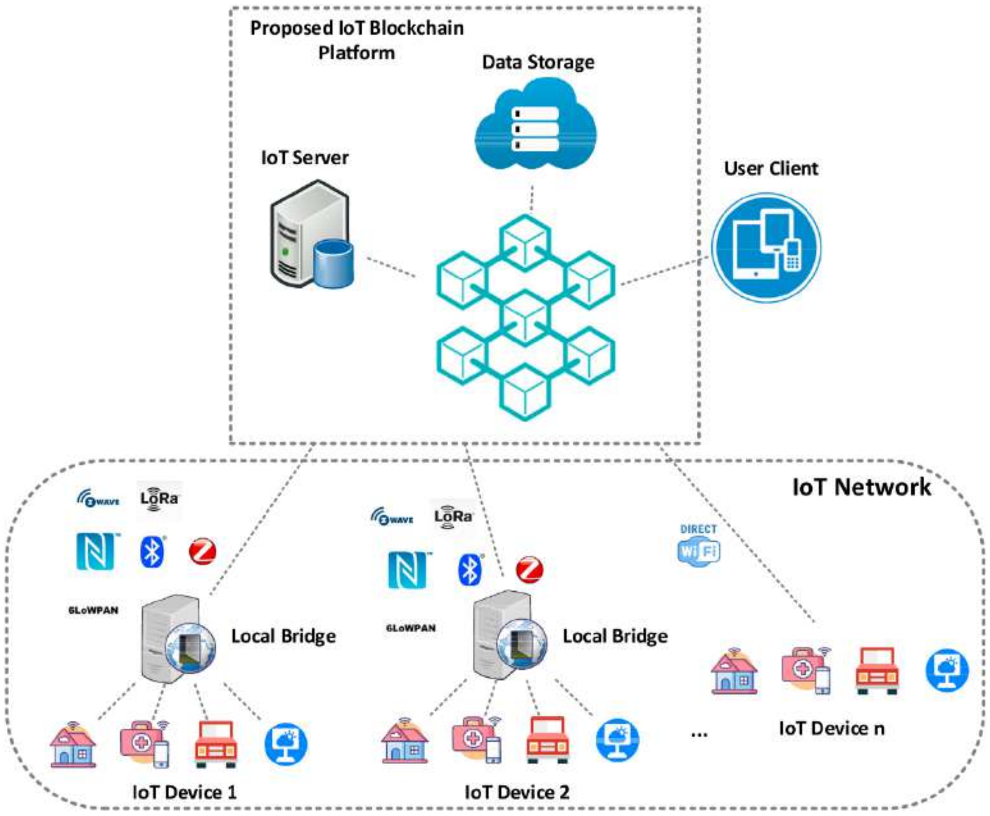


Figure 3.7: IoT Blockchain Platform Conceptual Scenario, Sourced From: Hang et al. [1]

The IoT layer manages physical devices, environmental data acquisition, data storage and controls physical actuators. A connectivity layer will then route connections to the IoT modules serving as a message broker and a network and security manager. Only by the third layer is the blockchain present. The final layer allows for the web-frontend, where a client may visualize the data stored in the ledger and manipulate physical devices. Due to the architectural complexity, latency is quite high. Figure, 3.9, shows the performance analysis done by the authors.

With a somewhat limited number of 50 devices, latency times were already as high as 2286ms for a transaction request to be sent and acknowledgement to be received by the client. For this reason only, a solution like this could not be applied to a scenario such is seen in Renault CACIA where the required response latency might be as low as two full transactions per second. Moreover, these proposed architectures are heavily based in open-source third-party platforms, such as Hyperledger Fabric, that may bring unforeseeable problems when applied to an industrial context.

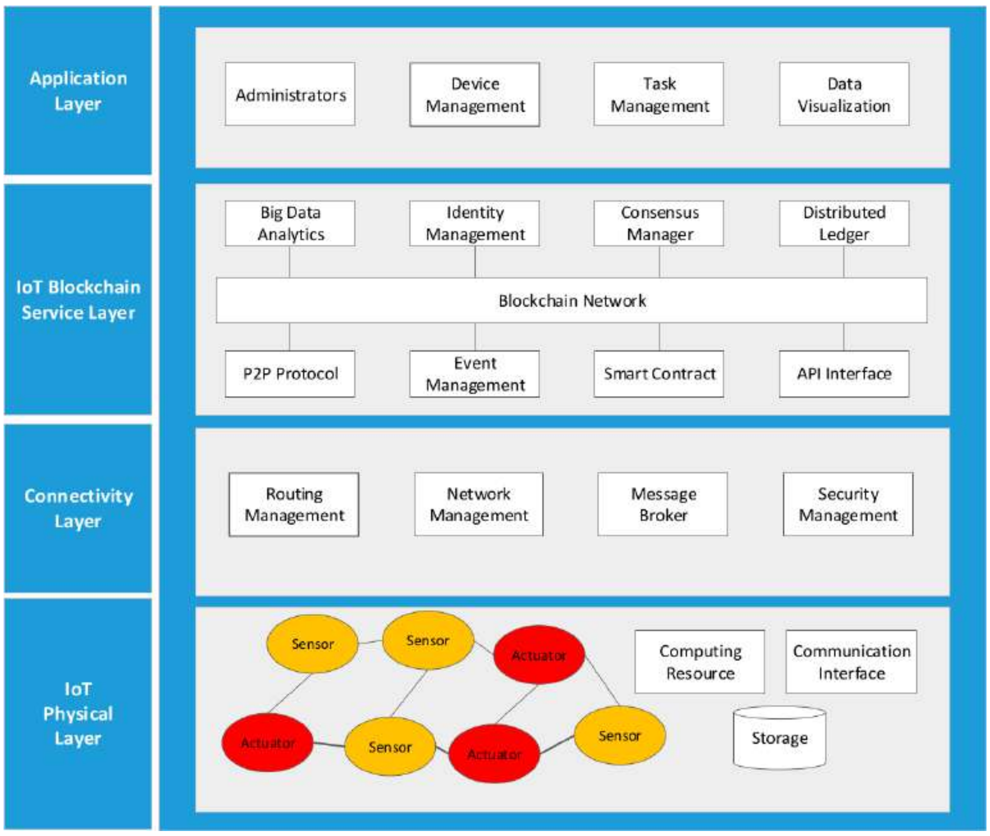


Figure 3.8: Layer-Based IoT Blockchain Platform Architecture, Sourced From: Hang et al. [1]

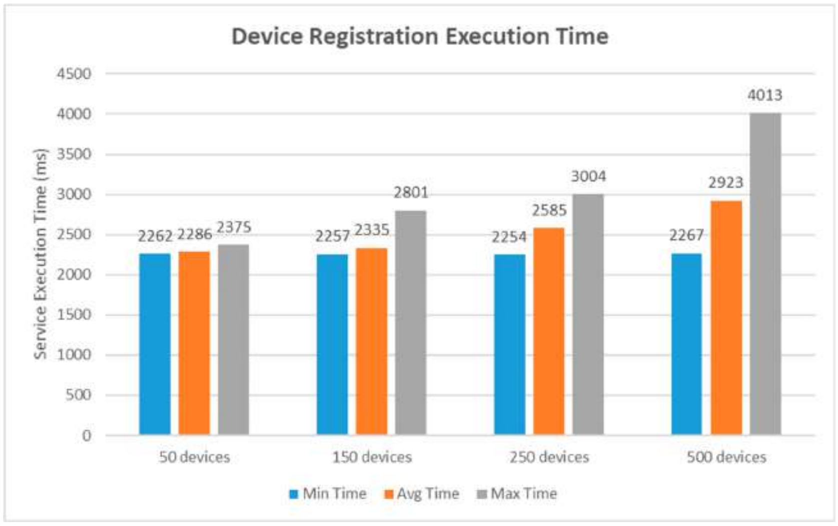


Figure 3.9: Performance Analysis Graph of Device Creation, Sourced From: Hang et al. [1]

3.7.2 Blockchain Platform for Industrial IoT Proposed by Bahga and Madisetti.

A decentralized, peer-to-peer platform for IIoT based on blockchain technologies was proposed by Bahga and Madisetti [27]. The main focus of the solution is to enhance the functionalities of CBM platforms while allowing legacy devices without networking capabilities to be integrated into cloud environments. Their proposed platform, named BPIIoT, comprises of a Single Board Computer (SBC) - that has networking capabilities - and an interface board to allow monitoring of the legacy devices. Figure 3.10 shows the proposed platform.

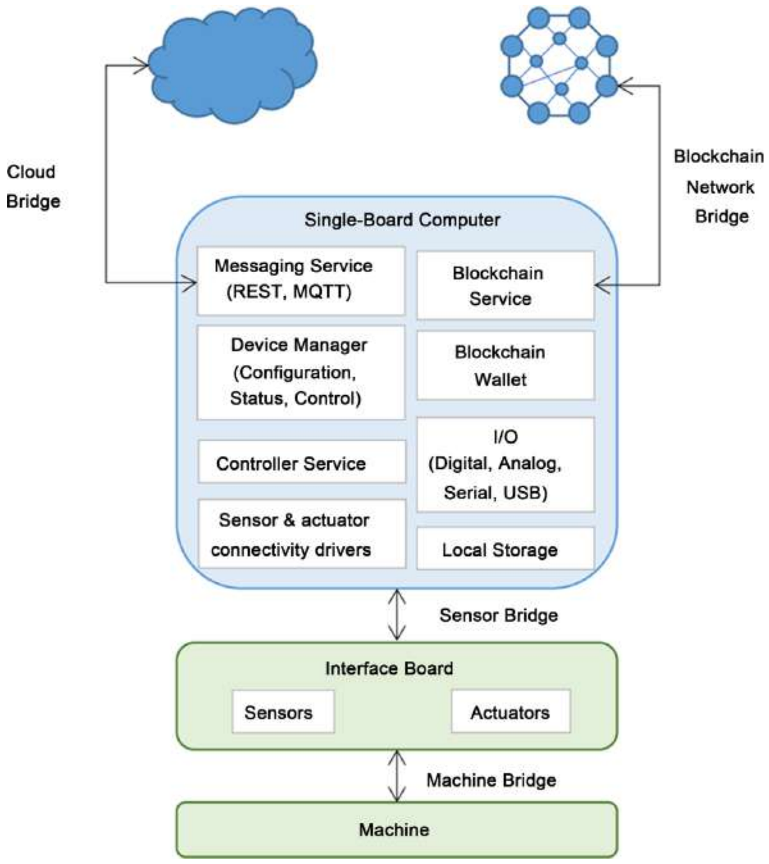

Cloud Bridge
Blockchain Network Bridge
Single-Board Computer
Messaging Service (REST, MQTT)
Blockchain Service
Device Manager (Configuration, Status, Control)
Blockchain Wallet
Controller Service
I/O (Digital, Analog, Serial, USB)
Sensor & actuator connectivity drivers
Local Storage
Sensor Bridge
Interface Board
Sensors
Actuators
Machine Bridge
Machine

Figure 3.10: Blockchain Platform for Industrial Internet of Things (BPIIoT). Sourced from: [27]

A blockchain network communicates with the SBC, sending/receiving transactions to/from the network. The authors' solution requires that each IoT device has an individual account. The interface board - based on an Arduino module - contains sensors external to that of the legacy device itself. This is an interesting concept, as is the idea of retrofitting the current shop-floor equipment with IoT capabilities. Despite this, the solution offers little in the way of realist solutions for implementations such as on-demand manufacturing or supply chain. Furthermore, much is relied on the computationally-deficient IoT devices for the processing of data, which would be a limiting factor in networks comprised of a large number of nodes. The scope of most current studies fails yet to establish concrete solutions in terms of a truly scalable, multiple node and sustainable blockchain network.

3.7.3 Blockchain-Based Solution for Security and Privacy Enhancements in Smart Factories by Wan et al.

Wan et al. [28] defend a multicentre, partially decentralized, IIoT architecture with several security features to provide better privacy protection when compared to traditional solutions. A major emphasis was given towards a higher-performing system, pointing out shortcomings related to high latencies times in platforms such as BPIIoT [27]. Figure, 3.11, illustrates the smart factory tailored solution.

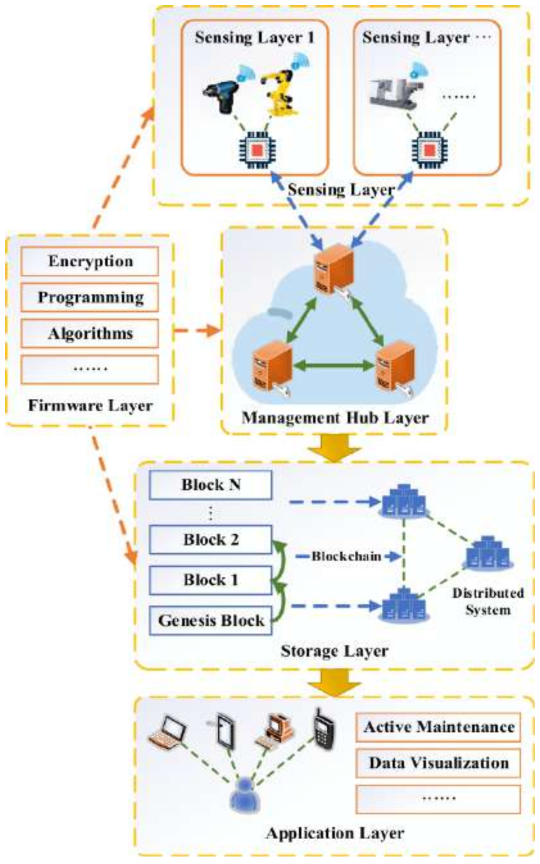


Figure 3.11: Blockchain-Based IIoT Architecture for Smart Factories. Sourced from: [28]

Five layers make up the architecture: sensing layer, management hub layer, firmware layer, storage layer and application layer. The first of these, the sensing layer, includes several sensors powered by a microcomputer, collecting and pre-processing data. The management hub will then parse, encrypt and package this data into blocks, storing it in the database. Storage layer will function as a data-centre recording the blockchain, with the firmware layer bridging all the previous layers together. Lastly, application layer retains functionalities seen in previous solutions, as it allows for the real-time monitoring by users of the gathered data. As for the cyber-security feature-set, a whitelist/blacklist mechanism was implemented to control the influx of data from the sensors into the management hub alongside a PoW consensus algorithm and a Statistical Process Control (SCP), figure 3.12.

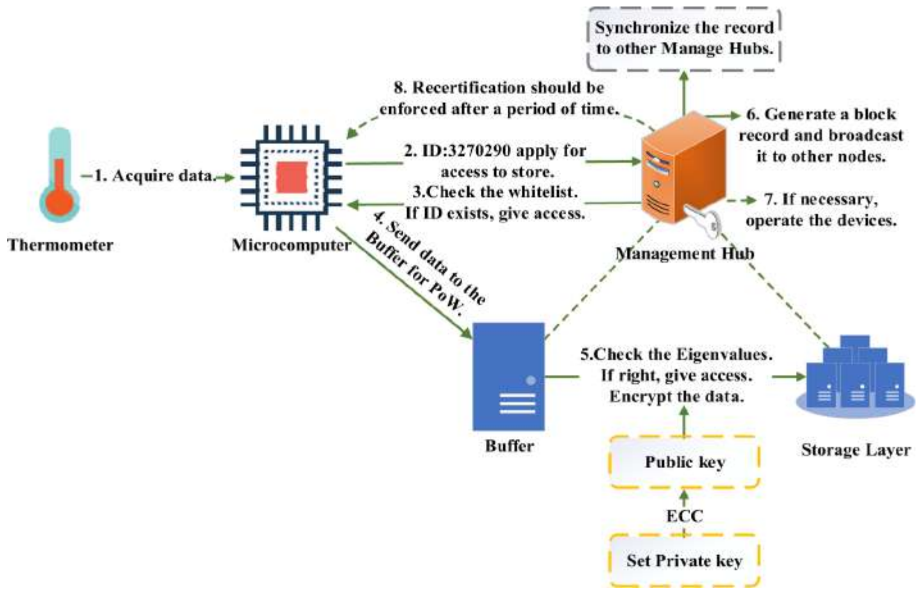

Synchronize the record to other Manage Hubs.
8. Recertification should be enforced after a period of time.
6. Generate a block record and broadcast it to other nodes.
1. Acquire data.
2. ID:3270290 apply for access to store.
3.Check the whitelist. If ID exists, give access.
7. If necessary, operate the devices.
Thermometer
Microcomputer
Management Hub
4. Send data to the Buffer for PoW.
5.Check the Eigenvalues. If right, give access. Encrypt the data.
Buffer
Storage Layer
Public key
ECC
Set Private key

Figure 3.12: Flowchart of the Sensor-Management Hub Data Acceptance Procedure. Sourced from: [28]

The same principle was applied to the user and management hub connection, figure 3.13.

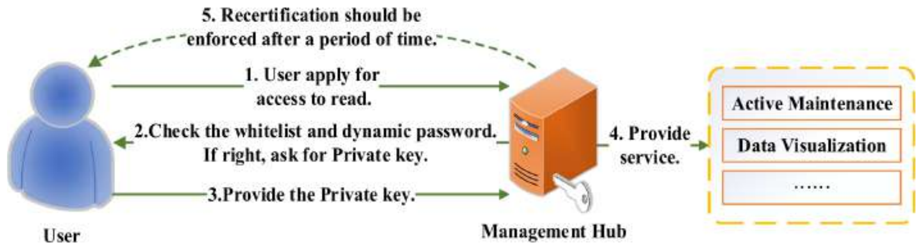

5. Recertification should be enforced after a period of time.
1. User apply for access to read.
2.Check the whitelist and dynamic password. If right, ask for Private key.
3.Provide the Private key.
4. Provide service.
Active Maintenance
Data Visualization
......
User
Management Hub

Figure 3.13: Flowchart of the User-Management Hub Data Access Procedure. Sourced from: [28]

The authors point out the lack of smart contract integration as one of the proposed work's flaws, being referenced as future work for the architecture's development. However, the usage of a PoW consensus mechanism within a permissioned blockchain seems to be an inefficient way to assure trust within an already somewhat trustworthy network. Other consensus mechanisms could have been used, with potentially less computational workload required.

3.7.4 Implementing a Blockchain From Scratch by Knirsch et al.

Authors of [29] set out to create a lightweight and highly customizable blockchain platform, designed to manage energy trading of photovoltaic power between customers. Although several of the already existing implementations of both private and permissioned blockchains were contemplated, ultimately it was decided to custom build a blockchain architecture programmed in a high-level language, Java. This choice is backed by the authors' need for a highly tailored solution to conform to local legal requirements. Figure 3.14 depicts the work's general setup.

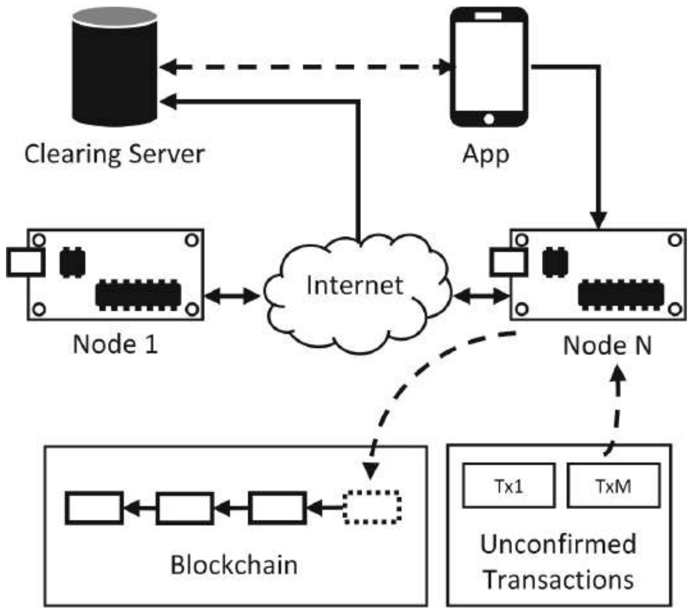


Figure 3.14: Overview of the Energy Management Setup. Sourced from: [29]

Through the deployment of Raspberry Pi Model 2 B development boards to act as *smart-meters* networked together by blockchain based technologies, it is possible to account for the energy transfers between residents of apartment buildings equipped with photovoltaic power generators. Alongside the Raspberry Pi nodes act two other components: the clearing server and the smartphone app. The clearing server exists as an off-chain database containing relevant information for each of the customer's utility providers' internal use. A smartphone app was added to provide an user-accessible interface for monitoring the network.

Communication links between the nodes (i.e., smart-meters) as well as between the node and the app are secured with TLS v1.2 using a hybrid encryption scheme based on Elliptic Curve Diffie-Hellman key exchange and AES-256 with CBC [29]. The link between the app and clearing server uses standard HTTPS with authentication and encryption. Due to the use of TLS-based secure sockets, the overhead created in the network was significant causing high delays when compared to unencrypted connections, even with reduced amounts of data. The time required to establish a connection both through a secured socket as well as through a regular, unprotected socket, was measured, with the results shown in figure 3.15.

The handshake, specifically the key exchange, requires about as much time as the entire unencrypted connection does in total. However, in an application such as a private blockchain network for enterprise use, priority must be placed on data integrity and security, with potential overheads caused by encryption techniques such as TLS having to be accepted as a necessary. The major drawback found in this solution was the possibility of stability issues due to delays induced by the TLS-handshakes, as stated by

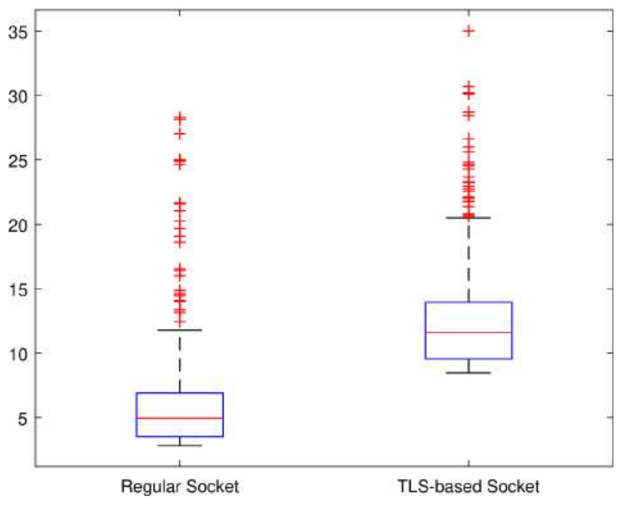


Figure 3.15: Time in ms for Establishing a Connection With Sockets. Sourced from: [29]

the authors. Furthermore, use of the Java 8 language seemed somewhat inappropriate for this application due to its limit on cryptographic key sizes - limited to 128 bits - requiring workarounds to achieve better levels of security.

3.7.5 Summary of Previous Scholar Work

Previously studied work all suffer from one (or multiple) of the following shortcomings:

- **Performance Insufficiencies** or **High Latency Times**: final application must ensure low latency times throughout the entire traceability chain due to the fast-paced production found at Renault CACIA's facilities coupled with the need for constant real-time data access. Data might be gathered at a rate as high as two measurements per second, with the final implementation having to cope flawlessly with the influx;

- Third-party **Open-Source Solutions**: open-source products have multiple advantages for both personal and corporate use, however an extremely reliable platform is needed, with very specific requirements (e.g., must be incorporated into existing infrastructure). Therefore, use of pre-existing and in-development platforms such as Hyperledger Fabric prove to be unstable and unfitted for deployment at Renault CACIA;

- Found solutions lack clear **Scalability** and **Flexibility** enabling mechanisms: none of the platforms let on exactly how it is possible to scale up or down the architectures. For instance, shown layered-based solutions ([1] and [28]) are conceptually scalable, being only required the duplication of current hardware and software, but not in a real-time fashion. An interesting proposition would be the ability to scale the solution during operation, even on the lower layer (i.e., sensors and IIoT), without the need to essentially recreate the traceability platform every time a change to the supply chain occurred.

3.8 Commercial and Open Source Solutions

3.8.1 Hyperledger Fabric

Hyperledger is an open source effort hosted by The Linux Foundation towards the development of cross-industry blockchain technologies, since 2016 [30]. Being a global collaboration, it has gathered support from several high-profile members such as IBM, Daimler, Intel, Systems, Applications and Products (SAP), Airbus, among many others.

Hyperledger seeks to provide enterprise grade distributed ledger frameworks and code bases to support business transactions, through a neutral, open and community driven infrastructure. Currently, a range of enterprise blockchain technologies are being developed and promoted including distributed ledger frameworks, graphical interfaces, client libraries and smart contract engines. Among these is Hyperledger Fabric [31], a foundation for developing distributed ledger applications or solutions.

Fabric's biggest selling point is its ability to provide modular and extensible architectures, whilst providing mechanisms for the deployment of smart contracts (called *chaincode* in Hyperledger Fabric). Unlike other blockchain systems, Fabric is private and permissioned, with members having to enrol through a trusted Membership Service Provider (MSP). Several options of data storage formats, consensus mechanisms and MSPs are provided. Another uncommon trait is the ability for *channel* creation, allowing different groups of participants to run separate ledgers of transactions. This feature is important for when not all transactions are meant to be disclosed through all of the network's participants. An overview of the Hyperledger Fabric system is shown in figure 3.16

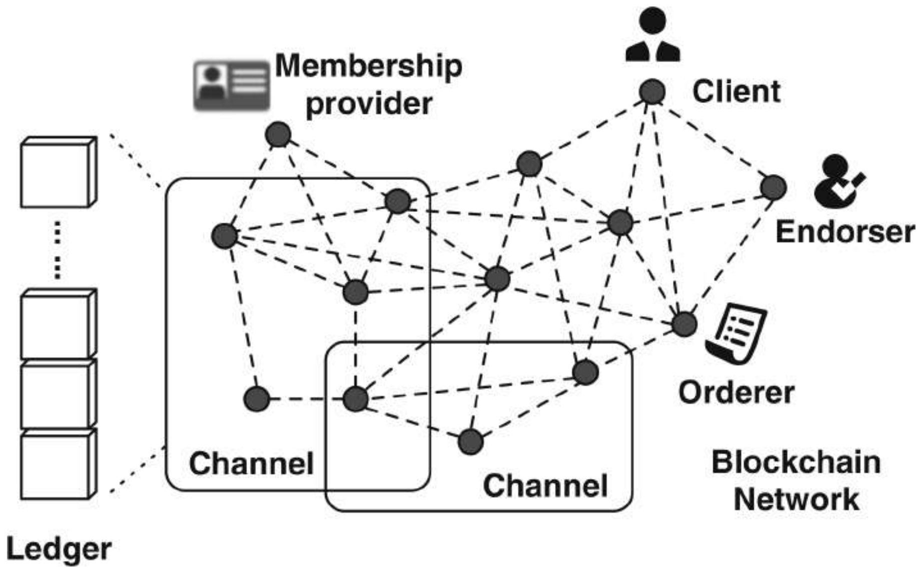


Figure 3.16: Overview of the Hyperledger Fabric System. Sourced from: [20]

3.8.2 Ethereum

Ethereum [32] follows Bitcoin's footsteps, improving on the notion of the capabilities of smart contracts when deployed within the blockchain. In Ethereum, smart contracts become a mean as to perform general computational tasks with the assurance that execution of code will be faithful. With a relatively short time interval between blocks (13 to 15 seconds on average) [20] Ethereum can assure high transactional throughput, however it is very prone to the creation of several concurrent blocks, figure 3.17. A modified Greedy Heaviest Observed Subtree (GHOST) protocol is implemented to address the concurrency issue, with the winning sub-chain of the blockchain not being the one with the most length, but the *heaviest* chain. *Weight* in Ethereum, refers to the cumulative difficulty of mining a chain.

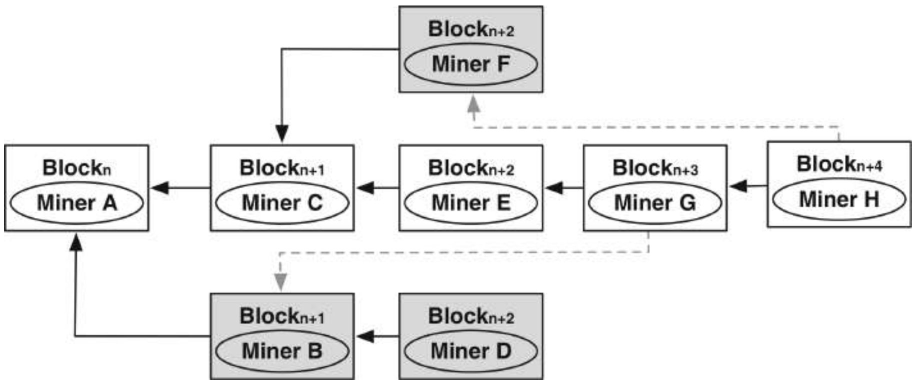


Figure 3.17: Ethereum Blockchain and Concurrent Blocks. Sourced from: [20]

Smart contracts in Ethereum are first programmed in a high-level language called Solidity [33] and then compiled into low-level bytecode, which is run by the Ethereum Virtual Machine (EVM). As running intricate programs adds significant overhead to the network, Ethereum uses the concept of *gas* to limit resource consumption. Deployment of smart contracts costs the user a specified fee of gas plus a varying amount based on the contract's data and computational requirements. This gas value can then be converted into Ether, Ethereum's version of a crypto-currency.

3.9 Enterprise and Industry Applications

Numerous successful applications of blockchain technology can already be found in key sectors including agriculture, manufacturing, retail, finance and governmental. For instance, supply chain management can deeply benefit from blockchain when tracking physical assets through changes in ownership or handling, with provenance certification assured. Prominent example of this is the AgriDigital initiative (Section 3.9.1).

3.9.1 AgriDigital

Founded in 2015, AgriDigital [34] uses blockchain technologies as a means to ensure digital trust within agriculture supply chains. The platform acts as the application layer through which farmers and traders can be connected to manage contracts, deliveries, inventory, invoices and payments [20]. Figure 3.18 shows a conceptual overview of AgriDigital's solution.

The main focus of AgriDigital is to bring together otherwise disparate information flows in a way that creates a single source of truth, providing supply chain assurance

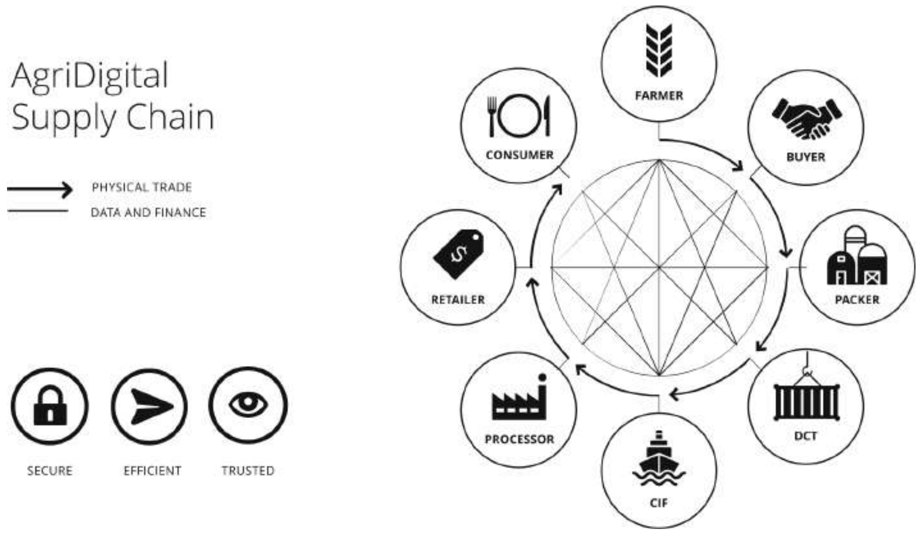


Figure 3.18: AgriDigital solution for data integration. Sourced from: [20]. ©2018 AgriDigital.

and transaction security whilst using that information to mitigate challenges such as the lack of transparency within the agricultural industry and minimize the propagation of counterfeit goods. Extensive use of smart contracts is enforced to provide proof of ownership and quality assurance.

Technical requirements faced by the AgriDigital architecture are applicable to any other blockchain-based supply chain [20]. For instance, the need for real-time transactions completed with low-latency, high availability of data and high scalability are imperative features in any industry-scale platform. Several pilot programs where already conducted, with each iteration showing advances with added security mechanisms and more complex smart contracts. Currently, AgriDigital is already present in the Australian, United States and Canadian markets.

Chapter 4

Proposed Blockchain Framework

4.1 Conceptual Architecture Layers

In light of what was previously found in works done namely by Hang et al. [1] and Wan et al. [28], the most efficient way to ensure a modular and scalable architecture is by stratifying into a multi-layer platform. As such, and as shown in figure 4.1, five layers are proposed, including: the IoT and Sensing Layer; the Gateway Layer; the Blockchain and Storage Layer; the Network and Security Layer; and lastly, the User Layer.

4.1.1 IoT and Sensing Layer

The **IoT and Sensing Layer** is the lowest level of the network and the one with highest user abstraction. It deals in data collection from the field, through the use of sensors and ID devices. Information regarding product, process, identification and localization are gathered allowing for full traceability. Each sensor needs to identify itself and attribute a timestamp to the collected data. Some of the sensors or IoT devices might possess intelligence, enabling them for edge computing tasks, performing additional actions and taking autonomous decisions at the edge level. On-edge analytical analysis is also possible.

As a measure to increase the overall security of the architecture, each device in the Sensing Layer is not connected to exterior networks, but instead operating in a local *intranet*. Each one of these intranets consists in one or more sensing units, forming a **Business Unit**. Connection with outer networks, and by extent, to the blockchain network is only allowed by first interfacing with the Edge Staging Gateways Layer.

4.1.2 Gateway Layer

The **Edge Staging Gateway Layer** bridges communication between the sensing devices and the decentralized ledger. Gateway devices are required a wide range of interfacing capabilities: accommodating several different sensors and IoT; and if required, interface existing brownfield equipment, effectively granting them with wireless capabilities bringing one step closer to a greenfield industrial landscape.

Accessing the Blockchain and Storage Layer will require connection to external networks, as the physical location of devices operating as network Nodes might vary greatly.

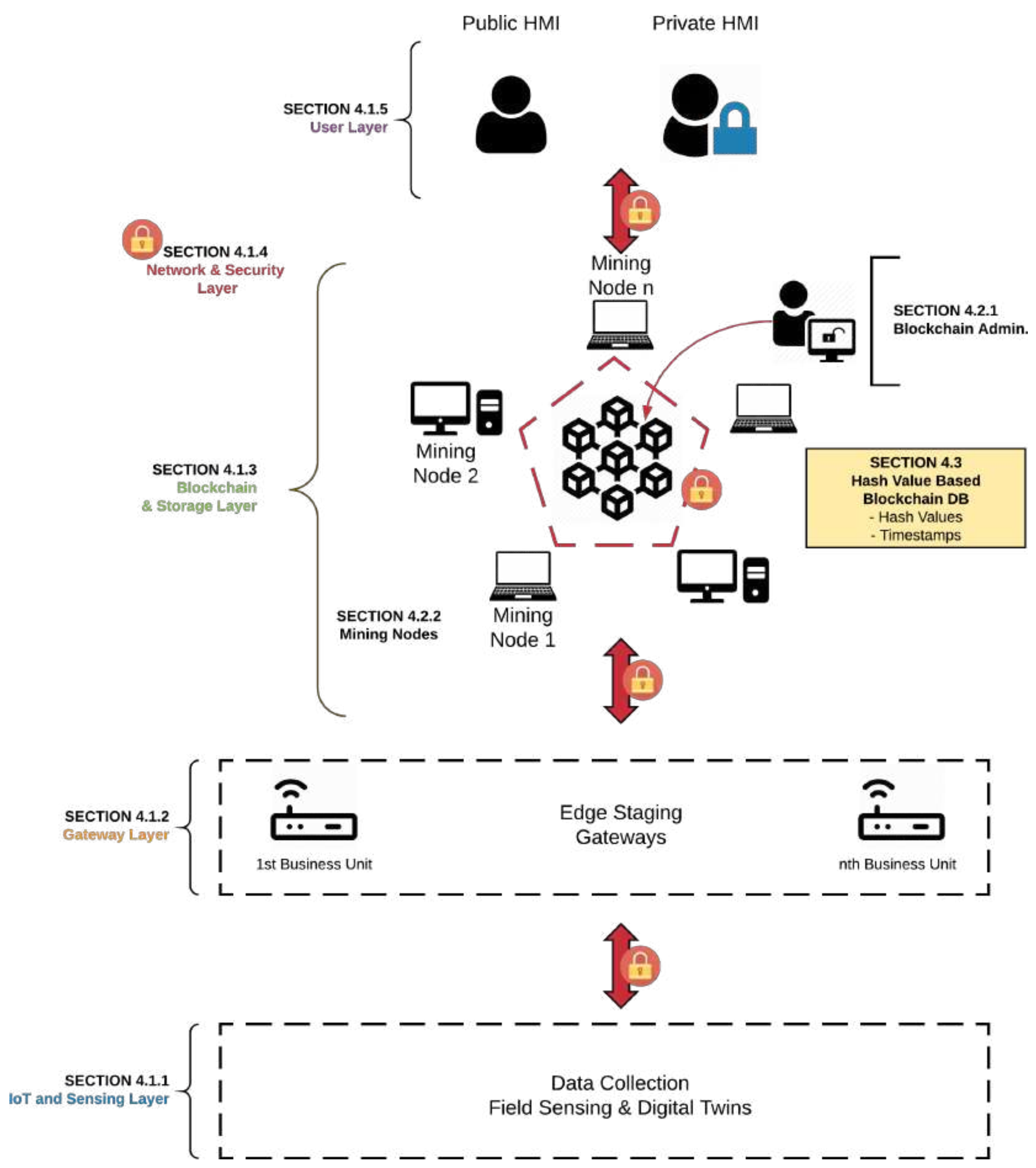


Figure 4.1: Architecture for the Proposed Solution

As external connections come with an inherently high risk, gateways will require deployment of several security features, such as the capability to establish secure connections (e.g., TLS and Hypertext Transfer Protocol Secured (HTTPS)) and some sort of firewall software. Appropriate hardware must be selected with sufficient compute power to allow a high inflow of input data coming from lower-level devices, whilst still maintaining secure connections with other higher-level network elements. As data comes from multiple sources, with different formats and at varying collection rates, the Edge Staging Gateways must temporarily store data and normalize it. Effectively, the Gateway might function as a buffer. As each node in the network is required access to the all information, Gateways have to transmit the collected, and normalized, data onto every node in the blockchain network.

4.1.3 Blockchain and Storage Layer

Two main functions are attributed to the **Blockchain and Storage Layer**: validation of data by the mining nodes and its storage. As the blockchain network is desirable to be as lightweight as possible, only the necessary information for certifying data on the enterprise system is retained on-chain. As such, the *simplified distributed ledger* will only contain the hash information characterizing each transaction. If the hash-value of a database entry does not correctly correlate with the hash-value stored in the blockchain network, across multiple entities (nodes), then it is possible to dismiss the database entry as incorrect or as being tampered.

It must be possible to deploy and remove nodes, conforming the network's architecture to the current enterprise needs. This requires the establishment of a synchronization mechanism across nodes, as the information kept on-chain is only relevant if it is possible to trace it back onto every node element in the network. Both the IoT and Sensing Layer and the Blockchain and Storage Layer must work side-by-side with the currently implemented traceability solutions, meaning that these levels will be interfaced with the enterprise systems, where all data is fully stored. Communication is maintained by implementing several Application Programming Interface (API).

4.1.4 Network and Security Layer

The **Network and Security Layer** involves all underlying technologies that concern secure operation of the network, for instance, implementation of cryptography tools and transaction encryption. All communications, with the exception of ones occurring in the IoT and Sensor Layer, are achieved using the TLS protocol with both server-side and client-side certification required, establishing HTTPS connections. A certification authority must be created, signing certificates that grant access to the network.

Other management roles credited to this layer include the establishment and enforcement of rules that dictate the blockchain's validation mechanisms. Lastly, information restriction to external users of the network - it is expected that in-house users of the network are granted with a higher level of information detail than third party entities.

4.1.5 User Layer

The upper-most layer, provides validated users access to the records gathered, allowing for real-time monitoring of the network, through use of APIs. Two types of user permissions are considered: *private* permissions and *public* permissions. Private permissions are only granted to in-house users, having a higher level of access to information stored on the network. Alternatively, public permissions can be granted to external agents, allowing them access only to the required data, blocking out additional features. This selection comes from enterprise decisions to assure process confidentiality. In-house or internal users might be internal plant departments, external plants or corporate users. External users may be suppliers, partners or even the final consumers.

4.2 Characterization of the Network Nodes

Network nodes can serve one of two functions: administrative responsibilities, section 4.2.1, or block mining, section 4.2.2. Nodes responsible for mining blocks can be deployed across multiple cells in a production line, different production lines, departments, factories or even organizations. As the number of mining nodes increases, so does the overall trust of the network, as more entities exist to validate and share the common ledger. On the other end, Administrator Nodes are reserved for the sovereign entity that fundamentally holds legislative power over the network, in this case Renault CACIA.

4.2.1 Blockchain Administrator

The **Blockchain Administrator Node** serves as a mediator between all the devices acting on the blockchain network, whether they are mining nodes, gateways or even front-end users. Main roles performed by the Blockchain Administrator include: managing message flow between the several layers and the nodes; and validate front-end users' access to the network and regulate the information to which they have access to.

Communication and access logs are also created and curated by the Administrator Node, allowing for future fail analysis to be conducted, and the creation of user log-in analytics. Lastly, the Administrator Node allows access to the system administrator Human Machine Interface (HMI), where it is possible to conduct such tasks as enrolling a new Mining Node or Gateway onto the network - admittedly, of course, that the to-be-enrolled entities have provided validated certificates. This, however, does not mean that the Blockchain Administrator has influenced over the data that circulates within the network. Administrators only have access to data blocks on a viewing privilege basis.

4.2.2 Mining Nodes

Mining Nodes are responsible for creating new data blocks, according to the governing rules deployed on the network. To increase the solutions modularity, they must be easily deployed, with minimal configuration, in a plug-and-play fashion. The network must be self-sustaining, requiring little in the way of maintenance. As such, Mining Nodes are required to solve any conflicts that may arise in terms of block acceptance as well as network synchronization between nodes, autonomously.

Each node will keep two copies of a *simplified shared ledger*: one in RAM for fast data access and quick response to enquiries, done for instance by a user or Administrator Node; and another kept in a auxiliary database, as a backup. This allows for both fast data transfers and calculations, whilst assuring data integrity in case of system failure - such as power loses. This simplified shared ledger is comprised only of the block headers, as explained in further detail in section 4.4.

Blocks are approved according to a variation of the Proof of Authority consensus algorithm. Any given miner cannot mine two consecutive blocks, with the miner that has not mined a block the longest, first in line to forge a new block. The proposed consensus mechanism is shown in figure 4.2.

Mining Nodes must agree upon an order for which to mine the blocks. If the order is not maintained by a given node, the forged block will not be accepted by the network. This mechanism allows for high data throughput, taking advantage of the high-level of established trust that inherently can be found in a permissioned private network.

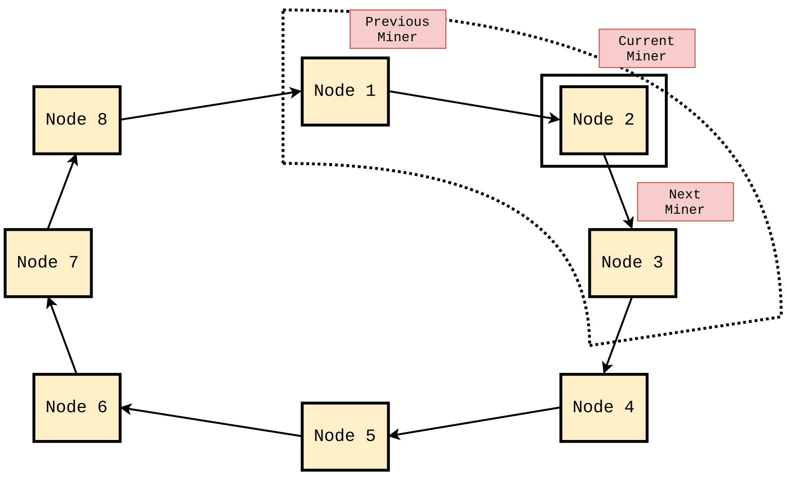


Figure 4.2: Diagram of the Mining Order to Achieve Consensus.

4.3 Databases

One of the architecture's main requirements is flexibility, and as such a malleable approach of handling data is fundamental. With that in mind, a non-relational, or also commonly referred to as non-SQL or NOSQL, database is used. This brand of databases does away with the tabular relations used in relational databases in a favour of other data modelling means.

Three different types of collections, comparable to tables in a relational database, can be identified as in use in the designed framework, figure 4.3. These include: the main collection of data making up the entirety of the blockchain; the nodes' simplified distributed ledger backup; and a user information collection, storing users' logs onto the network and usage statistics.

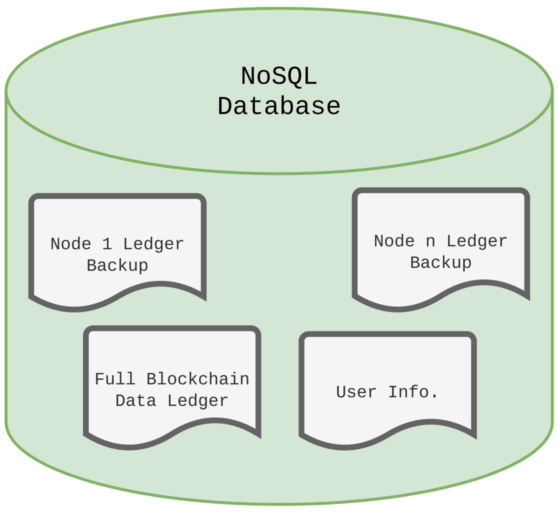


Figure 4.3: Diagram of the Collections in the NoSQL Database.

Although some drawbacks are associated to non relational databases, such as the cumbersome querying mechanisms resulting from the lack of tabular relations, and the potentially higher usage of disk space when compared to a Structured Query Language (SQL) database, this does make the architecture more easily deployable in a large variety of scenarios without any reconfiguration whatsoever.

4.4 Block Topology

Each block is constituted by two elements, the **block header** and the **block transactions**, as seen in figure 4.4.

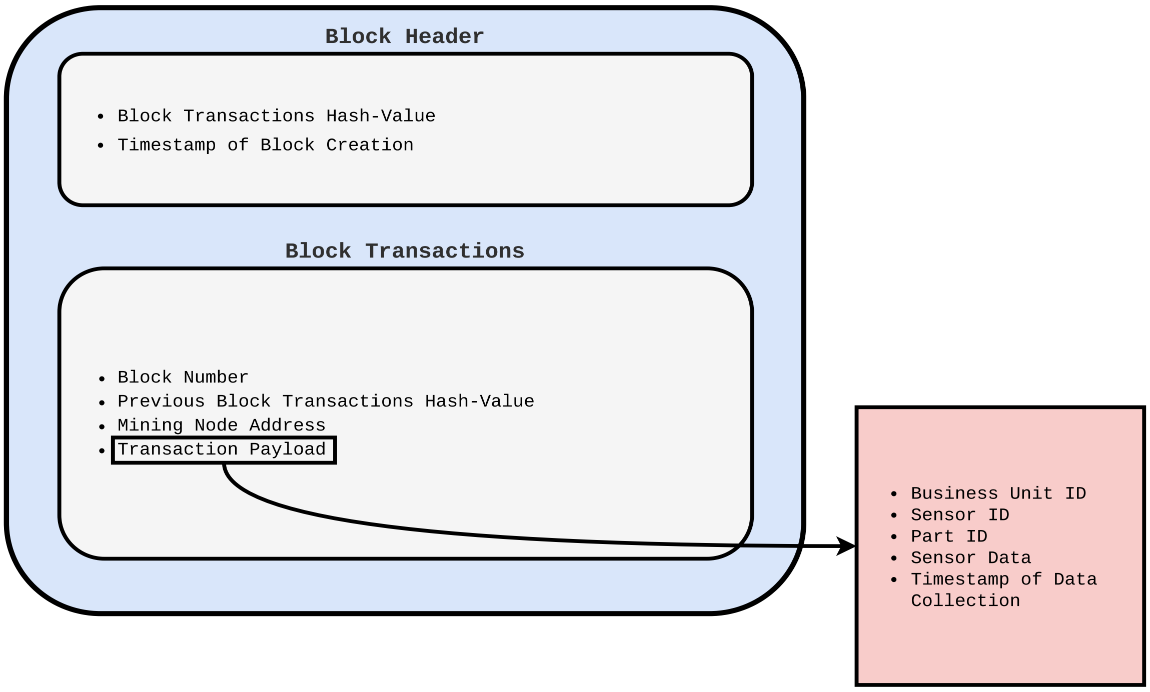


Figure 4.4: Topology of a Block of the Framework.

The block header contains the calculated hash-values of the block transactions portion of the block, as well as a timestamp of when the block was forged. The block headers are used to create the simplified distributed ledger in the nodes. As for the block transactions, it encompasses several important metrics to characterize the data, such as the current block number - an incremental counter, to account for the total sum of blocks; the previous block's transactions hash-value to keep the linkage in the chain; the address of the node set to mine the block and the transaction payload. The transaction payload is the data wished to store. As a non relational database is being used, there is a certain freedom as to which format the stored data follows, however, for the use case of the Variable Displacement Oil Pump Line, it is of reasonable assumption that the metrics shown in figure 4.4 are of crucial importance.

4.5 Integration with Enterprise Systems

The proposed framework works alongside existing enterprise traceability technologies, as illustrated in figure 4.5. For instance, data collected from the IoT and Sensing Layer and later processed at the Gateway Layer will still have to be stored in the existing corporate databases. Additionally, information in the traditional systems will have to cross-checked with the records found in the distributed ledger in the blockchain.

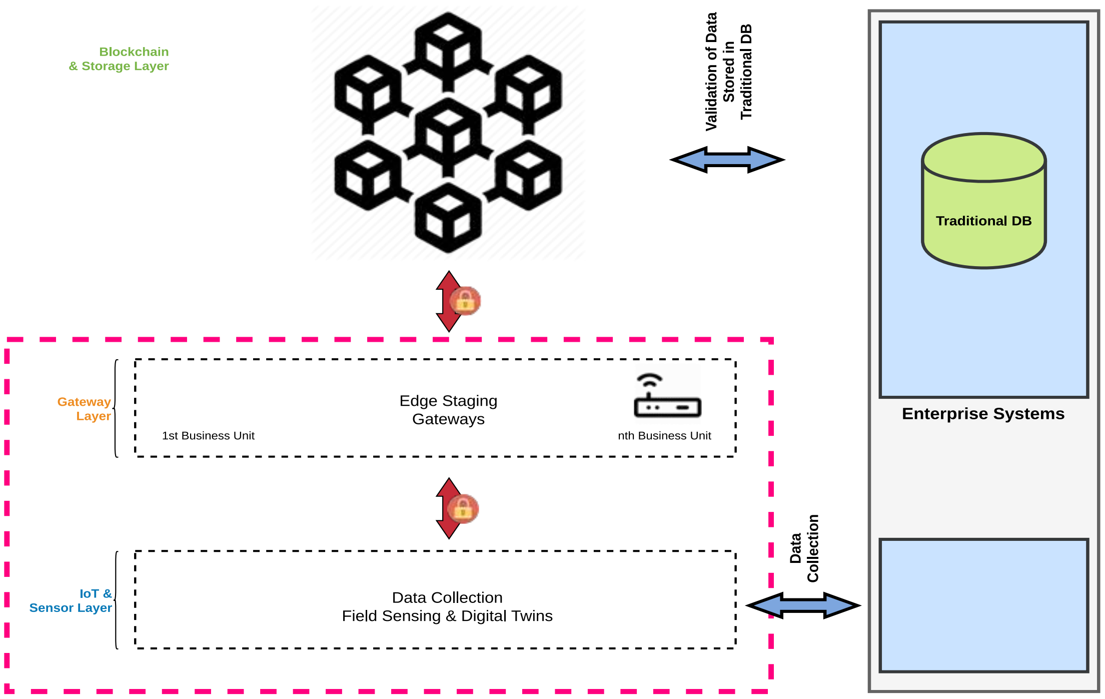


Figure 4.5: Framework Integration with Current Enterprise Systems.

In the variable displacement oil pump production line, each part is initially marked with a barcode tag, used to identify the components during production processes. In the proposed framework, each one of these tags will be associated with a set of hash-values characterizing the stored data-blocks. The manner in which data is saved, must remain similar between existing traceability systems and the proposed blockchain-based system, ensuring that hash-values can be easily determined and cross-checked between the two systems.

Intentionally blank page.

Chapter 5

Blockchain Framework Implementation

Several stages of development are required to implement the complex framework proposed at Chapter 4. In the first stage it is required to establish a working *back-end* of the planned architecture with all the necessary security features to assure data protection, integrity and immutability against malicious foreign agents whilst operating at high speed and with great data throughput. The second stage involves the establishment of all the *front-end* technologies that make possible the monitoring and access of data to authorized users. All of the segments that make up the planned technological structure are shown in figure 5.1.

5.1 IoT and Sensing Layer

The IoT and Sensing Layer is above anything, dependent on the existing equipment on the production floor. As testing at the Renault CACIA facility is not practical, three different IoT devices are used to simulate multiple interfacing possibilities with the Gateway Layer. These include an Arduino Uno R3, an ESP8266 and an ESP32 development boards. The main objective behind the simulation assembly is to show that connecting with the Gateway Layer is not dependent on one type of communication interface or low-level hardware.

Production flow is replicated according to what is depicted in figure 5.2. The arriving part is immediately identified by the first sensing unit - S001 - through an identifying Radio-Frequency Identification (RFID) tag. At this station, environmental variables are also monitored. In the real life production line at Renault CACIA, this up front part identification is conducted through a barcode tag reading. As more closely resembling hardware was not available, a RFID based system was chosen instead. Following the initial part identification, two further manufacturing processes are simulated - S002 and S003 - with each being activated by the press of a push button, representing the inductive proximity sensors that acknowledge a part's arrival at a station. Comprised by three IoT devices - S001, S002 and S003 - along with the remaining of their respective hardware, and an Edge Staging Gateway, the mock-up of a Business Unit, RBU01, is complete.

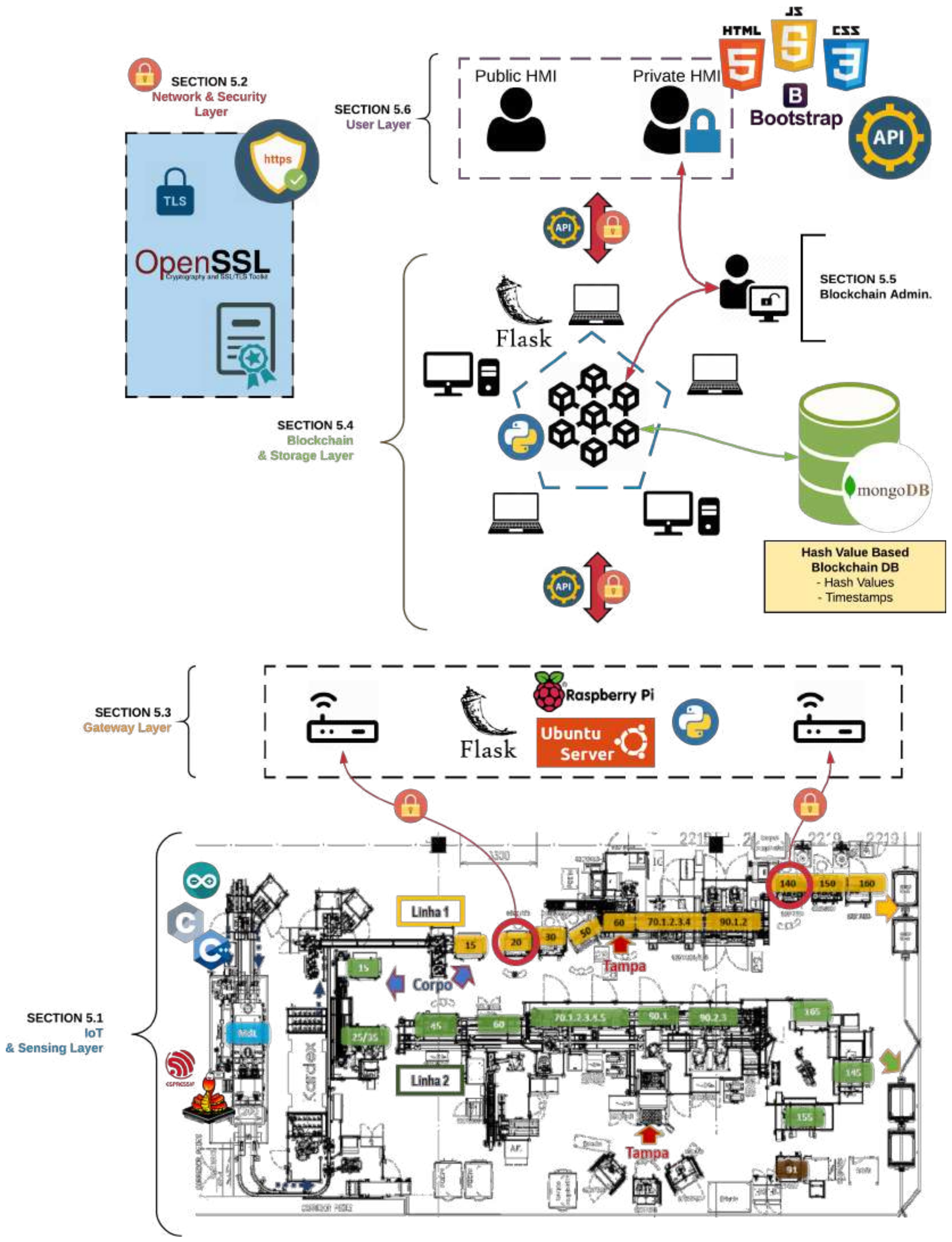


Figure 5.1: Implementation Architecture.

5.1.1 Hardware and Assembly

Sensing Unit S001

The first of the sensing units, S001, is built upon an Arduino Uno Revision-3 development board, which in itself is based on the Atmel ATmega328P microcontroller[1]. Figure 5.3

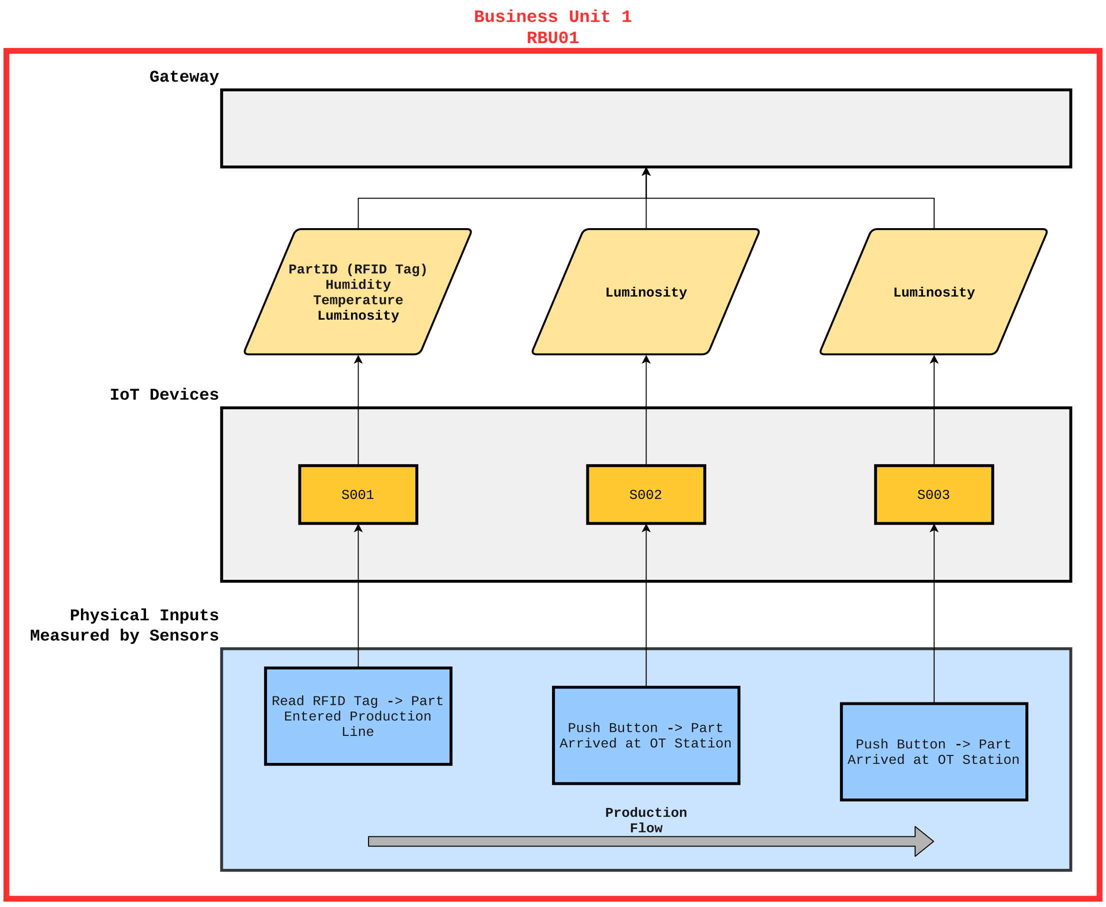


Figure 5.2: Diagram of Designed Production Flow Simulation.

depicts the IoT device, and figure 5.4 its respective pinout diagram.

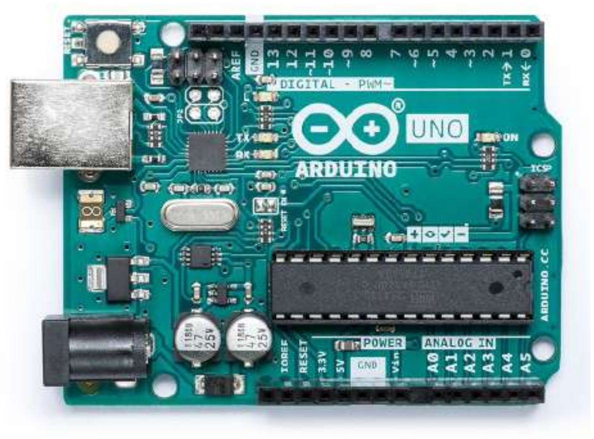


Figure 5.3: Arduino Uno R3.
Source[1].

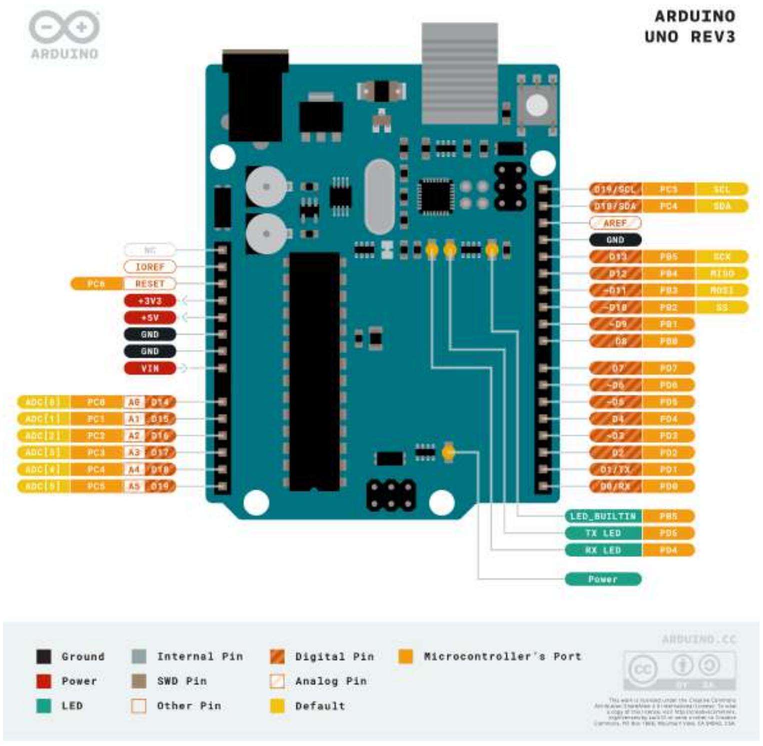


Figure 5.4: Arduino Uno R3 Pinout Diagram.
Source[1].

Basic characteristics of the Arduino Uno development board include:

- 14 digital input/output pins (of which 6 are Pulse-Width Modulation (PWM) capable);
- 6 analogue input pins;
- 16 MHz ceramic resonator (CSTCE16M0V53-R0);
- 32 KB of Flash memory;
- 1 KB of Electrically Erasable Programmable Read-Only Memory (EEPROM);
- 2 KB of Static Random Access Memory (SRAM);
- 5V operating voltage.

A MFRC522 module, figure 5.5, is used to enhance the Arduino board with RFID read/write capabilities. This module allows for contactless communication at 13.56 MHz, and is ISO/IEC 14443 A/MIFARE and NTAG compliant [35]. A wide range of host interfaces, including Serial Universal Asynchronous Receiver/Transmitter (UART), I^{2}C bus interface or SPI are available, with the latter being used for communication in the developed solution, allowing for up to 10 Mbit/s.

[1] Arduino Foundation - *Arduino Uno REV3.* Accessed 30 April, 2020. URL: `https://store.arduino.cc/arduino-uno-rev3`

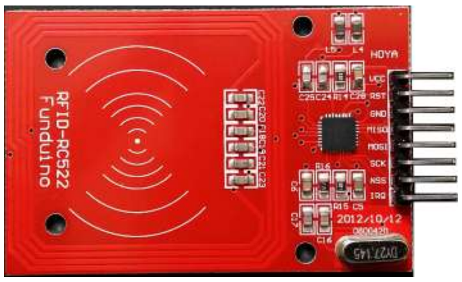


Figure 5.5: Funduino MFRC522 RFID Module.

Measurement of temperature and humidity is done through a DHT11 module, figure 5.6. These sensors are extremely basic, containing a capacitive humidity sensor and a thermistor, along with a microchip capable of performing analogue to digital conversions.

Figure 5.6: DHT11 Temperature and Humidity Module.

Lastly, luminosity is gathered by a Light Dependent Resistor (LDR). The complete assembly schematic for the S001 sensing unit is shown in figure 5.7. Communication between the Edge Staging Gateway and the Arduino is established through a Serial RS232 connection, using the Arduino's built-in full-size USB Type-B port.

Sensing Unit S002

Sensing unit S002, represents the second work post for which the part is subject to. Based on an ESP32 TTGO T7 v1.3, figure 5.8, this development board has significantly more processing capabilities compared to the Arduino, as a result of its much more powerful Xtensa dual-core microprocessor. Specifications for each ESP32 board may vary according to each manufacturer, but for the used model it is found [36]:

- Xtensa® dual-core 32-bit LX6 microprocessors;
- 448 KB Read-Only Memory (ROM);
- 520 KB SRAM;
- 16 KB SRAM in Real Time Clock (RTC);
- Internal 8 MHz oscillator;
- External 32 MHz crystal oscillator for RTC;

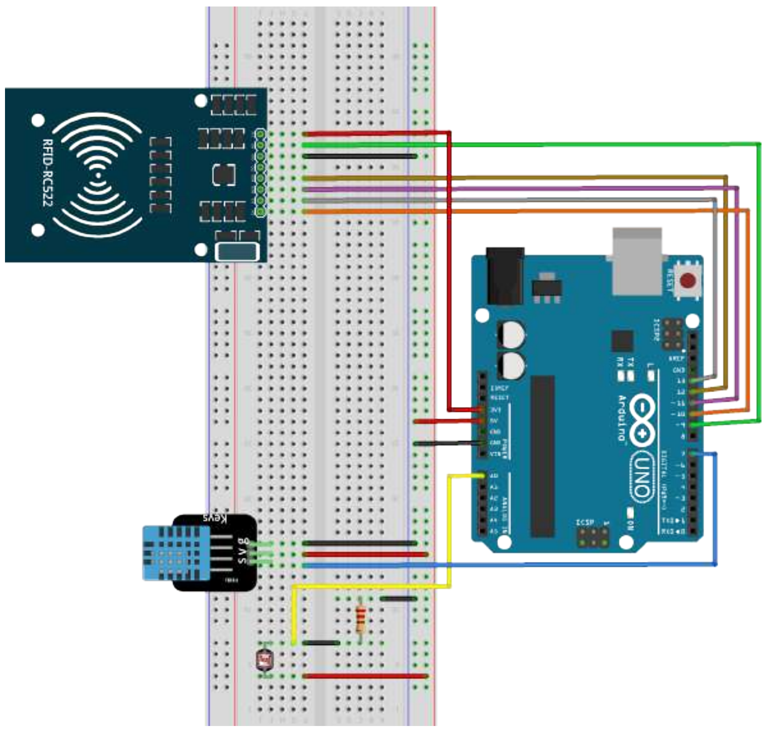


Figure 5.7: Assembly Schematic for S001.
Designed in Fritzing[2].

- 34 programmable General Purpose Input/Output (GPIO) Pins including SPI, I^{2}C, UART, TX/RX and PWM capabilities.

Furthermore, this module is capable of establishing WiFi connections (2.4 GHz up to 150 Mbps) and has built-in cryptographic hardware acceleration, namely for hashing functions (SHA-2 standard) and AES. Taking advantage of these capabilities, communication with the upper Gateway Layer is done over an encrypted wireless connection.

Assembly of the second simulated work post is done as shown in figure 5.9. A push button is used to mimic the arrival of a new part into the work post, after which the luminosity value is gathered and sent to the upper layers. A LED is used in the assembly as a status indicator, used for debugging.

Sensing Unit S003

The final sensing unit, S003, has the same basic working principle as sensing unit S002, with the only difference being it based on a ESP8266 ESP-12F NodeMCU, figure 5.10, rather than on a ESP32. Specifications and features stay largely unchanged from the ESP32, with the notable exception being the use of only a single-core 32-bit microprocessor. For the light workloads being demanded from these modules, this loss of computing power does not cause major limitations to the network.

Tasks performed by sensing unit S003 are cloned from S002, however the selected ESP8266 development board already features an integrated photo-resistor as well as a

[2]Fritzing home-page. Accessed 30 April, 2020. URL: `https://fritzing.org/home/`

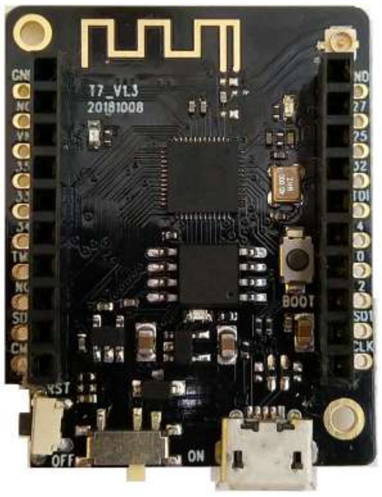


Figure 5.8: ESP 32 TTGO T7 v1.3 Development Board.

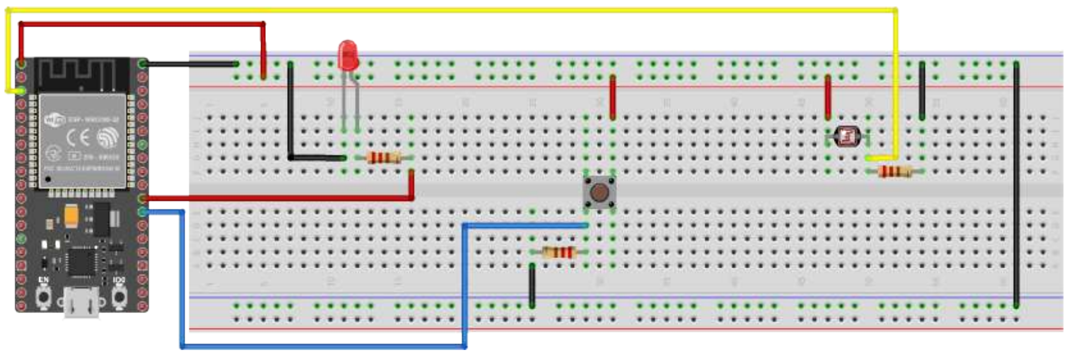


Figure 5.9: Assembly Schematic for S002.
Designed in Fritzing[2].

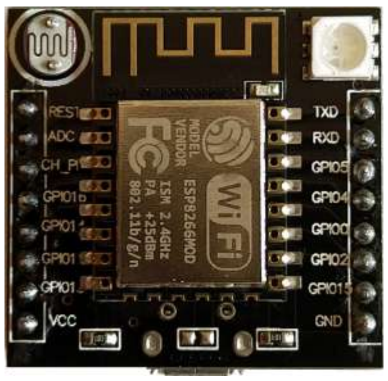


Figure 5.10: ESP8266 ESP-12F NodeMCU.

RGB LED, figure 5.11,therefore no additional hardware assembly was required. Much like what is seen in S002, this sensing unit also communicates with the upper Gateway Layer over an encrypted wireless connection.

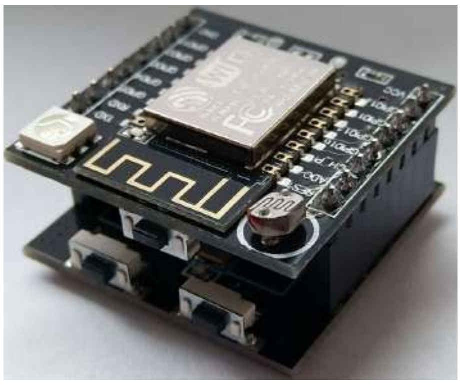


Figure 5.11: ESP8266 ESP-12F NodeMCU - Front 3/4 View.

5.1.2 Software and Programming

Much like how a wide range of communication protocols might be used for the IoT and sensing devices to interface the Edge Staging Gateways, several programming languages might also be used to achieve this purpose. As an example, programming of the Arduino and the ESP devices followed two different routes: the former being programmed in a more traditional language for microcontrollers, C/C++, and the latter being programmed in a variation of Python, applied to microcontrollers, MicroPython. Further insights on MicroPython are given in Appendix A. A flowchart describing the general operation of the program created for the Arduino, simulating sensing unit S001, can be found in figure 5.12.

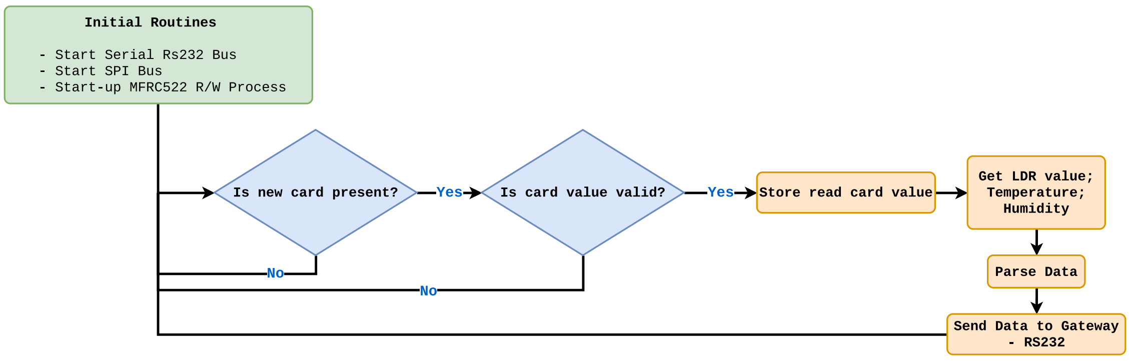


Figure 5.12: Flowchart of the Arduino Program.

The output string being sent over from sensing unit S001 follows a simple pattern that can be easily parsed at the Gateway, as characterized in figure 5.13.

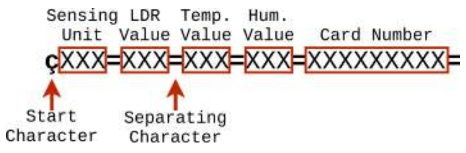


Figure 5.13: Diagram of the String Output at S001.

As for sensing units S002 and S003, programming is more complex to ensure the securing of established wireless connections with the Gateway Layer. Leveraging the ESP's cryptographic hardware acceleration, message payloads are encrypted using AES CBC. Flowchart 5.14 shows operation of the program running in the ESP devices.

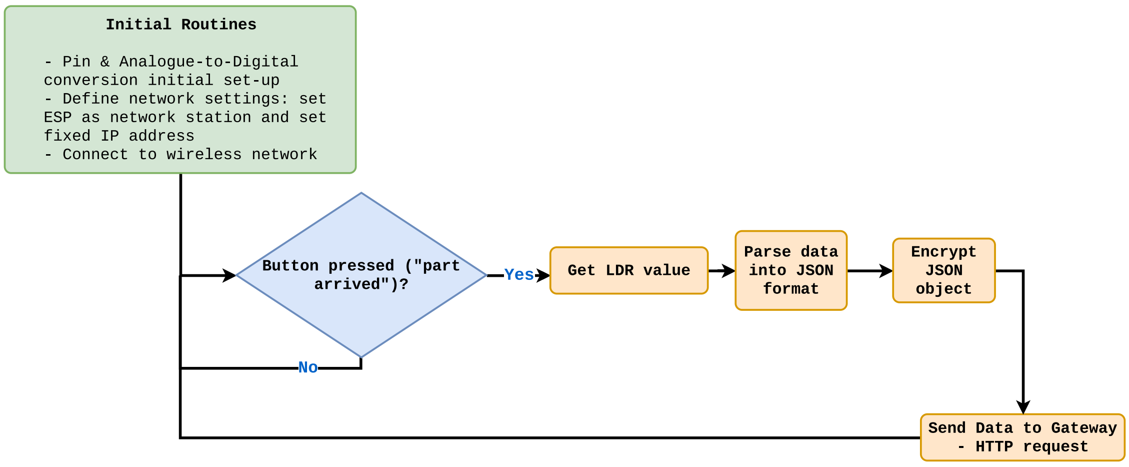

Initial Routines
- Pin & Analogue-to-Digital conversion initial set-up
- Define network settings: set ESP as network station and set fixed IP address
- Connect to wireless network
Button pressed ("part arrived")?
Yes
No
Get LDR value
Parse data into JSON format
Encrypt JSON object
Send Data to Gateway - HTTP request

Figure 5.14: Flowchart of the ESP Devices Program.

Encryption is achieved by first creating a unique identifier for each machine. This was generated using Python's **uuid** module, which allows the creation of Universally Unique Identifier (UUID) objects according to the RFC 4122 standard [3]. The UUID is then hardcoded into the IoT and sensing devices with network capabilities. Gateways also have knowledge of these identifiers through an internally stored list that contains all UUIDs from the authorized IoT and sensing devices operating in the network. During communication between equipments, the key used for data encryption is the hash-value calculated for the UUID using the SHA256 hash-function. Diagram 5.15 exemplifies the encryption process.

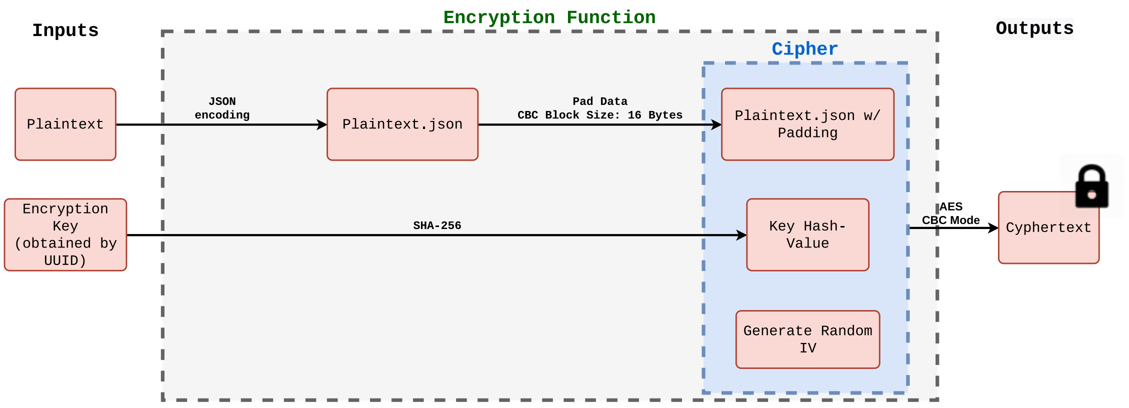


Figure 5.15: Message Encryption Process in IoT Devices.

As symmetric-key encryption is used, the process becomes very efficient, even when

[3]Python 3 Documentation - *uuid — UUID Objects According to RFC 4122*. Accessed 4 May, 2020. URL: `https://docs.python.org/3/library/uuid.html`

used in relatively low-power IoT devices, granting a high data throughput. Collected and sent data does not have to follow a specific format, requiring only to be encoded in a JSON format. Listing 5.1 shows an example of the passed data-structure from the IoT devices to the Gateway Layer.

```
# Data structure to send to Gateway Node.
data = {'sensing_unit': 'S002', # Sensing Unit No.
'LDR': analog,                  # LDR read value
'LED': LED.value()}             # Value of debug LED
```

Listing 5.1: Example of IoT Device Data-Structure.

This data-structure might not be a multiple of 16 bytes, as explained in 3.4.2, and possible need of padding is contemplated in the devised solution. Furthermore, the aforementioned section also mentions the need to produce an IV when using AES in CBC mode. This value is randomly generated using the device's built-in random number generator.

5.2 Network and Security Layer

Securing a network is a complex task that requires an in-depth analysis of all possible points of failure. The most blatant security feature found in the implemented framework is in form of encrypted connections, of which both symmetric-key and asymmetric-key encryption can be found. The creation of a Certification Authority is vital to the use of asymmetric key encryption. Moreover, besides encrypted communication channels, other more traditional means of cyber-security are applied, such as the need for passwords to access the frontend HMI. Lastly, network security depends not only on maintaining the network's integrity against malicious attacks, but also preventing data loss or mutation in case of hardware or software related failures.

5.2.1 Certification Authority

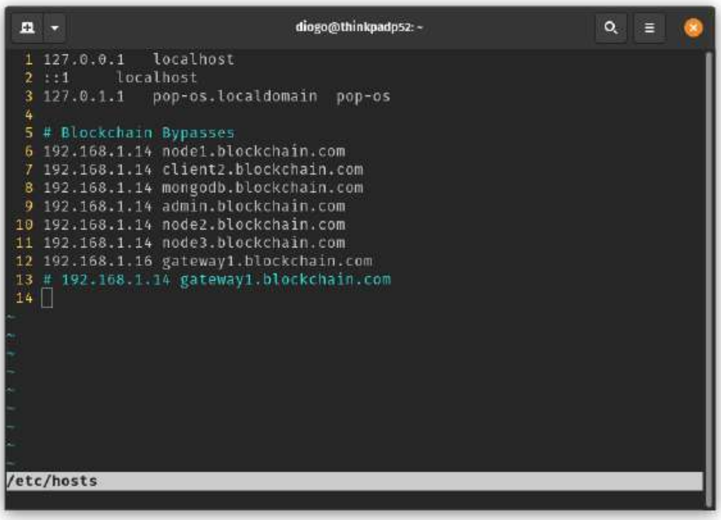
In the proposed framework, all elements operating above the IoT and Sensing Layer - who only operate in a restricted intranet - maintain connections to outside networks, foreseeing Mining Nodes in multiple remote locations, such as different departments, production-plants or even in customers' or suppliers' facilities. This calls for additional security measures, one of which comes in form of a CA, responsible for validating network participants. Services provided by a CA consist in the binding of an element's public key to an entity, certifying it [13]. Commercial CA utilities exist, however, in order to expedite the development process, a CA was implemented from the ground-up. These self generated CA agents create what is often referred to as self-signed certificates. To create a CA, OpenSSL[4](v1.1.1f) was used, which is a commercial-grade toolkit for TLS/SSL, as well as a general purpose cryptography library. Producing a local Certificate Authority can be done with the following command using a UNIX Bash Shell, listing 5.2.

```
openssl req -newkey rsa:2048 -new -x509 -sha256 -extensions v3_ca -out
   ca.cert -keyout ca.key -subj "/C=PT/ST=Aveiro/L=OAZ/O=UA-DEM/CN=
   blockchain.com" -nodes
```

Listing 5.2: Creating a Local Certificate Authority.

Used **req** command produces and processes certificate requests. With **-newkey**, both a new certificate and private key are created, in this case with a RSA key of 2048 bits. To guarantee that a self signed root CA is outputted, **-x509** command is used. Setting a subject name is done through **-subj**, with the following argument string being used to characterize the certificate recipient. For instance, in listing 5.2, the country (C), state/province (ST), locality (L), organisation (O) and common name (CN) were provided. Common names represent the name of the server verified by the certificate - e.g., *www.example.com*.

Several servers are used in the framework, and each requires a different certificate. However, during simulation of network operation, all servers are located within the same remote machine. Therefore, common names were artificially created, through aliases located in the **/etc/hosts** file. This file is used by the machine's Operating System (OS) when retrieving IP addresses: the system will first look in **/etc/hosts** for a matching hostname; if not defined, then the Domain Name System (DNS) server will be used. Figure 5.16 shows the implemented **/etc/hosts** file for simulation of the blockchain network. Although added domain names mostly all point to the same IP address, they fundamentally act as representing different entities.



```
127.0.0.1   localhost
::1     localhost
127.0.1.1   pop-os.localdomain  pop-os

# Blockchain Bypasses
192.168.1.14 node1.blockchain.com
192.168.1.14 client2.blockchain.com
192.168.1.14 mongodb.blockchain.com
192.168.1.14 admin.blockchain.com
192.168.1.14 node2.blockchain.com
192.168.1.14 node3.blockchain.com
192.168.1.16 gateway1.blockchain.com
# 192.168.1.14 gateway1.blockchain.com

/etc/hosts
```

Figure 5.16: Hosts File Created for Implemented Blockchain Network.

By applying the last command, **-nodes**, key encryption is suppressed. Note that key encryption in this context means that the key *file* - *.key* - is encrypted and that a user would be prompted to insert a password in order to read or modify it. The process of signing a certificate and consequently granting an entity access to the network can now take place. The following listing, 5.3, is a certificate creation and CA signage example, for a generic certificate/private-key pair, **client1.csr** and **client1.key**, with the common name *client1.blockchain.com*.

```
# Creating a certificate for client
openssl req -newkey rsa:2048 -new -sha256 -out client1.csr -keyout
    client1.key -subj "/C=PT/ST=Aveiro/L=OAZ/O=UA-DEM/OU=user/CN=client1.
```

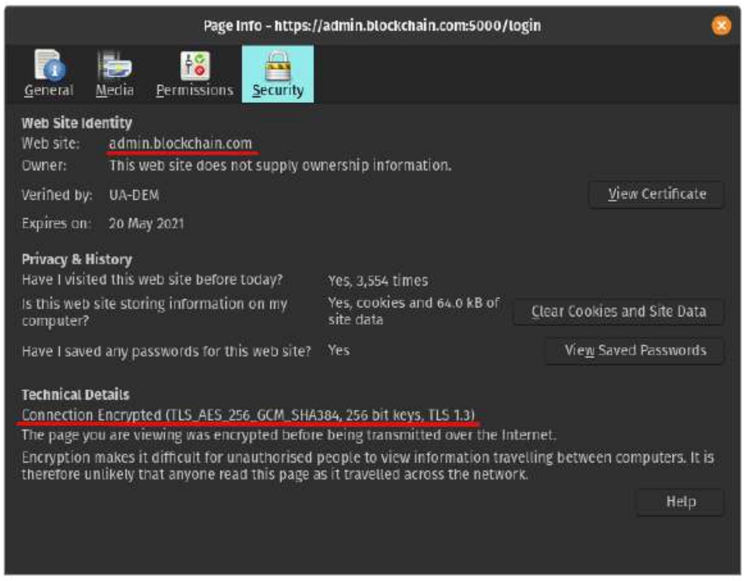
[4]OpenSSL Homepage. Accessed 5 May, 2020. URL: `https://www.openssl.org/`

```
    blockchain.com" -nodes

# Signing certificate by root CA
openssl ca -in client1.csr -out client1.cert -keyfile ca.key -cert ca.
    cert -outdir .

# Concatenate key-file and certificate-file into one .pem file
cat client1.cert client1.key > client1.pem
```

Listing 5.3: Creating and Signing a Client Certificate.

5.2.2 Encrypted Connections

Both symmetric and asymmetric encryption is used. Devices with higher computational constraints and that communicate over wireless connections, such as IoT, rely on symmetric encryption schemes, in this case AES in CBC mode. Machines with greater hardware specifications and that require connections over external, unsafe, networks, use the comparatively more resource demanding RSA public-key cryptosystem. Private and public keys are generated and signed in according to what is shown in section 5.2.1. When accessing the blockchain frontend interface through a common web-browser, it is possible to infer that the connection is indeed encrypted. For instance, when accessing the HMI via Mozilla Firefox 76.0 web browser, the following information is found - figure 5.17.


Page Info - https://admin.blockchain.com:5000/login

Web Site Identity
Web site: admin.blockchain.com
Owner: This web site does not supply ownership information.
Verified by: UA-DEM
Expires on: 20 May 2021

Privacy & History
Have I visited this web site before today? Yes, 3,554 times
Is this web site storing information on my computer? Yes, cookies and 64.0 kB of site data
Have I saved any passwords for this web site? Yes

Technical Details
Connection Encrypted (TLS_AES_256_GCM_SHA384, 256 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorised people to view information travelling between computers. It is therefore unlikely that anyone read this page as it travelled across the network.

Figure 5.17: Security Features of Implemented Frontend Interface.

The most current version of TLS is being used, making it mandatory for all elements operating in the network to be TLS 1.3 compliant. As seen in the previous figure, TLS_AES_256_GCM_SHA384 ciphersuite is being used. This is an Elliptic Curve Diffie Hellman Ephemeral (ECDHE) based ciphersuite[5]that provides forward secrecy as sessions keys are generated for each new session, as well as faster performance, being based on Elliptic-Curve cryptography.

5.2.3 Identity Authentication

Other conventional methods of security and authentication are used. For example, when trying to access the frontend interface, users are greeted with a login-screen, figure 5.18.

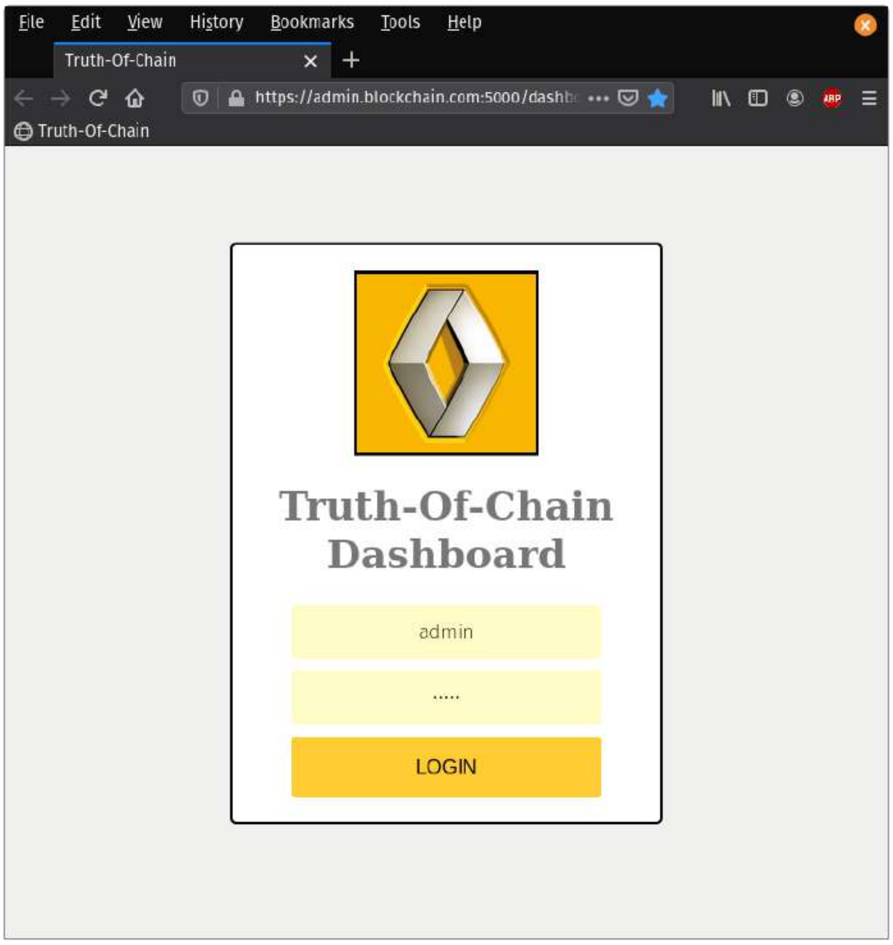

File Edit View History Bookmarks Tools Help
Truth-Of-Chain
https://admin.blockchain.com:5000/dashb
Truth-Of-Chain
Truth-Of-Chain Dashboard
admin
.....
LOGIN

Figure 5.18: Frontend Interface Login Screen.

According to user permissions, different interfaces are shown. Only system administrators have access to network settings and configuration pages. User information, such as passwords or usernames are indirectly kept in a database. In reality, only the hash-values of the passwords and usernames are kept, assuring private data confidentiality. Furthermore, login attempts are limited to once-per-second-per-user. This was implemented in order to hinder possible brute-force attacks.

[5]OpenSSL Wiki (OpenSSL Documentation). Accessed 7 May, 2020. URL: `https://wiki.openssl.org/index.php/TLS1.3`

5.2.4 Preserving Network Integrity

Hardware and software failures were also contemplated, with existing safeguards acting against power-failures or sudden stops in one of the network's Nodes. For once, backups of simplified distributed ledgers exist, one for each node. Furthermore, in the eventuality that a network node is decommissioned, it can later be reaccepted without any setbacks, as re-syncing with the network is done automatically. Lastly, all equipment that communicates through HTTPS requests, is subject to response wait period timeouts, avoiding indefinite wait loops. Devices will stop expecting a response from the message recipient once a timeout period expires. This also prevents Denial of Service attacks and avoids stopping the entire network if suddenly one device stops responding.

5.3 Gateway Layer

To the Gateway Layer it is attributed two main functions: retrieve values from the IoT and Sensing Layer; parse collected data into a normalized data-structure and pass it on to the upper layers. Retrieval of data can be either through a wired RS232 communication channel or over a wireless connection. Contemplating these functions, three main processes occur at the gateway, as shown in figure 5.19. This results in a considerable usage of computational power, with a Raspberry Pi 2 Model B+ running a lightweight Linux distribution, Ubuntu Server 18.04 LTS. All features were built upon the Flask[6]micro web framework.

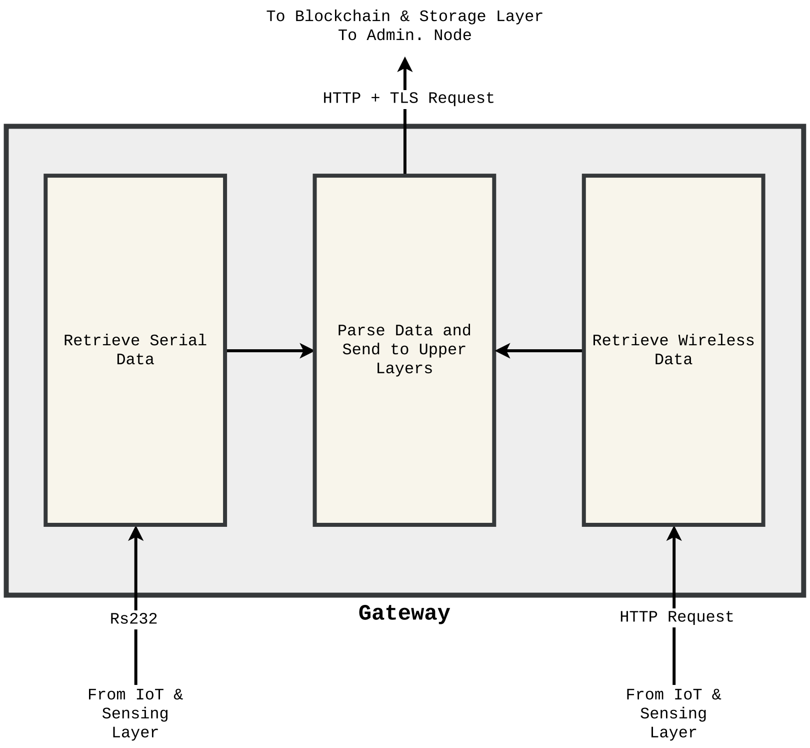


Figure 5.19: Processes Running in Gateway.

[6]Flask. Accessed 7 May, 2020. URL: `https://flask.palletsprojects.com/en/1.1.x/`

The Raspberry Pi 2 Model B+, figure 5.20, has the following, abridged, feature-set[7].

- 900MHz quad-core ARM Cortex-A7 CPU;
- 1GB RAM;
- 100 Base Ethernet;
- 4 USB ports;
- 40 GPIO pins.

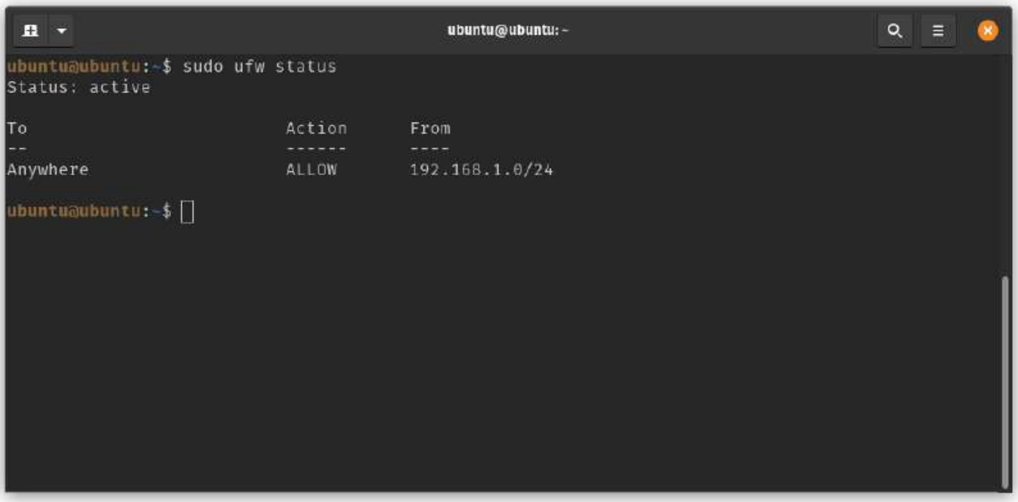
Figure 5.20: Raspberry Pi 2 Model B+.
Source[7].

Sending data to the upper Blockchain and Storage Layer requires use of external networks. As this poses high security risks, a firewall is implemented to avoid unsolicited connections to the Gateway. For implementation simulation and testing, the built-in Ubuntu firewall - **ufw** - is used, and network traffic restricted to only IP addresses operating within the same local network. Figure 5.21, shows the created firewall rules table. Likewise, the Raspberry's **/etc/hosts** file is reconfigured to accommodate the attributed common names of the created TLS/SSL certificates.

5.3.1 Program Operation

As previously mentioned, three processes occur simultaneously within each gateway device: a serial communication handler; a wireless communication handler; and a main process responsible for interfacing the upper layers. Incoming serial data is decoded and immediately sent to the main process. Alternatively, data-packets received by HTTP requests require first a decryption procedure before conveying the data. In the main process, data gets parsed into a normalized form-factor. Every data-parcel being sent by a gateway contains the respective business unit in which the inputs were gathered; the part or component identification - the RFID tag reading in case of the simulation, in real-life Renault CACIA it would be the barcode tag; a timestamp of when data first arrived at the gateway; and lastly the information itself making up the message payload.

[7]Raspberry Pi Organization - *Raspberry Pi 2 Model B.* Accessed 6 May, 2020. URL: `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`



```
ubuntu@ubuntu:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Anywhere                   ALLOW       192.168.1.0/24

ubuntu@ubuntu:~$
```

Figure 5.21: Firewall Rule Table.

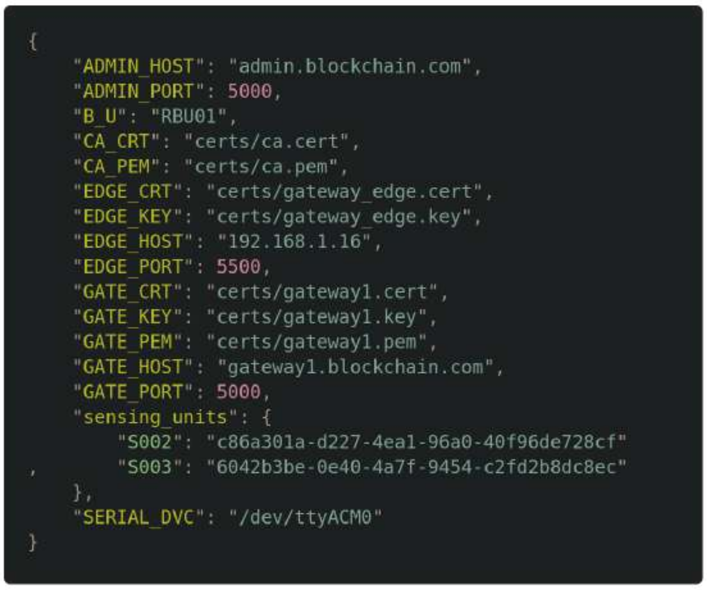
After finishing the data-processing stage, it is sent via a HTTP over TLS, i.e. HTTPS, POST-request to all the existing nodes on the network.

Communication between the Gateway Layer and the upper levels of the network is bi-directional. On one hand, the gateway is responsible for acting as an intermediary with the blockchain and the low-level hardware; on the other hand, gateways must also be made aware of the participants of the network to know where to send data to. When first adhering to a network, or whenever any changes occur to the list of active Mining Nodes, it is of the Blockchain Administrator's responsibility to notice gateways of these alterations. Whenever a HTTP request is performed, a response is expected. If this response is intercepted, hindering its arrival to the original sending device, the program execution comes to a halt, possibly jeopardizing the entire network. To avoid these Denial of Service attacks, all HTTPS requests performed on the network are subject to timeout periods, after which the sender stops expecting a response from the recipient, avoiding indefinite waiting periods.

5.3.2 Gateway Deployment

Creation of a modular and scalable solution demands not only forethought in questions regarding compatibility with a wide range of technologies, but also the ability to quickly and easily deploy and demise network elements, such as in this case, gateways. To facilitate deployment process, a configuration file exists - **config.json** - in which all customizable fields are present and can be quickly altered. When first starting one of the three processes required for gateway operation, they all search for the standard **config.json** file, however a user can provide a different configuration file by starting the program with –**config** or **-c** options, followed by the file-path. Second and last step to fully adhere a gateway to a network is done by the Blockchain Administrator when he officially registers a gateway's address into the Blockchain system. Figure 5.22 depicts an example of a configuration file.

Worthy of note is the existence of a **sensing_units** field in the configuration file.



```
{
    "ADMIN_HOST": "admin.blockchain.com",
    "ADMIN_PORT": 5000,
    "B_U": "RBU01",
    "CA_CRT": "certs/ca.cert",
    "CA_PEM": "certs/ca.pem",
    "EDGE_CRT": "certs/gateway_edge.cert",
    "EDGE_KEY": "certs/gateway_edge.key",
    "EDGE_HOST": "192.168.1.16",
    "EDGE_PORT": 5500,
    "GATE_CRT": "certs/gateway1.cert",
    "GATE_KEY": "certs/gateway1.key",
    "GATE_PEM": "certs/gateway1.pem",
    "GATE_HOST": "gateway1.blockchain.com",
    "GATE_PORT": 5000,
    "sensing_units": {
        "S002": "c86a301a-d227-4ea1-96a0-40f96de728cf"
,       "S003": "6042b3be-0e40-4a7f-9454-c2fd2b8dc8ec"
    },
    "SERIAL_DVC": "/dev/ttyACM0"
}
```

Figure 5.22: Example of a Gateway Configuration File.

Wireless sensors or IoT devices with their ID here declared are the only ones from which payloads are accepted by the gateway. This was implemented as to prevent unauthorized devices from pinging the network with malicious information. Furthermore, authorized serial ports also require an initial declaration - **SERIAL_DVC**. Removing a gateway, stopping it from further operating in the network, is done through the Blockchain Administrator's frontend interface.

5.4 Blockchain and Storage Layer

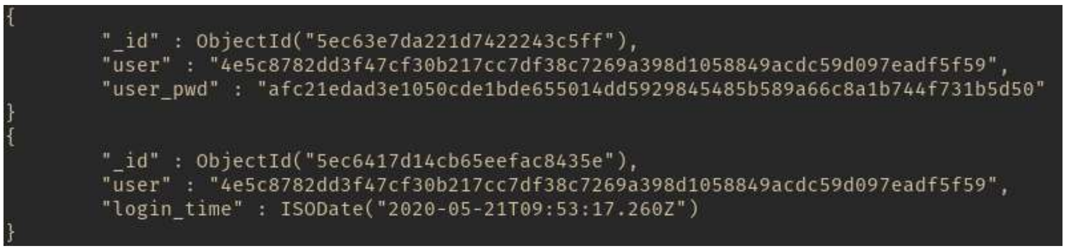
The Blockchain and Storage Layer has two main acting agents: the Database and the Mining Nodes. The Database contains a complete record of the gathered data, both the one generated by the IoT and sensing devices, as the one collected through network analytics systems. To simulate implementation deployment, three different types of collections within the database exist: the full record of all the data gathered by the sensing devices; the backup of the node's simplified distributed ledger (one per node); and the user access credentials and network activity journal.

Mining Nodes are responsible for forging new blocks, according to the established rule-set, and thus storing the data within an immutable shared ledger. Although each node does not store the entirety of the forged data-block, it does store the block's header, which contains enough information to fully identify a transaction latter on.

5.4.1 Database

The Database is based in MongoDB, a non-relational database. This grants a high degree of freedom on how the data is structured and accessed. For instance, instead of requiring two distinct tables in a traditional SQL database, both authorized users' credentials and users' login activities can be stored in a single, non-relational, collection. Of course, this comes at the cost of speed, as querying the database is not as fast as it otherwise would be with a SQL ledger. Nevertheless, this adds flexibility to the implemented framework, as gathered data does not have to follow an overly rigid structure.

The first of existing collections is the **User Collection**, which serves three main functions, namely storing usernames and respective passwords; login information such as a timestamp of when a login occurred; and the clearance level that each user has on the network (e.g., it could be a **Administrator** account or a **Client** account). Figure 5.23 exemplifies two possible entries found within the User Collection, one storing user credentials and another saving a new frontend access timestamp from a certain user.



```
{
        "_id" : ObjectId("5ec63e7da221d7422243c5ff"),
        "user" : "4e5c8782dd3f47cf30b217cc7df38c7269a398d1058849acdc59d097eadf5f59",
        "user_pwd" : "afc21edad3e1050cde1bde655014dd5929845485b589a66c8a1b744f731b5d50"
}
{

        "_id" : ObjectId("5ec6417d14cb65eefac8435e"),
        "user" : "4e5c8782dd3f47cf30b217cc7df38c7269a398d1058849acdc59d097eadf5f59",
        "login_time" : ISODate("2020-05-21T09:53:17.260Z")
}
```

Figure 5.23: Example of Data in the User Collection.

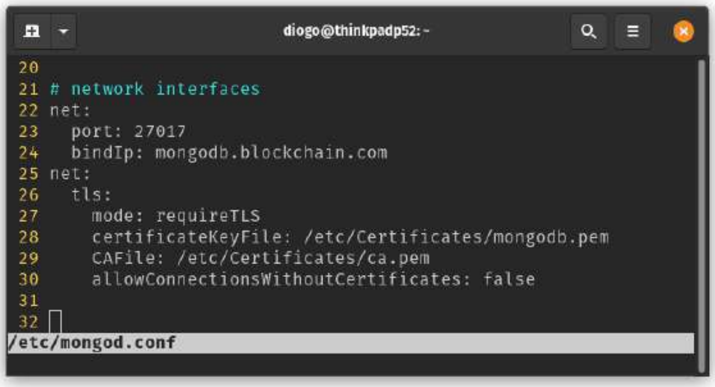
Both usernames and passwords are indirectly stored, through the storing of their corresponding hash-values. Information regarding each user's clearance level is also present, but hidden in plain-sight: before being hashed, the username is associated with a clearance level. This means that a malicious entity that somehow might have gained access to the network, would not even know which registries were of Client or Administrator accounts. A **Node Collection** is associated to every Mining Node on the network serving as a backup of the simplified distributed ledger. Full record of data is kept on the **Blockchain Collection**. This record could be replaced by existing traditional enterprise database systems, however, in this manner, storage of a wider range of data types that might not have been contemplated when the traditional systems where put into action, is feasible.

Secure Deployment of MongoDB

When deploying a MongoDB server, one can either use the standard configuration file or provide a custom one. As the default MongoDB set-up does not contemplate the use of secure TLS connections, a bespoke configuration file was created, by adding the following commands, figure 5.24, to the original **mongod.conf** file. This leads to ensuing behaviour from the MongoDB server:

- The server is bound to the *mongodb.blockchain.com* address, in accordance with the common name in its authentication certificate;
- TLS connections are now required to access the server, and connections without valid certificates are no longer allowed;
- Path to the server's own **.pem** file containing its private and public-key is specified, permitting server-side authentication;

- By stipulating a CA file in the configuration, client-side authentication is enabled.



```

# network interfaces
net:
  port: 27017
  bindIp: mongodb.blockchain.com
net:
  tls:
    mode: requireTLS
    certificateKeyFile: /etc/Certificates/mongodb.pem
    CAFile: /etc/Certificates/ca.pem
    allowConnectionsWithoutCertificates: false


/etc/mongod.conf
```

Figure 5.24: Modifications Done to MongoDB Configuration File.

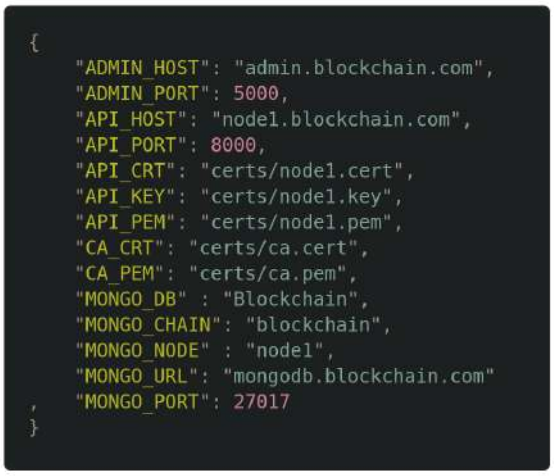
5.4.2 Mining Nodes

In the created architecture, mining nodes are not only responsible for creating and validating data blocks, through an agreed upon consensus mechanism, but are also capable of confirming past transactions. Data entries found in enterprise systems can easily be cross-checked with historical information gathered within the blockchain, simply by calculating the hash-value correspondent to the transaction wished to verify. If that hash-value can be found stored on-chain, then it is verifiably considered true. Centralized systems, such as traditional databases found in enterprises, inherently introduce single-points of failure. Data losses might occur for a wide range of reasons, since hardware failure to malicious tampering done by a third-party. Tampering in the blockchain, on the other hand, while not impossible, poses a much bigger challenge, as it is unlikely for several wide-spread machines to suffer simultaneous black-out, or in case of deliberate malice, to be synchronously attacked by a malign agent.

Mining nodes function at a core-level as web-servers who accept HTTPS requests via specific remote-addresses, and are, much like gateways, based on the Flask micro-framework with additional extensions used to provide integration with the MongoDB database. Comparable to what is seen in the deployment of gateways, mining nodes allow for a **config.json** file to expedite the configuration of several parameters, figure 5.25.

5.5 Blockchain Administrator

The Blockchain Administrator is also based on the Flask micro-framework, now using a number of additional extensions to add not only database integration, but also form validation (e.g., login forms) and user authentication. Operation of the administrator node is not intrusive to the remaining of the blockchain network, having no influence over the data being stored or on how the mining nodes validate transactions. However, administrators do have the power to add and remove nodes - and by extension gateways -



```
{
    "ADMIN_HOST": "admin.blockchain.com",
    "ADMIN_PORT": 5000,
    "API_HOST": "node1.blockchain.com",
    "API_PORT": 8000,
    "API_CRT": "certs/node1.cert",
    "API_KEY": "certs/node1.key",
    "API_PEM": "certs/node1.pem",
    "CA_CRT": "certs/ca.cert",
    "CA_PEM": "certs/ca.pem",
    "MONGO_DB" : "Blockchain",
    "MONGO_CHAIN": "blockchain",
    "MONGO_NODE" : "node1",
    "MONGO_URL": "mongodb.blockchain.com"
,   "MONGO_PORT": 27017
}
```

Figure 5.25: Example of a Mining Node Configuration File.

from the network, which is a hallmark of permissioned blockchain networks. Whenever a node is added to the blockchain system, a request for synchronization is sent to all nodes. This will ensure that all nodes will sync their internally kept simplified distributed ledgers, as to not disrupt the mining process. During this procedure, the agreed upon consensus mechanism, which is based in a pre-determined order for the next mining node, is also updated as to include the newly added nodes. This update will also happen in case one or multiple nodes are removed.

From a web-frontend standpoint, due to the innate exposure the blockchain administrator is subject to, added security measures are in use. Namely, Flask-Login extension is used to limit login attempts to one per remote address per second, dodging denial-of-service attacks through flooding of the web-server with multiple HTTPS requests. Routes on the web-server that are user accessible, such as *https://admin.blockchain.com:5000/settings* or *https://admin.blockchain.com:5000/dashboard* are also protected from being accessed if the user login procedure is bypassed. User credentials and network permission attributes are verified by the administrator, in conjunction with the User Collection kept in the NoSQL database.

Dynamic update of information shown in the frontend is handled by this web-server, along with the JavaScript/AJAX script embedded in the **.html** templates rendered. Most of the data processing is done at a server level and not on the users' browser. Data passed on to the user might, or not, be cross-checked with the data kept in the blockchain network beforehand. If data is requested directly using the *Query D.B.* function deployed in the frontend interface, it is checked with all nodes the validity of the block(s) being displayed. In case of data being used for real-time analytics, this step is skipped as to avoid further stressing the mining nodes with a high number of validation requests, hindering the block mining operation, and, therefore, block throughput. As all other Flask-based scripts deployed, a **config.json** file contains all configurable parameters.

5.6 User Layer

The amount of information shown to a given user will depend upon permissions granted to him by the corporation managing the network. For instance, it is not desirable or required that an external entity (e.g., an outside costumer or supplier) has access to all network analytics, but these are fundamental metrics for the systems administrator in charge with maintaining and supervising the system. As seen in figure 5.1, a Public and Private HMI are both implemented in the final solution, making up the web frontend and dashboard to the blockchain network.

Hypertext Markup Language (HTML) is used as the backbone of rendered web-pages, with a combination of JavaScript/AJAX to access the implemented API to retrieve data from the server. Stylization is provided mostly by custom Cascading Style Sheets (CSS), although the open source Bootstrap library was also lightly used. Figure 5.26(a) shows the final web frontend interface for an internal user account and figure 5.26(b) for an external user account.

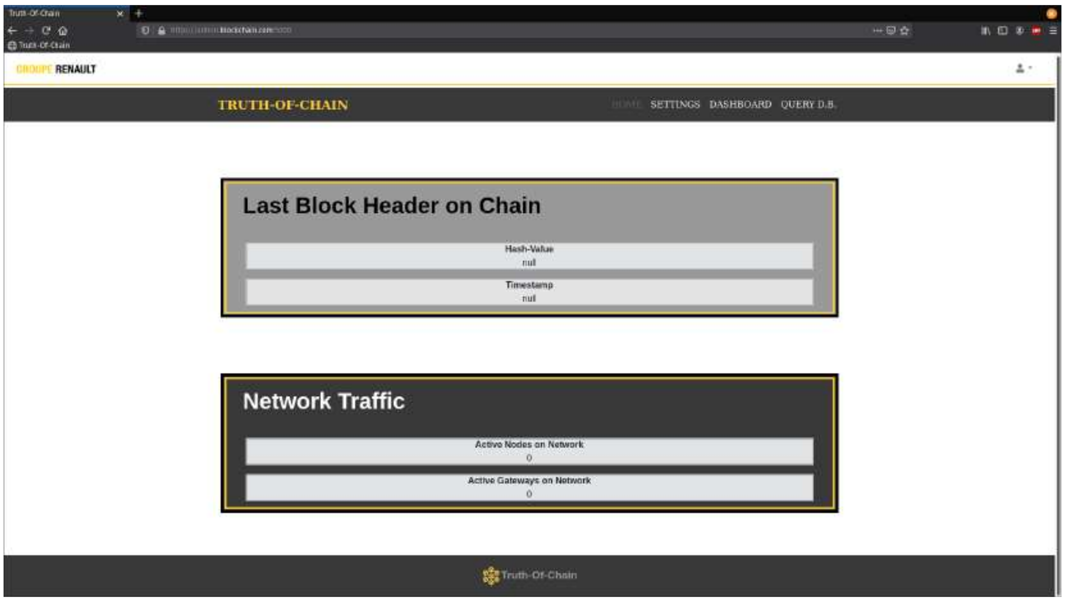
As seen in figures, the *Settings* and *Dashboard* pages are inaccessible to external accounts. Furthermore, if the address of these pages is inserted manually (i.e., *ht-tps://admin.blockchain.com:5000/settings* or *https://admin.blockchain.com:5000/dashboard*) in the browser URL search, unauthorized users will be re-routed back to the home-page. For a fully authorized user, the frontend has the following main features:

- Manage network settings such as adding and removing Mining Nodes; adding and removing Gateways; and adding and revoking user accounts.
- Real time blockchain system dashboard, with data visualization and plotting of main network parameters.
- Blockchain may be directly queried to check validity of a specific hash-value or find matches of a certain block number, business unit or part ID.

This last feature is also available to external users, in case it is desired to check historic values of a certain transaction. For example, a customer might be given this hash-value along with the purchased part or batch of parts. The costumer, wishing to validate the quality assurance given, merely has to query the hash-value from the blockchain. The process that happens server-side is illustrated in diagram 5.27.

Data queries to the blockchain can follow one of two processes: either a hash-value is given and it is directly used, checking its existence within all the simplified distributed ledgers in all nodes; or in case the query is done for a specific business unit, block number or part ID, possible hash-values are first retrieved from the centralized database - Blockchain Collection - and only then are those hash-values cross-checked along the blockchain. If hash-values match those stored in the distributed ledger, query response is displayed in the frontend interface.

From the *Dashboard* page, it is possible to see in real-time the last block forged and on the blockchain, network information such as total frontend visits or number of active user accounts, and graphed data of time between each block being forged and of the distribution of mined blocks in a per-node basis. Metrics directly being measured by the MongoDB server are also available, like the average block size. Plotting of data is done based on the Google Charts API, being fed dynamic data updates. Figure, 5.28, shows the designed dashboard in operation.



Last Block Header on Chain

Hash-Value
null

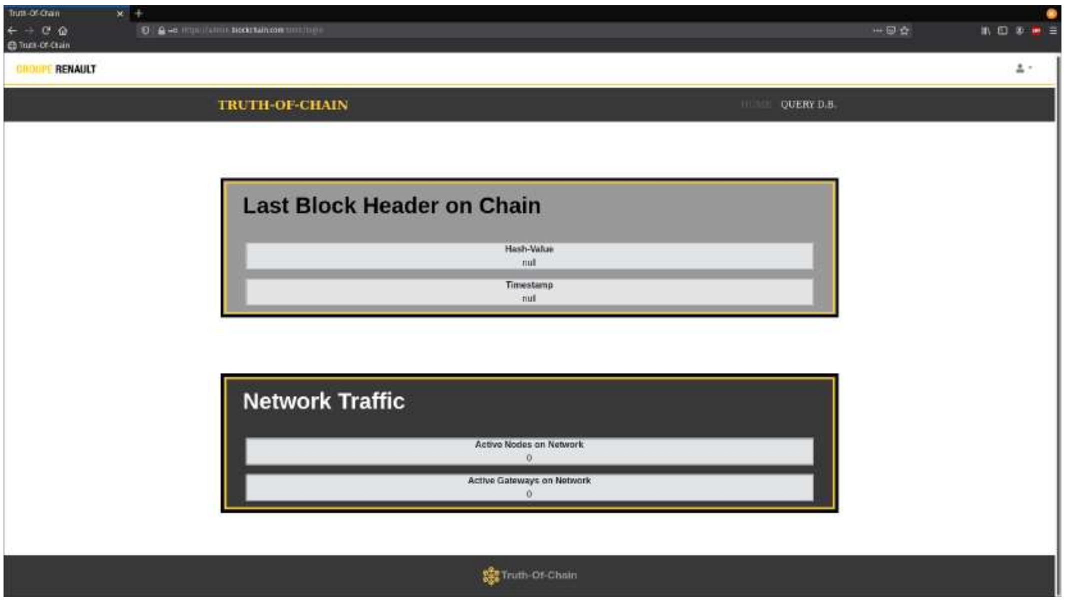
Timestamp
null

Network Traffic

Active Nodes on Network
0

Active Gateways on Network
0

(a) Internal Account Interface.



Last Block Header on Chain

Hash-Value
null

Timestamp
null

Network Traffic

Active Nodes on Network
0

Active Gateways on Network
0

(b) External Account Interface.

Figure 5.26: Frontend Interfaces for Different User Permissions.

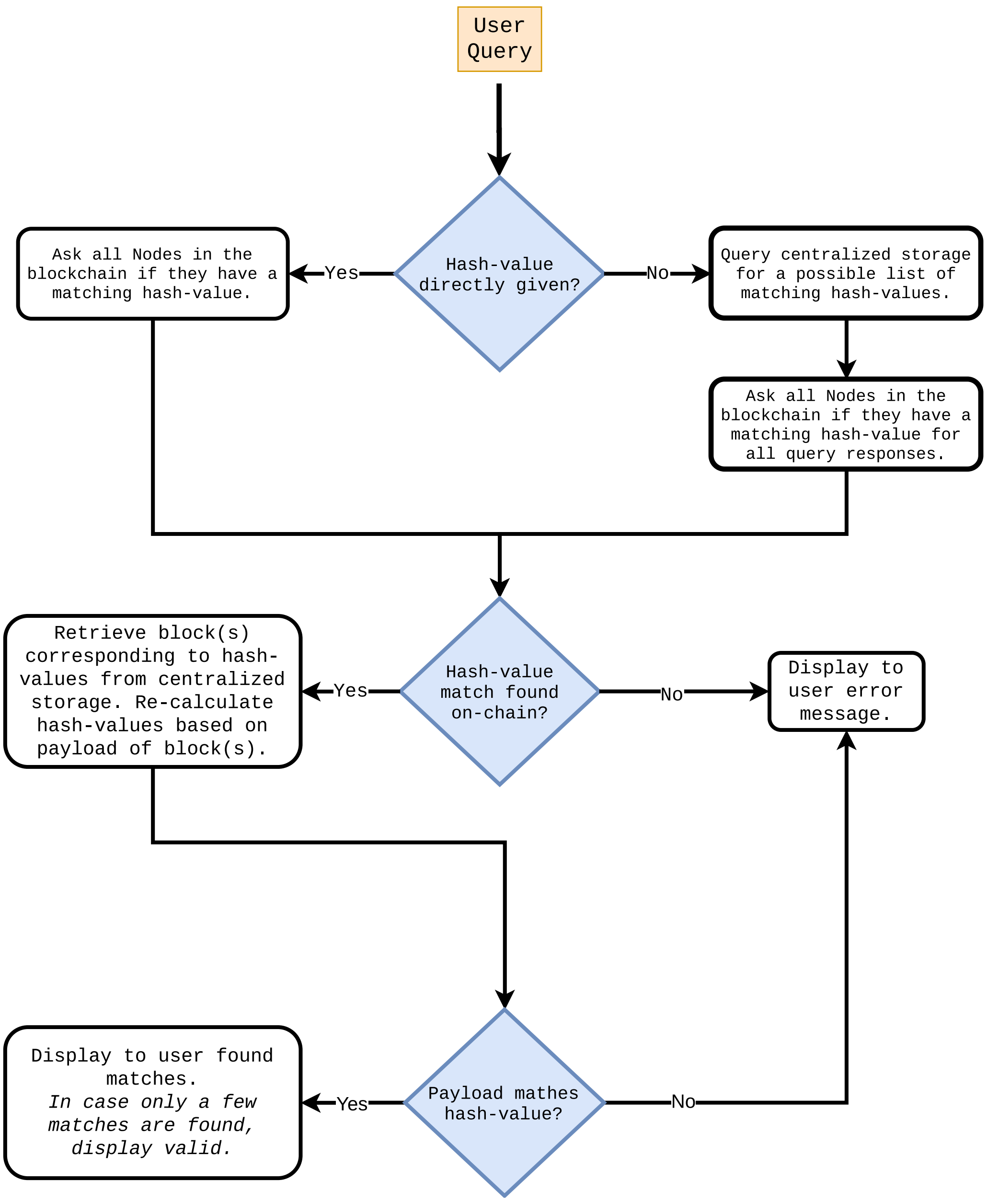

User Query
Hash-value directly given?
Yes
No
Ask all Nodes in the blockchain if they have a matching hash-value.
Query centralized storage for a possible list of matching hash-values.
Ask all Nodes in the blockchain if they have a matching hash-value for all query responses.
Hash-value match found on-chain?
Yes
No
Retrieve block(s) corresponding to hash-values from centralized storage. Re-calculate hash-values based on payload of block(s).
Display to user error message.
Payload mathes hash-value?
Yes
No
Display to user found matches. *In case only a few matches are found, display valid.*

Figure 5.27: Diagram of Querying Mechanism.

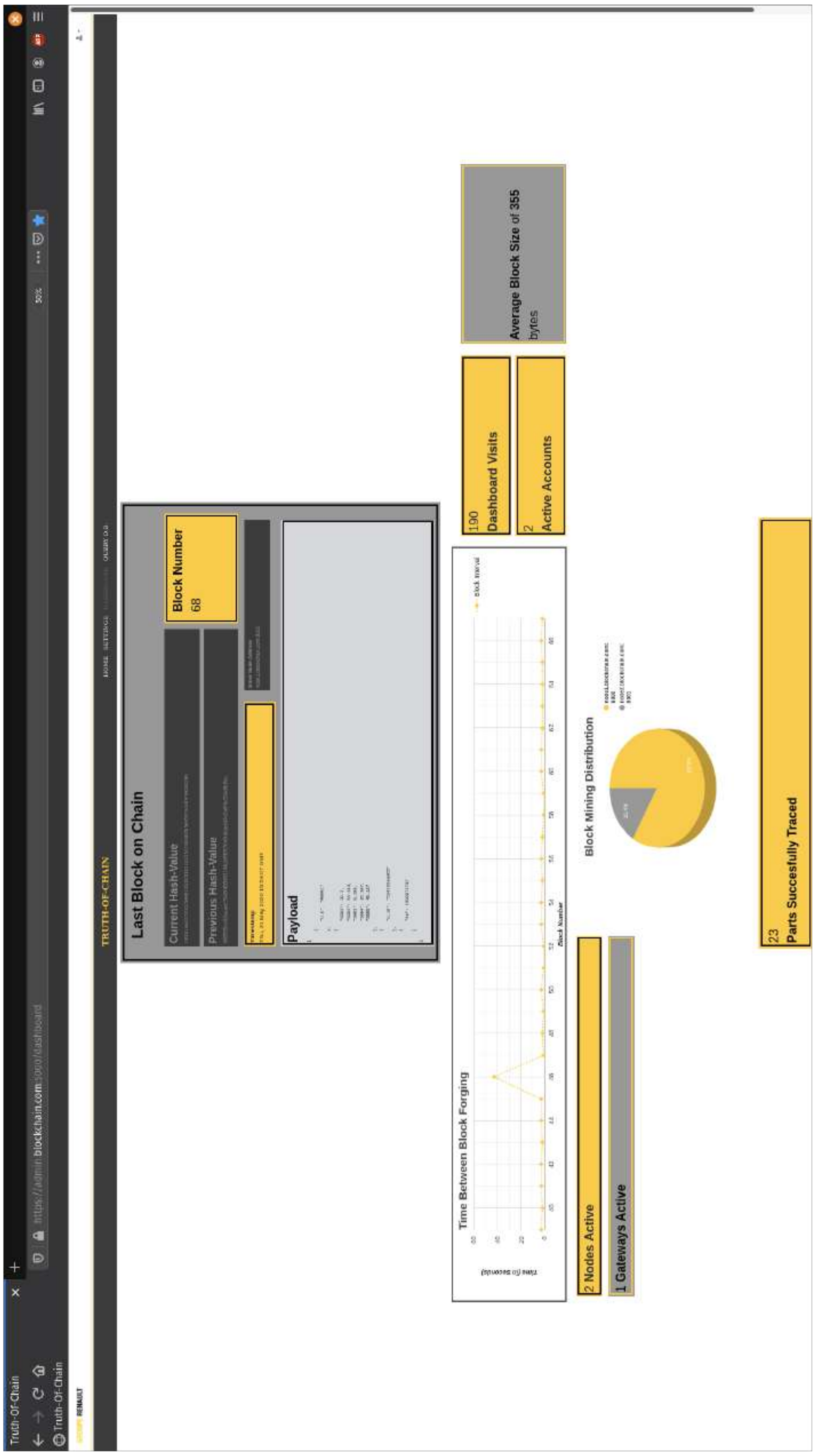


Figure 5.28: Designed Frontend Interface.

Chapter 6

Testing and Conclusions

For testing, two distinct methodologies were used: firstly, the aforementioned simulation platform was assembled and used to test the network's functionalities; secondly, a simulation program was devised to test response times and possible shortcomings of the architecture. As the platform, being based on physical hardware, requires manual input to operate, testing done by software allows for a greater exploit of the limits able to achieve.

6.1 Network Security Testing

In order to test if correct network protocols are in place, namely if connections being established are in fact TLS encrypted, Wireshark[1]was used. As all software procedures for communication follow the same routines, testing was done mostly on what might be considered one of the most data-sensible operations: user input of the login password. As seen in figure, 6.1, connections are in fact TLS encrypted. Highlighted in blue, is the message payload corresponding to the user login credentials.

Although version 1.3 of TLS was used in the web-servers, Wireshark displays TLS 1.2. This is likely caused by the not-yet optimized implementation of the newest TLS version in the Flask micro web framework. However, using Mozilla's Firefox Developer Tools[2], connections are shown being based on TLS 1.3, figure 6.2.

Lastly, connections performed using ESP devices also have encrypted payloads, albeit in a symmetric-key scheme. Figure 6.3 shows one of the ESPs' encrypted payloads, monitored used Wireshark.

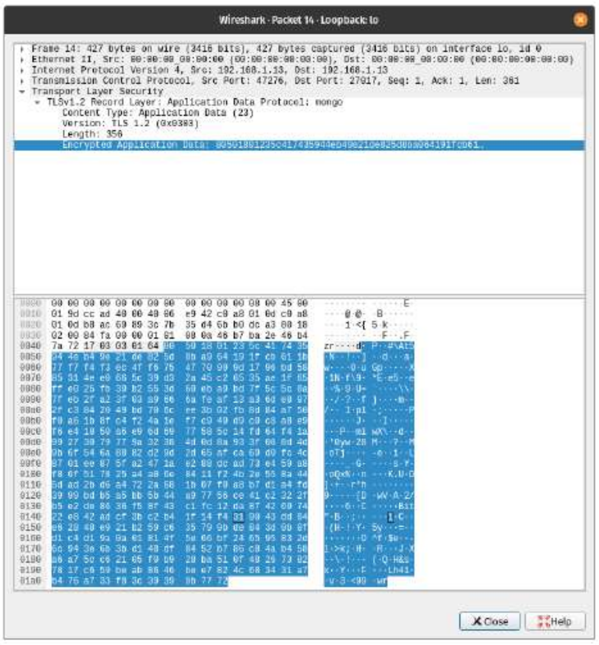
6.2 Network Performance Testing

6.2.1 Mining Node Performance

Performance testing with a varying amount of Mining Nodes was done by measuring response times of the *Query D.B.* feature of the frontend interface. Seeing that the network querying operation not only depends on the acting speed of the Blockchain

[1]Wireshark. Accessed 26 May, 2020. URL: `https://www.wireshark.org/`

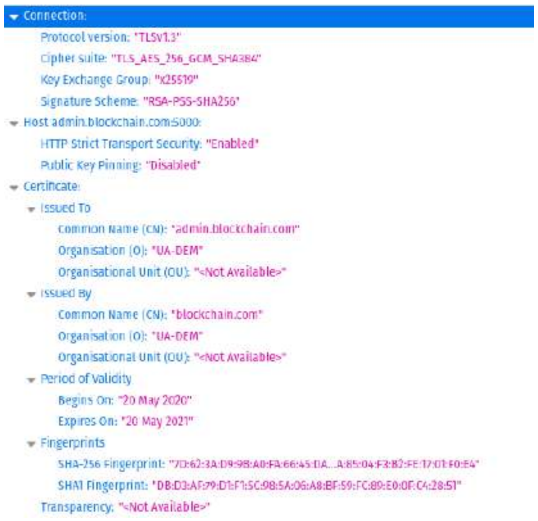
[2]MDN Web Docs - *Firefox Developer Tools*. Accessed 26 May, 2020. URL: `https://developer.mozilla.org/en-US/docs/Tools?utm_source=devtools&utm_medium=tabbar-menu`



```
▸ Frame 14: 427 bytes on wire (3416 bits), 427 bytes captured (3416 bits) on interface lo, id 0
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 192.168.1.13, Dst: 192.168.1.13
▸ Transmission Control Protocol, Src Port: 47276, Dst Port: 27017, Seq: 1, Ack: 1, Len: 361
▾ Transport Layer Security
  ▾ TLSv1.2 Record Layer: Application Data Protocol: mongo
      Content Type: Application Data (23)
      Version: TLS 1.2 (0x0303)
      Length: 356
      Encrypted Application Data: 00501801235c41743594 4eb49e21de825d0ba9 64191fcb61…
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ··············E·
0010  01 9d cc ad 40 00 40 06  e9 42 c0 a8 01 0d c0 a8   ····@·@··B······
0020  01 0d b8 ac 69 89 3c 7b  35 d4 6b b0 dc a3 80 18   ····i·<{5·k·····
0030  02 00 84 fa 00 00 01 01  08 0a 46 b7 ba 2e 46 b4   ··········F··.F·
0040  7a 72 17 03 03 01 64 00  50 18 01 23 5c 41 74 35   zr····d·P··#\At5
0050  94 4e b4 9e 21 de 82 5d  0b a9 64 19 1f cb 61 1b   ·N··!··]··d···a·
0060  77 f7 f4 f3 ec 4f f6 75  47 70 90 0d 17 06 bd 58   w····O·uGp·····X
0070  85 31 4e e0 66 5c 39 d3  2a 45 c2 65 35 ae 1f 65   ·1N·f\9·*E·e5··e
0080  ff e0 25 fb 39 b2 55 3d  60 eb a9 bd 7f 5c 5c 0a   ··%·9·U=`····\\·
0090  7f eb 2f a2 3f 03 a9 66  6a fe af 13 a3 6d e0 97   ··/·?··fj····m··
00a0  2f c3 84 20 49 bd 70 6c  ee 3b 02 fb 8d 84 a7 50   /·· I·pl·;·····P
00b0  f0 a6 1b 8f c4 f2 4a 1e  f7 c9 40 d9 c0 c8 a8 e9   ······J···]·····
00c0  f6 e4 10 50 a6 e9 6d 60  77 58 5c 14 fd 64 f4 1a   ···P··m`wX\··d··
00d0  99 27 38 79 77 9a 32 38  4d 0d 8a 93 3f 00 8d 4d   ·'8yw·28M···?··M
00e0  0b 6f 54 6a 80 82 d2 9d  2d 65 af ca 69 d9 fc 4c   ·oTj····-e··1··L
00f0  87 01 ee 87 5f a2 47 1a  e2 80 dc ad 73 e4 59 a8   ······G·····s·Y·
0100  f8 6f 51 78 25 a4 a8 6e  84 11 f2 4b 2e 55 8a 44   ·oQx%··n···K.U·D
0110  5d ad 2b d6 a4 72 2a 68  1b 07 f0 a8 b7 d1 a4 fd   ]·+··r*h········
0120  39 99 bd b5 a5 bb 5b 44  a9 77 56 ce 41 c2 32 2f   9·····[D·wV·A·2/
0130  b5 e2 da 86 36 f5 8f 43  c1 fc 12 da 87 42 69 74   ····6··C·····Bit
0140  22 e8 42 ad cf 3b c2 b4  1f 14 f4 31 90 43 dd 84   "·B··;·····1·C··
0150  e0 28 40 e9 21 b2 59 c6  35 79 9b de 84 3d 90 8f   ·(@·!·Y·5y···=··
0160  d1 c4 d1 9a 0a 01 81 4f  5e 66 bf 24 65 95 83 2d   ·······O^f·$e··-
0170  6c 94 3e 6b 3b d1 48 df  84 52 b7 86 c8 4a b4 58   l·>k;·H··R···J·X
0180  a6 a7 5c c6 21 65 f9 b9  28 ba 51 0f 48 26 73 82   ··\·!e··(·Q·H&s·
0190  78 17 c6 59 be ab 86 46  be e7 82 4c 68 34 31 a7   x··Y···F···Lh41·
01a0  b4 76 a7 33 f8 3c 39 39  0b 77 72                  ·v·3·<99·wr
```

Figure 6.1: Wireshark TCP Packet Analysis.

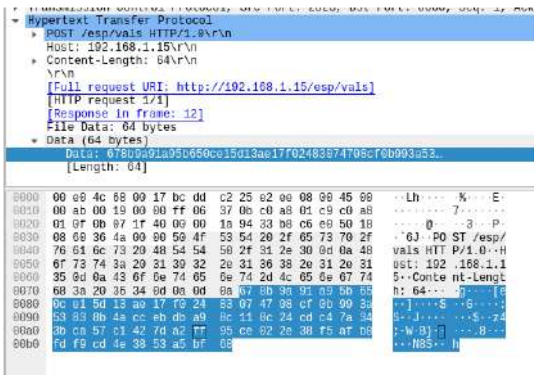


```
▾ Connection:
    Protocol version: "TLSv1.3"
    Cipher suite: "TLS_AES_256_GCM_SHA384"
    Key Exchange Group: "x25519"
    Signature Scheme: "RSA-PSS-SHA256"
▾ Host admin.blockchain.com:5000:
    HTTP Strict Transport Security: "Enabled"
    Public Key Pinning: "Disabled"
▾ Certificate:
  ▾ Issued To
      Common Name (CN): "admin.blockchain.com"
      Organisation (O): "UA-DEM"
      Organisational Unit (OU): "<Not Available>"
  ▾ Issued By
      Common Name (CN): "blockchain.com"
      Organisation (O): "UA-DEM"
      Organisational Unit (OU): "<Not Available>"
  ▾ Period of Validity
      Begins On: "20 May 2020"
      Expires On: "20 May 2021"
  ▾ Fingerprints
      SHA-256 Fingerprint: "7D:62:3A:D9:98:A0:FA:66:45:DA...A:85:04:F3:B2:FE:17:01:F0:E4"
      SHA1 Fingerprint: "DB:D3:AF:79:D1:F1:5C:98:5A:06:A8:BF:59:FC:89:E0:0F:C4:28:51"
    Transparency: "<Not Available>"
```

Figure 6.2: Firefox Developer Tools Connection Security Analysis.

Administrator, but also of the Mining Nodes and of the MongoDB database, it becomes representative of the latency seen all across the network. During this operation the Blockchain Administrator must maintain the frontend's functions whilst verifying hash-values all along the blockchain, and, likewise, Mining Nodes must maintain regular mining procedures as well as confirming hash-values and receiving new value submissions from Gateways.

To see how well adding additional Mining Nodes scales in the network, a test sample of 3, 5, 10, 15 and 20 nodes, along with one gateway pinging data at a random interval between 0.5 and 3 seconds, were used. The gateway had a payload of five sensor value



```
▾ Hypertext Transfer Protocol
  ▸ POST /esp/vals HTTP/1.0\r\n
    Host: 192.168.1.15\r\n
  ▸ Content-Length: 64\r\n
    \r\n
    [Full request URI: http://192.168.1.15/esp/vals]
    [HTTP request 1/1]
    [Response in frame: 12]
    File Data: 64 bytes
  ▾ Data (64 bytes)
      Data: 678b9a91a95b650ce15d13ae17f02483874708cf0b993a53…
      [Length: 64]

0000  00 e0 4c 68 00 17 bc dd  c2 25 e2 ee 08 00 45 00   ··Lh···· ·%····E·
0010  00 ab 00 19 00 00 ff 06  37 0b c0 a8 01 c9 c0 a8   ········ 7·······
0020  01 0f 0b 07 1f 40 00 00  1a 94 33 b8 c6 e0 50 18   ·····@·· ··3···P·
0030  08 60 36 4a 00 00 50 4f  53 54 20 2f 65 73 70 2f   ·`6J··PO ST /esp/
0040  76 61 6c 73 20 48 54 54  50 2f 31 2e 30 0d 0a 48   vals HTT P/1.0··H
0050  6f 73 74 3a 20 31 39 32  2e 31 36 38 2e 31 2e 31   ost: 192 .168.1.1
0060  35 0d 0a 43 6f 6e 74 65  6e 74 2d 4c 65 6e 67 74   5··Conte nt-Lengt
0070  68 3a 20 36 34 0d 0a 0d  0a 67 8b 9a 91 a9 5b 65   h: 64··· ·g····[e
0080  0c e1 5d 13 ae 17 f0 24  83 07 47 08 cf 0b 99 3a   ··]····$ ··G····:
0090  53 83 8b 4a cc eb db a9  8c 11 8c 24 cd c4 7a 34   S··J···· ···$··z4
00a0  3b ca 57 c1 42 7d a2 ff  95 ce 02 2e 38 f5 af b8   ;·W·B}·· ····8···
00b0  fd f9 cd 4e 38 53 a5 bf  68                        ···N8S·· h
```

Figure 6.3: Wireshark HTTP Packet Analysis of a ESP Device Communication.

readings. Request timings, i.e., the time taken for a request to be sent and a response to be received, was measure using Mozilla's Firefox Developer Tools. Each manner of blockchain query has a different impact on the network, for instance, queries that use a hash-value directly are comparably much faster than those done using a part ID. This is due to the amount of data being gathered - it is expected that a part ID returns more than one match on-chain vs. one exact result for the hash-value - and differing amounts of accesses required to the enterprise or centralized data-storage systems. Queries sampled for testing were of three types: by block number; by hash-value or using each part's ID. All of the shown results were based on the average of 100 samples for each one of the number-of-nodes combinations. Timings in queries by block number, plotted in figure 6.4, were the fastest seen during testing.

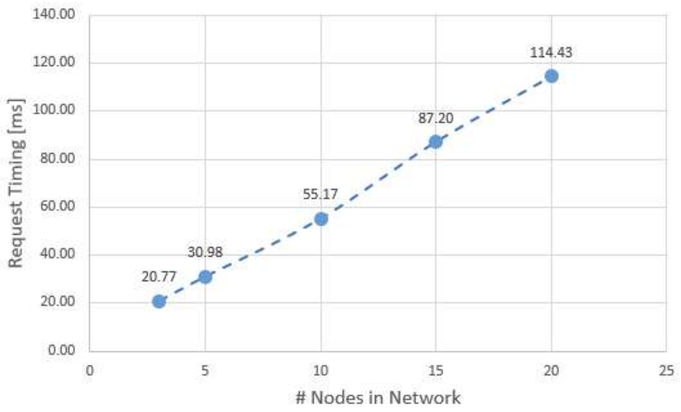
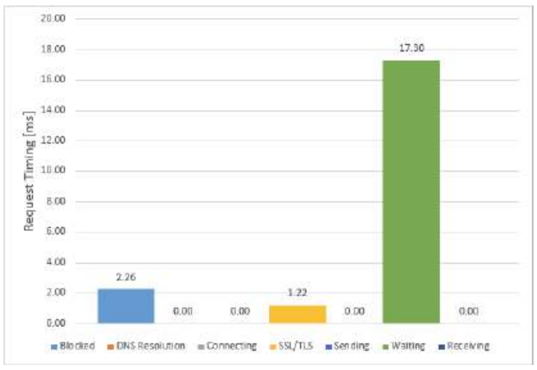


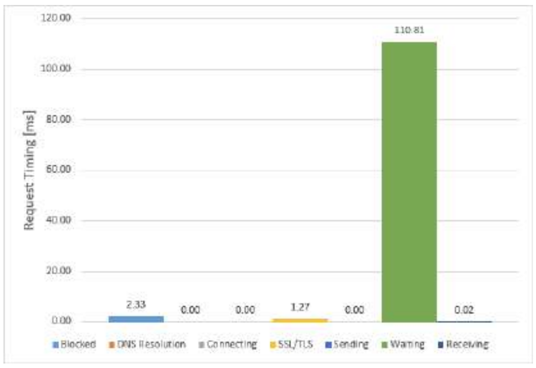
Figure 6.4: Request Timing in Query by Block Number.

The data clearly shows the linear increase of request processing times, with the number of nodes deployed. However, even for a somewhat large network, in terms of permissioned blockchain standards, latency times are bellow those that are noticeable for the end user. To see where the majority of time is being spent, for each request, distribution of time between each stage of a HTTP request was plotted for the most extreme scenarios - 3 and 20 nodes, figures 6.5(a) and 6.5(b) respectively.

It becomes clear that in both cases, the *Waiting* procedure is the biggest bottleneck,



(a) 3 Nodes on Network.



(b) 20 Nodes on Network.

Figure 6.5: Request Timing Subset Stages Division for Query by Block Number.

contemplating the time taken by the server, i.e., the Blockchain Administrator, to gather the queried data and check hash-values with each one of the Mining Nodes individually. Remaining of stages stay consistent with each other, regardless of number of nodes, which is to be expected. Timings for queries by hash-values took longer, compared to queries by block number, figure 6.6, in spite of a more reduced procedural workload.

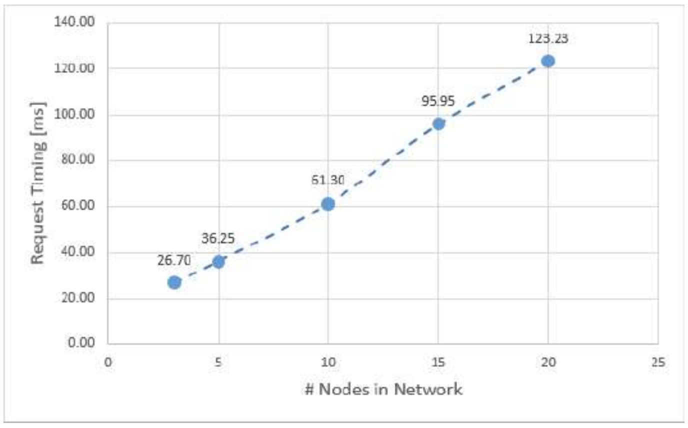
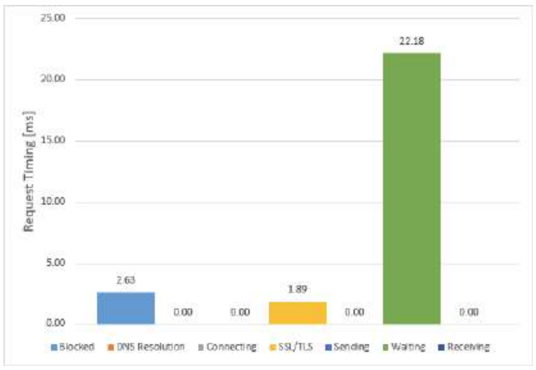
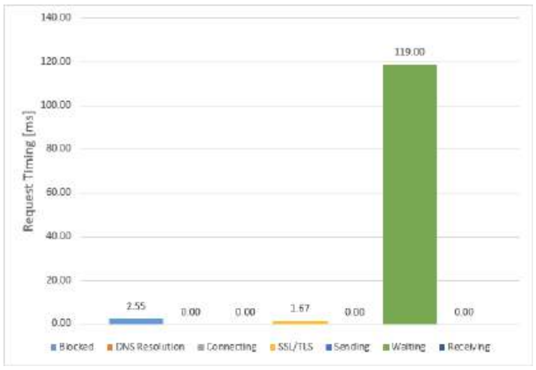


Figure 6.6: Request Timing in Query by Hash-Value.

However, analysis of the timing divisions for extreme cases, figures 6.7(a) and 6.7(b), hints at the possibility that additional delay might have been caused by network connection as both the *Blocked* and *SSL/TLS* set-up stages also took longer, and they are not dependent on server procedures.



(a) 3 Nodes on Network.



(b) 20 Nodes on Network.

Figure 6.7: Request Timing Subset Stages Division for Query by Hash-Value.

Lastly, plot of querying by Part ID, figure 6.8. This is the most work-intensive task that can be placed on the network. Not only are all nodes always asked to cross-check their internal ledgers with a given hash-values, as in the case of query by Part ID, typically, more than one hash-value has to be validated. For the purpose of testing, each query uses a part ID for which 3 results, blocks, are returned.

The trend of *Waiting* stage taking longest, continues, figures 6.9(a) and 6.9(b). As 3 blocks are returned to the user, this operation also takes roughly three times as much time to render results, compared to other queries. Nonetheless, user experience of the frontend is still perceived as quick.

On a final note, TLS connections introduce very little overhead in the final, implemented architecture. This is a testament to the Python language web-capabilities, and by extent the Flask web framework. Testing was also conducted with multiple gateways, showing little to none influence over query times. However, due to the synchronous-process nature of the Flask framework, complications could arise when two gateways submit a transaction for mining at, roughly, the same time. Furthermore, the architecture was also successfully deployed under a Windows10 environment, with very minimal alterations done. Only the configuration files required, as expected, small tweaking.

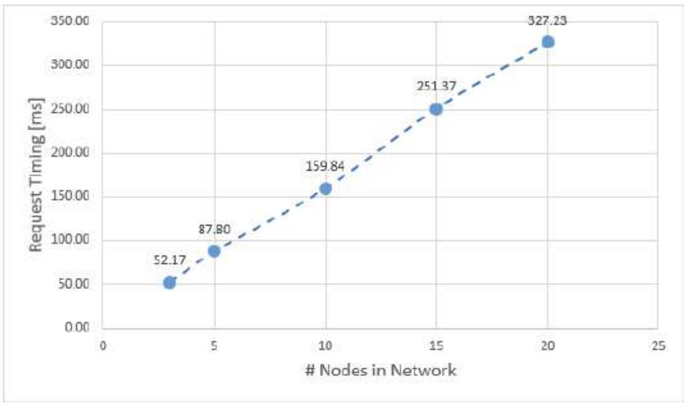
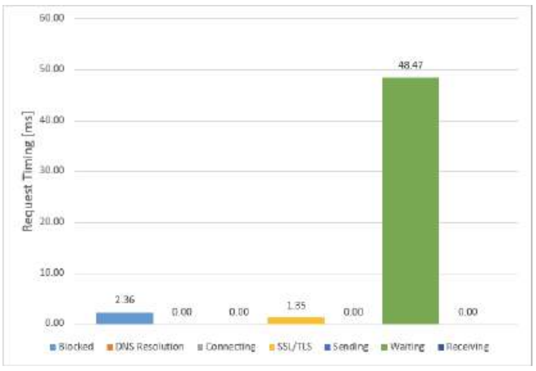
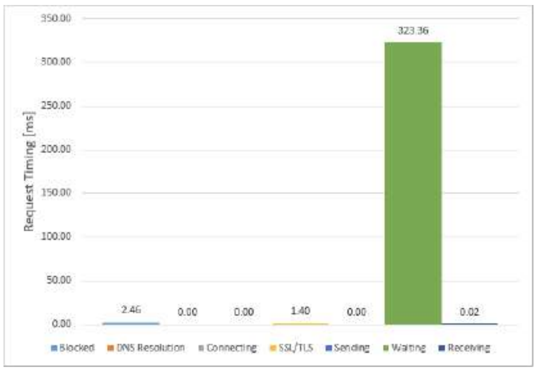


Figure 6.8: Request Timing in Query by Part ID.



(a) 3 Nodes on Network.



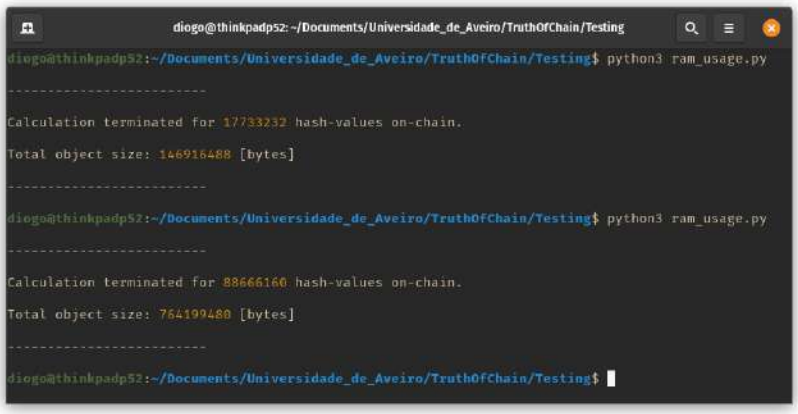
(b) 20 Nodes on Network.

Figure 6.9: Request Timing Subset Stages Division for Query by Part ID.

6.2.2 Mining Node RAM Usage

As Mining Nodes keep two simplified distributed ledgers, one on RAM and another as a backup in a MongoDB collection, RAM usage was measured. To test this, a simple Python script was devised to mimic procedures in the Mining Node. The hash-value of a string was calculated and then appended to a Python *dictionary*, representing what happens on a real node. As the output of a hash-function is always the same length, in this case 256 bits, the format or contents of the input string are not important. For a more realistic test, this appending process took place until the dictionary contained 17.733.232 objects within it. This value was not chosen at random. Considering the VDOP production line output of 1.616.112 yearly parts (as stated in section 2.2), and in a worst case scenario, that each one of the 11 OPs at Line 2 produce one block per part, 17.733.232 blocks per year are mined, and as such, the same number of hash-values are appended to the chain.

Dictionary object sizes were measured using Python's built-in **getsizeof()** function, present in the **sys** module. Returned values are in bytes. Python object sizes do not scale linearly, as additional overhead is given by the interpreter to avoid overflow. To study this, calculations were done for 17.733.232, representing one year of use, and for 88.666.160, representing five years of use. Obtained results are shown in figure 6.10.


diogo@thinkpadp52: ~/Documents/Universidade_de_Aveiro/TruthOfChain/Testing

```
diogo@thinkpadp52:~/Documents/Universidade_de_Aveiro/TruthOfChain/Testing$ python3 ram_usage.py

------------------------

Calculation terminated for 17733232 hash-values on-chain.

Total object size: 146916488 [bytes]

------------------------

diogo@thinkpadp52:~/Documents/Universidade_de_Aveiro/TruthOfChain/Testing$ python3 ram_usage.py

------------------------

Calculation terminated for 88666160 hash-values on-chain.

Total object size: 764199480 [bytes]

------------------------

diogo@thinkpadp52:~/Documents/Universidade_de_Aveiro/TruthOfChain/Testing$ 
```

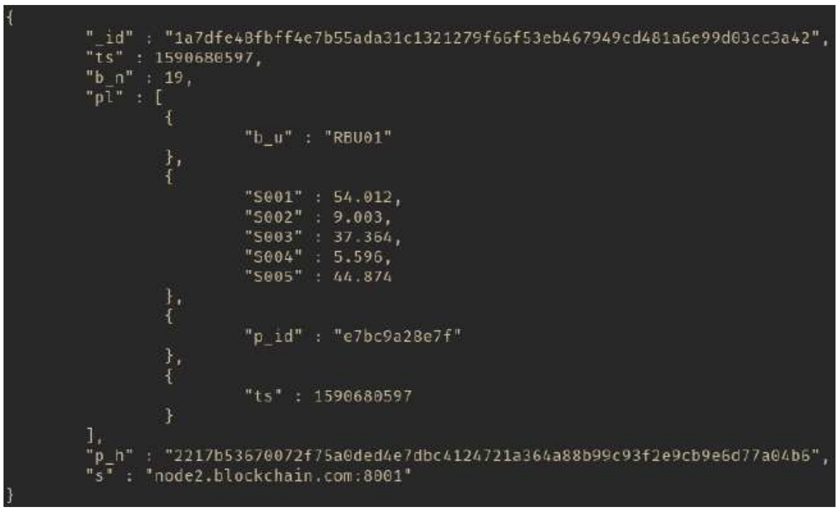
Figure 6.10: Results of RAM Usage Testing.

As seen, for an entire year's worth of block mining, space occupied in RAM for the simplified ledger stayed under 0.15 Gigabytes. Furthermore, the amount of overhead added is minimal, with the expected value for five years of use scaling roughly linearly with the one year value.

6.2.3 MongoDB Space Usage

Total storage space used by the MongoDB was tested to assess the viability of its use as an enterprise system, opposed to a traditional SQL ledger. To do this, several sizes of block payloads were used, simulating 5, 10, 20 and 50 sensor readings within a block. Figure 6.11 shows an example of a block with five of these sensor readings.

Using the MongoDB shell, it is possible to retrieve statistical data of the stored collections. Using this feature, total storage space used and average block size was collected. As MongoDB allocates space in an almost linear fashion, all tests were conducted for a sample of 1000 blocks, for later extrapolation of results. Table 6.1 summarizes found results. As stated in the MongoDB reference manual[3], Collection Size returns the total uncompressed size in memory of all records in a collection. An yearly estimate of space usage was calculated, shown in table 6.2, assuming the same value of 17.733.232 blocks per year. Even for the highest tested scenario, a value of around 20 Gigabytes per year



```
{
        "_id" : "1a7dfe48fbff4e7b55ada31c1321279f66f53eb467949cd481a6e99d03cc3a42",
        "ts" : 1590680597,
        "b_n" : 19,
        "pl" : [
                {
                        "b_u" : "RBU01"
                },
                {
                        "S001" : 54.012,
                        "S002" : 9.003,
                        "S003" : 37.364,
                        "S004" : 5.596,
                        "S005" : 44.874
                },
                {
                        "p_id" : "e7bc9a28e7f"
                },
                {
                        "ts" : 1590680597
                }
        ],
        "p_h" : "2217b53670072f75a0ded4e7dbc4124721a364a88b99c93f2e9cb9e6d77a04b6",
        "s" : "node2.blockchain.com:8001"
}
```

Figure 6.11: Example of a Block With Five Sensor Readings. As Stored in MongoDB.

of storage is not impractical, although higher than what would be seen if used a SQL database.

# Payload Values	Collection Size [Bytes]	Average Block Size [Bytes]
5	359120	358
10	429120	428
20	570120	569
50	989120	988

Table 6.1: MongoDB storage space used with varying number of payload values.

# Payload Values	Yearly Estimate [Gigabytes]
5	6.35
10	7.59
20	10.09
50	17.52

Table 6.2: Estimate of yearly storage space used by MongoDB, for each number of payload values.

6.3 Conclusions

Overall, the devised architecture performs adequately, accomplishing all initially set-out objectives. Mining and access of data happens at a fast pace, allowing for fast transaction certification. By choosing a highly supported general-use programming language,

[3]MongoDB Reference Manual - *collStats*. Accessed 27 May, 2020. URL: `https://docs.mongodb.com/manual/reference/command/collStats/#output`

Python, for the framework's foundations, it not only allowed for compatibility across multiple platforms, as also supports future expansion of the architecture by easily adding or improving functionalities. Compatibility with multiple different interfacing devices, such as sensors or IoT, was also assured. With minimal alteration, this framework could be used to highly strengthen the security of IoT powered network architectures, typically associated with serious security shortcoming. Security measures were implemented all throughout the solution, from device-to-device communications; personal user-data; to the securing of used databases. All selected security and cryptographic protocols, are among, if not the very best, currently on the market.

Choosing MongoDB for data-storage has both advantages and disadvantages. For one, not requiring a strict data format is an obvious selling point, allowing to miss-match different information all within the same collection. This does come at the cost of storage space and query speed. If a SQL database were to be used, total storage space would be considerably smaller. However, during all the tests performed, database read and write speeds were never an issue; even when a very specific query was made (e.g., a certain payload value within a block). To reduce storage size, small alterations were done, such as using hash-values as *_id* fields, and by shortening the name of each field (e.g., *pl* instead of *payload*).

For a commercial deployment, instead of using the built-in web-server of the Flask micro framework, a commercial grade server should be used. This transaction process is eased, as Flask's main objective is to be deployed within a Web Server Gateway Interface (WSGI) server, such as Heroku, Microsoft Azure, Google App Engine or Apache. Nonetheless, to manage multiple gateways submitting data, within a very short interval of each other, an asynchronous server should be deployed. Currently, this is the biggest limitation in the framework. Another alternative, would be the use of a queuing method, to avoid having two nodes mining blocks at an almost simultaneous time. Node scalability and performing real-time changes to the network layout is also guaranteed, as special care was given into adding a degree of abstraction between each network component. Lastly, this is a cost-effective way of strengthening any existing network, with the only expenditure being associated with cost of hardware used for testing and time required for development, as all software is open-source.

6.4 Future Work

Combined scholar and enterprise work has made possible new advances in quantum computing. Recent efforts have proven that quantum technology has the potential to largely outperform current computational platforms, solving in a matter of a few seconds, problems that would otherwise take several hundred years [37]. Although, existing quantum processors are not yet fully developed, it is expected that they will be capable of breaking Elliptic Curve Digital Signature Algorithm (ECDSA) digital signature schemes within the next decade [38]. Post-Quantum digital signature schemes then emerge as having the potential to withstand quantum attacks.

These post-quantum cryptographic primitives have several tradeoffs when compared with traditional algorithms, often-times having significantly slower computation times. Nevertheless, Google, Cloudflare and Microsoft have all reported positive results of integrating post-quantum key exchange algorithms with the current TLS 1.3 protocol [39].

Likewise, NIST Post-Quantum Cryptography Standardization project has began work in preparing the existing TLS mechanisms for post-quantum cryptography; and a fork of OpenSSL called OQS-OpenSSL has been created to provide an open-source tool-set for prototyping quantum-resistant cryptography[4].

With these new emerging technologies, it is not unfeasible the integration of quantum-resistant cryptography into the devised architecture as to ensure confidence in a long standing deployment of the platform, particularly with the use of newer, quantum compliant, versions of TLS.

[4]Open Quantum Safe - *Open Quantum Safe*. Accessed 26 June, 2020. URL: `https://openquantumsafe.org/`

Bibliography

[1] HANG, Lei; KIM, Do Hyeun - *Design and implementation of an integrated IoT blockchain platform for sensing data integrity.* Sensors (Switzerland). ISSN 14248220. 19:10 (2019). doi: 10.3390/s19102228.

[2] BECK, Roman et al. - *Blockchain Technology in Business and Information Systems Research.* Business and Information Systems Engineering. ISSN 18670202. 59:6 (2017). 381–384. doi: 10.1007/s12599-017-0505-1.

[3] ROCHA, D.F.J. - *Proposal of an automatic traceability system, in industry 4.0.* Universidade de Aveiro (2018). URL: `http://hdl.handle.net/10773/26048`

[4] Renault Portugal - Groupe Renault. - *Renault Cacia.* Accessed 9 April, 2020. URL; `https://www.renault.pt/renault-cacia.html`

[5] HEIZER, Jay; RENDER, Barry; MUNSON, Chuck – *Operations Management – Sustainability and Supply Chain Management.* ISBN 9780134130422.

[6] STASA, Pavel et al. - *Ensuring the Visibility and Traceability of Items Through Logistics Chain of Automotive Industry based on AutoEPCNet Usage.* (2016) 378–388. doi: 10.15598/aeee.v14i4.1788.

[7] Eclipse Foundation - *Open Source Software for Industry 4.0.* Accessed 18 February, 2020. URL: `https://iot.eclipse.org/white-papers/industry40/`

[8] GILCHRIST, Alasdair - *Industry 4.0: The Industrial Internet of Things.* Apress Media (2016). ISBN: 9781484220467.

[9] Eclipse Foundation - *The Three Software Stacks Required for IoT Architectures.* Accessed 18 February, 2020. URL: `https://iot.eclipse.org/white-papers/iot-architectures/`

[10] Smart Factory Task Group - *Smart Factory Applications in Discrete Manufacturing - An Industrial Internet Consortium White Paper.* Industrial Internet Consortium. (2017). IIC:WHT:IS2:V1.0:PB:20170222. Available at: `www.iiconsortium.org`

[11] MENEZES, Alfred et al. - *Handbook of Applied Cryptography.* CRC Publications (1997).

[12] SCHNEIER, Bruce et al. -*Cryptography Engineering: Design Principles and Practical Applications.* Wiley Publishing, Inc. (2010). ISBN: 978-0-470-47424-2

[13] FOROUZAN, Behrouz A. - *TCP/IP Protocol Suite (4th Edition).* McGraw-Hill (2010). ISBN: 978-0-07-337604-2.

[14] MCKAY, Kerry A.; COOPER, David A. - *Guidelines for the Selection , Configuration , and Use of Transport Layer Security (TLS) Implementations* National Institute of Standards and Technology Special Publication 800-52 Revision 2 (2019)). Publication available at: `https://doi.org/10.6028/NIST.SP.800-52r2`

[15] MOSCA, Michele - *Cybersecurity in a quantum world: will we be ready?* NIST. April (2015) 13–16.

[16] STEBILA, Douglas; MOSCA, Michele - *Post-quantum Key Exchange for the Internet and the Open Quantum Safe Project. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* ISSN 16113349. 10532 LNCS:2017) 14–37. doi: 10.1007/978-3-319-69453-5_2.

[17] COARFA, Cristian; DRUSCHEL, Peter; WALLACH, D. A. N. S. - *Performance Analysis of TLS Web Servers.* 24:1 (2006) 39–69.

[18] NAKAMOTO, Satoshi – Bitcoin: A Peer-to-Peer Electronic Cash System. Available at: `https://bitcoin.org/bitcoin.pdf`. Accessed: 2020-05-18.

[19] RESTUCCIA, Francesco et al. - *Blockchain for the Internet of Things: Present and Future.* November (2019).

[20] XU, Xiwei; WEBER, Ingo; STAPLES, Mark - *Architecture for Blockchain Applications.* ISBN 9783030030346.

[21] WANG, Huaimin et al. - *Blockchain challenges and opportunities: a survey.* International Journal of Web and Grid Services. ISSN 1741-1106. 14:4 (2018) 352. doi: 10.1504/ijwgs.2018.10016848.

[22] ZHENG, Zibin et al. - *An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends.* (2017). doi: 10.1109/BigDataCongress.2017.85.

[23] ANGELIS, Stefano D.E. et al. - *PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain.* CEUR Workshop Proceedings. ISSN 16130073. 1:11 (2018).

[24] LAMPORT, Leslie; SHOSTAK, Robert; PEASE, Marshall - *The Byzantine Generals Problem.* ACM Transactions on Programming Languages and Systems (TOPLAS). ISSN 15584593. 4:3 (1982). 382–401. doi: 10.1145/357172.357176.

[25] Coelho, P.T. - *A Federated Ledger for Regulated Self-Sovereignty.* Universidade de Aveiro (2018). URL: `http://hdl.handle.net/10773/25973`

[26] YANG, Lu. - *Journal of Industrial Information Integration The Blockchain: State-of-the-art and research challenges.* 15:01 (2019). 80-90. doi: 10.1016/j.jii.2019.04.002.

[27] BAHGA, Arshdeep; MADISETTI, Vijay K. - *Blockchain Platform for Industrial Internet of Things.* (2016). 533–546. doi: 10.4236/jsea.2016.910036.

[28] WAN, Jiafu et al. - *A Blockchain-Based Solution for Enhancing Security and Privacy in Smart Factory.* 15:6 (2019). 3652–3660.

[29] KNIRSCH, Fabian; UNTERWEGER, Andreas; ENGEL, Dominik - *Implementing a Blockchain From Scratch: Why, How, and What We Learned.* EURASIP Journal on Information Security (2019). doi:10.1186/s13635-019-0085-3.

[30] Hyperledger Project Website - *About Hyperledger.* Accessed 17 February, 2020. URL: `https://www.hyperledger.org/about`

[31] Hyperledger-Fabric Website - *Hyperledger Projects - Fabric.* Accessed 17 February, 2020. URL: `https://www.hyperledger.org/projects/fabric`

[32] Ethereum Website - *Ethereum.* Accessed 17 February, 2020. URL: `https://ethereum.org/`

[33] Official Solidity Documentation. Accessed 21 February, 2020. URL: `https://solidity.readthedocs.io/`

[34] AgriDigital Project Home-Page, URL: `https://www.agridigital.io/`

[35] NXP LTD. - *MFRC522 Standard Performance MIFARE and NTAG Frontend.* Rev. 3.9 (2016). Available at URL: `https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf`

[36] ESPRESSIF - *ESP32 Series Datasheet.* Espressif Systems. (2019) 1–61.

[37] ARUTE, Frank et al. - *Quantum supremacy using a programmable superconducting processor. Nature.* ISSN 14764687. 574:7779 (2019) 505–510. doi: 10.1038/s41586-019-1666-5.

[38] SHAHID, Furqan; KHAN, Abid - *Smart Digital Signatures (SDS): A post-quantum digital signature scheme for distributed ledgers.* Future Generation Computer Systems. ISSN 0167739X. 111:2020) 241–253. doi: 10.1016/j.future.2020.04.042.

[39] PAQUIN, Christian; STEBILA, Douglas; TAMVADA, Goutam - *Benchmarking Post-quantum Cryptography in TLS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* ISSN 16113349. 12100 LNCS:2020) 72–91. doi: 10.1007/978-3-030-44223-1_5.

[40] TOLLERVEY, Nicholas H. - *Programming with MicroPython.* O'Reilly Media, Inc. (2017) ISBN 9781491972731.

[41] MicroPython Organization - Download Page. Accessed 25 January, 2020. URL: `https://micropython.org/download`

[42] MicroPython Organization - Documentation and Reference Guides for MicroPython. Accessed 25 January, 2020. URL: `http://micropython.org/`

[43] Official Python 3 Online Documentation. Accessed 25 January, 2020. URL: `https://docs.python.org/3/`

[44] Official JSON Online Documentation. Accessed 25 January, 2020. URL: `https://www.json.org/json-en.html`

[45] Fritzing Eletrical Design. Accessed 25 January, 2020. URL: `https://fritzing.org/`

[46] SZABO, Nick. - *Smart Contracts.* (1994). Accessed 11 February, 2020. URL: `http://szabo.best.vwh.net/smart.contracts.html`

[47] IBM - *Cheat sheet: What is Digital Twin?* Accessed 8 April, 2020. URL: `https://www.ibm.com/blogs/internet-of-things/iot-cheat-sheet-digital-twin/`

Appendices

Appendix A

MicroPython

A.1 What is MicroPython?

MicroPython, or μPython, was created by Damien George, and is a reimplementation of the Python programming language, developed for use with microcontrollers and embedded systems [40]. Most features of Python 3 are present in MicroPython, such as Python's style of object orientation; data types and data structures; Python objects; functions as first-class objects; and exception handling.

MicroPython is designed to work under extremely constrained devices, some with as little as 16 kilobytes of Random Access Memory (RAM). As such, concessions were made, namely, the full Python library is not present by default, although it is possible to import desired modules later. Standard libraries included in MycroPython's firmware will vary according to the device at hand, but modules for interacting with low-level hardware; GPIO pins; and peripherals are universal.

Compared with other programming languages traditionally used in constrained devices, like for instance the C language, MicroPython runs slower as additional overheads are required. However, in many situations, MicroPython might actually outperform C, as prototyping is much faster due to the wide range of built-in features inherited from conventional Python.

In any case, it is still possible to write MicroPython modules entirely in C, as well as using inline assembler, for use cases that require every-bit of speed or memory efficiency.

A.2 Installing MicroPython Firmware on ESP8266/32 Devices

A wide range of embedded and IoT devices are capable of running MicroPython, among the most notable are:

PyBoard Development board designed and sold directly by the MicroPython Organization. It was the first device to have MicroPython support, and is one of the most powerful of this list.

BBC micro:bit Beginner computing device, created by the British Broadcasting Company (BBC).

Adafruit Circuit Playground Express Development board with a high number of built-in inputs/outputs. This board runs a fork of MicroPython called CircuitPython, developed by Adafruit themselves.

ESP8266/32 Two of the most well-known IoT devices, with the ESP32 being an evolution of the ESP8266. They are both powerful boards that include a full TCP/IP network stack built-in.

All of the previously mentioned devices run MicroPython natively, with the exception of the ESP8266/32 boards. For these two microcontrollers, manual installation of firmware is required.

Firstly, download the most recent build of the firmware from MicroPython's release page [41]. The appropriate build must be chosen considering the board and device variant. If the wrong firmware is used, an error will appear during flashing.

To flash the firmware, the device must be in boot-loader mode. Placing a device in boot-loader mode depends upon the device itself, but as a general rule of thumb, if a built-in USB connector is present, the process happens automatically.

A Python utility is required to start flashing the firmware, or alternatively, by using tools provided by an appropriate Integrated Development Environment (IDE) (further details on section A.3). To use the utility, first install it by typing the following command in the terminal/command line:

```
$ pip install esptool
```

Note that, dependent on what Python version is used, "pip3" might be necessary. Once esptool.py is installed, erase the flash memory, and with it previously installed firmwares:

```
$ esptool.py --port PORT erase_flash
```

PORT should be replaced with the actual port in the PC to which the device is connected. In Windows this would be a numbered COM port (e.g., COM4) and in *NIX operating systems it would be a tty interface (e.g., /dev/ttyUSB0).

Next, the firmware can finally be flashed:

```
$ esptool.py --port PORT --baud 460800 write_flash \
  --flash_size=detect 0 PATH/TO/firmware.bin
```

"PATH/TO/" must state the path to the firmware. Additionally, the name of the firmware will replace "firmware.bin". If any error occurs, reduce the baudrate to 115200 bauds.

To check if the board is now MicroPython complaint, start the Read, Evaluate, Print, Loop (REPL):

```
$ picocom --baud=115200 /dev/ttyUSB0
```

If no errors are prompted, check the pre-installed modules for the used device:

```
>>> help('modules')
```

A full list of modules should now be printed to the Command Line Interface (CLI).

A.3 MicroPython IDEs

MicroPython is still very much in its infancy, and third-party support is yet hard to find. As such, not many options for IDEs exist, however the following are available:

Thonny IDE A Python development IDE that also supports MicroPython.

PyCharm-Community Another Python development IDE, it is possible to install a plug-in allowing for MicroPython frameworks. Community is freeware, but a paid version also exists.

μ**PyCraft IDE** An IDE specifically designed to work with MicroPython.

All of above mentioned software allows scripting and transfer of programs; device filesystem browsing; firmware installation tools; and live REPL. Other viable options exist, but were not tested. Out of the three mentioned, Thonny IDE seems the most stable under *NIX environments and the easiest to use to browse MicroPython filesystems.

Complementary, and not an IDE, WebREPL is instead a web application hosted at MicroPython's website that allows access to the device's filesystem and live REPL.

A.4 MicroPython Code

MicroPython syntax varies very little from that of standard Python, with most differences residing on the availability of modules.

Some of the modules built-in to the firmware will have the μ prefix, indicating that it is a MicroPython variation of a standard Python module. Usually, most features are ported over, with a few deprecated or niche-use functions being removed, and others adapted for use in constrained devices.

Modules will also vary from device to device. For instance, accessing an analogue pin in the ESP32 is done in a slightly different manner compared to the ESP8266.

Lastly, there are two special file names: **boot.py** and **main.py**. When a boot.py file is uploaded to the device, the script is *always* ran after a hard-reset or startup. Any variables declared in boot.py will stay in memory.

After boot.py script ends, the device will automatically run main.py - the main loop is declared here. If no boot.py file is present, the device will search for main.py instead on start-up.

Other file names can be used, but they will require a manual run command. Best practice would be to use the boot.py file for constant variable declaration and module import; main.py for scripts with the main loop; and any other name for user created or non-standard modules wished to import.

GPIO Control

```
# main.py

# Code compatible with both
# the ESP32/8266

# machine module allows for
# low-level hardware control
```

```python
from machine import Pin

# Define GPIO pins to use
# Digital Output
X1 = Pin(16, Pin.OUT)
# Digital Input
btn1 = Pin(18, Pin.IN)

# Activate a digital output
X1.on()
# Deactivate a digital output
X1.off()

# Read digital input
read_val = btn1.value()

# Analogue input (ESP8266)
# Declaring Pin 0 as ADC input
# No other pin is ADC capable
# in the ESP8266
an_pin = ADC(0)

# Analogue input (ESP32)
an_pin = ADC(Pin(36))

# Following line is optional
# ESP32 internal ADC can be
# configured with several
# macros, e.g.:
# Full 3.3V range for ADC conversion
an_pin.atten(ADC.ATTN_11DB)

# Read value (ESP32/8266)
read_an = an_pin.read()
```

Listing A.1: Controlling GPIO in MicroPython.

Using Callback Functions

```python
# main.py

# Using callbacks to functions to
# handle an interrupt caused by a
# button click

from machine import Pin

def callback_funct(p):
  '''
  callback_funct Callback function to
  handle button press interrupt.

  Arguments:
    p {int} -- Pin number wished to print
  '''
  print('Pin', p)

p0 = Pin(0, Pin.IN)
```

```
p0.irq(trigger=Pin.IRQ_FALLING,
        handler=callback_funct)

```

Listing A.2: Using Callback Functions in MicroPython.

Network Connection

```
# main.py

import network

ssid = 'NETWORK_SSID'
password = 'PASSWORD'

station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(ssid, password)

while station.isconnected() == False:
  pass

# Device should be connected,
# but to check, use:

if station.isconnected():
  print('Successful Connection')

  # You can see the IP of your
  # device:
  station.ifconfig()

```

Listing A.3: Managing Network Connections in MicroPython.

Intentionally blank page.

Appendix B

Programming of ESP Devices

This appendix shows code used in encrypting communications done by the ESP devices, ESP32 and ESP8266.

B.1 ESP32

```python
# main.py - ESP32

'''
Main file for uPython edge/sending layer devices.
For use in ESP32.
'''
import gc
import select
import sys
import esp
import machine
import network
import ubinascii
import uhashlib
import ujson
import uos
import urequests
from machine import ADC, Pin
from ucryptolib import aes
from time import sleep

# Machine Unique ID
ID = "c86a301a-d227-4ea1-96a0-40f96de728cf".encode('utf-8')

# Disable on bootloader mode OS debug
# Enable garbage colector (free unused mem.)
esp.osdebug(None)
gc.collect()

# Encryption constants
MODE_CBC = 2
BLOCK_SIZE = 16

# Pin declaration and configuration
LDR = ADC(Pin(36))
LDR.atten(ADC.ATTN_11DB)
```

```
BTN = Pin(16, Pin.IN)
LED = Pin(17, Pin.OUT)

# Network settings
SSID = 'Cabovisao-8A6C'
PASSW = 'e0ca94b08a6c'

STATION = network.WLAN(network.STA_IF)
STATION.active(True)

STATION.ifconfig(('192.168.1.200',
'255.255.255.0',
'192.168.1.1',
'213.228.128.99'))

def j_encode(data):
'''
j_encode Function to encode data in a JSON format.

Arguments:
data {str} -- Data to convert to JSON object.

Returns:
Converted JSON object.
'''
return ujson.dumps(data).encode('utf-8')


def hash_data(data):
'''
hash_data Calculate hash-value of any input string using SHA-256
hash-function.

Arguments:
data {str} -- String from which to calculate the hash-value.

Returns:
Calculate hash-value.
'''
return uhashlib.sha256(data).digest()


def pad_data(data, block_size=BLOCK_SIZE):
'''
pad_data Create padding for plaintext string, making it possible to
use AES block ciphers.

Arguments:
data {str} -- Plaintext data to pad.

Returns:
Padded data, using blank spaces as pads.
'''
pad = BLOCK_SIZE - len(data) % BLOCK_SIZE
padded_data = data + " "*pad
return padded_data


```

```
def encryption(plaintext):
'''
encryption Function to encrypt plaintext, using AES block ciphers -
CBC.

Arguments:
plaintext {str} -- Unecrypted plaintext.

Returns:
Encryption of plaintext.
'''
plaintext = j_encode(plaintext)

key = uhashlib.sha256(ID).digest()

plaintext = pad_data(plaintext)

iv = uos.urandom(BLOCK_SIZE)
cipher = aes(key, MODE_CBC, iv)

encrypted = iv + cipher.encrypt(plaintext)

return encrypted


def check_LDR_vals():
'''
check_LDR_vals Send data to gateway node through a HTTP request.
The LED status is changed upon a certain threshold of the analog
read to simulate further inputs.
'''
analog = LDR.read()

if analog > 2000:
LED.on()
else:
LED.off()

# Data structure to send to Gateway Node.
data = {'sensing_unit': 'S002', # Sensing Unit No.
'LDR': analog,          # LDR read value
'LED': LED.value()}     # Value of debug LED

# Send HTTP request.
r = urequests.post('http://192.168.1.12:5500/esp/vals',
data=(encryption(data)))

# Using limited hardware, connection has to be manually
# terminated.
r.close()


STATION.connect(SSID, PASSW)

while STATION.isconnected() == False:
pass

print('Ready')
```

```python

while True:

if BTN.value():
check_LDR_vals()
else:
LED.off()

sleep(0.1)
```

Listing B.1: Script for Encrypting ESP32 Communications.

B.2 ESP8266

```python
# main.py - ESP8266

'''
Main file for uPython edge/sending layer devices.
For use in ESP8266.
'''
import gc
import select
import sys
import esp
import machine
import network
import ubinascii
import uhashlib
import ujson
import uos
import urequests
from machine import ADC, Pin
from ucryptolib import aes
from time import sleep

# Machine Unique ID
ID = "6042b3be-0e40-4a7f-9454-c2fd2b8dc8ec".encode('utf-8')

# Disable on bootloader mode OS debug
# Enable garbage colector (free unused mem.)
esp.osdebug(None)
gc.collect()

# Encryption constants
MODE_CBC = 2
BLOCK_SIZE = 16

# Pin declaration and configuration
LDR = ADC(0)
BTN = Pin(4, Pin.IN)
LED = Pin(12, Pin.OUT)

# Network settings
SSID = 'Cabovisao-8A6C'
PASSW = 'e0ca94b08a6c'

STATION = network.WLAN(network.STA_IF)
STATION.active(True)

STATION.ifconfig(('192.168.1.201',
'255.255.255.0',
'192.168.1.1',
'213.228.128.99'))


def j_encode(data):
'''
j_encode Function to encode data in a JSON format.

Arguments:
data {str} -- Data to convert to JSON object.

Returns:
Converted JSON object.
'''
return ujson.dumps(data).encode('utf-8')


def hash_data(data):
'''
hash_data Calculate hash-value of any input string using SHA-256
hash-function.

Arguments:
data {str} -- String from which to calculate the hash-value.

Returns:
Calculate hash-value.
'''
return uhashlib.sha256(data).digest()


def pad_data(data, block_size=BLOCK_SIZE):
'''
pad_data Create padding for plaintext string, making it possible to
use AES block ciphers.

Arguments:
data {str} -- Plaintext data to pad.

Returns:
Padded data, using blank spaces as pads.
'''
pad = BLOCK_SIZE - len(data) % BLOCK_SIZE
padded_data = data + " "*pad
return padded_data


def encryption(plaintext):
'''
encryption Function to encrypt plaintext, using AES block ciphers -
CBC.

Arguments:
plaintext {str} -- Unecrypted plaintext.

```

```
Returns:
Encryption of plaintext.
'''
plaintext = j_encode(plaintext)

key = uhashlib.sha256(ID).digest()

plaintext = pad_data(plaintext)

iv = uos.urandom(BLOCK_SIZE)
cipher = aes(key, MODE_CBC, iv)

encrypted = iv + cipher.encrypt(plaintext)

return encrypted


def check_LDR_vals():
'''
check_LDR_vals Send data to gateway node through a HTTP request.
The LED status is changed upon a certain threshold of the analog
read to simulate further inputs.
'''
analog = LDR.read()

if analog > 2000:
LED.on()
else:
LED.off()

# Data structure to send to Gateway Node.
data = {'sensing_unit': 'S003',
'LDR': analog,
'LED': LED.value()}

# Send HTTP request.
r = urequests.post('http://192.168.1.15:5500/esp/vals',
data=(encryption(data)))

# Using limited hardware, connection has to be manually
# terminated.
r.close()


STATION.connect(SSID, PASSW)

while STATION.isconnected() == False:
pass

print('ESP Ready!')

while True:

if not BTN.value():
check_LDR_vals()
else:
LED.off()

```

```
sleep(0.1)
```

Listing B.2: Script for Encrypting ESP8266 Communications.

Intentionally blank page.

Appendix C

Frontend Interface

This appendix contains all the developed frontend interface features not shown previously.

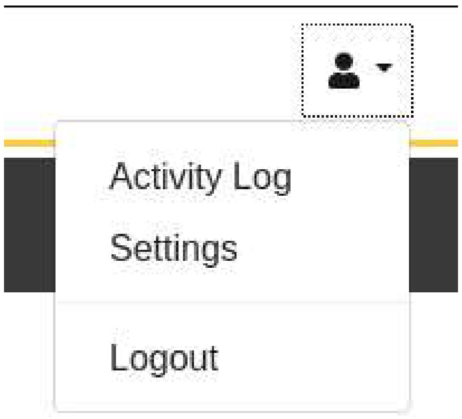
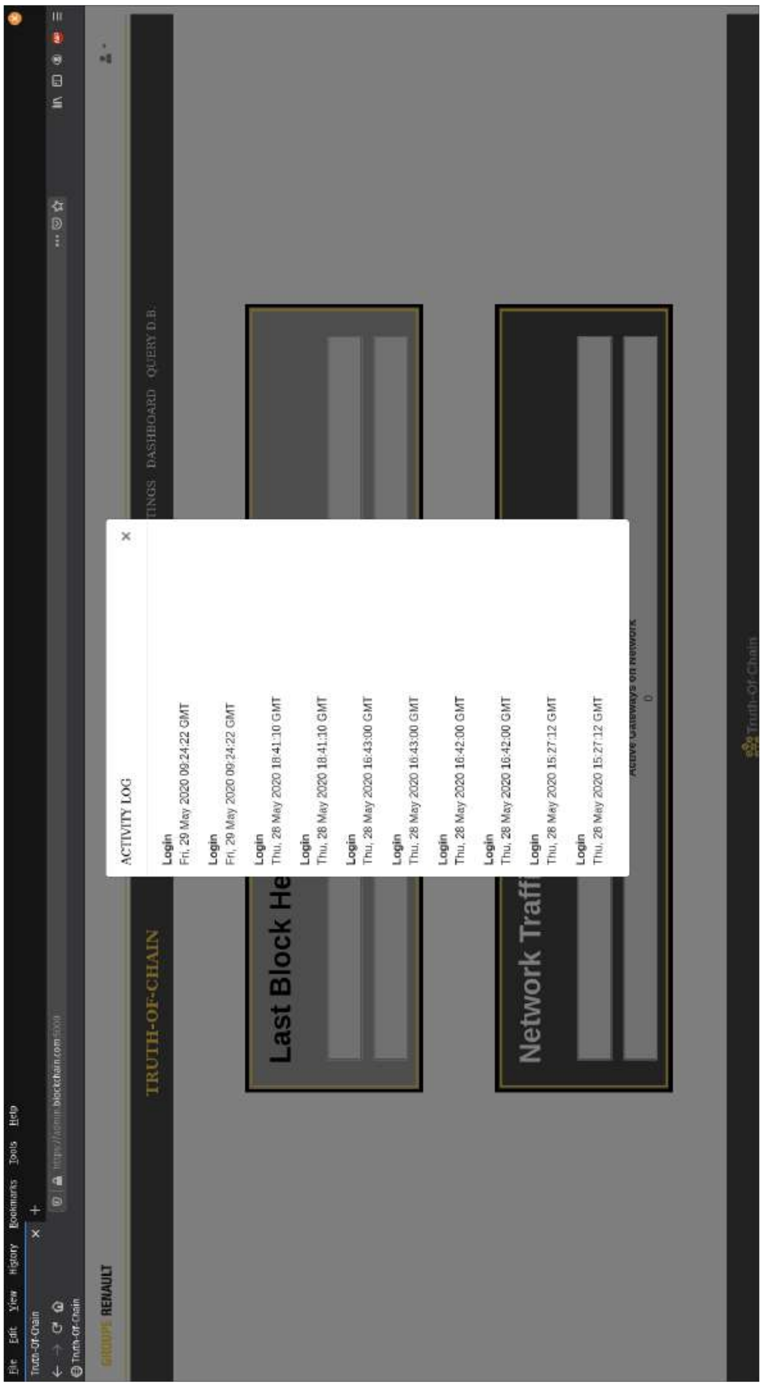


Figure C.1: User Account Settings.



ACTIVITY LOG

Login
Fri, 29 May 2020 09:24:22 GMT

Login
Fri, 29 May 2020 09:24:22 GMT

Login
Thu, 28 May 2020 18:41:10 GMT

Login
Thu, 28 May 2020 18:41:10 GMT

Login
Thu, 28 May 2020 16:43:00 GMT

Login
Thu, 28 May 2020 16:43:00 GMT

Login
Thu, 28 May 2020 16:42:00 GMT

Login
Thu, 28 May 2020 16:42:00 GMT

Login
Thu, 28 May 2020 15:27:12 GMT

Login
Thu, 28 May 2020 15:27:12 GMT

Figure C.2: User Activity Log.

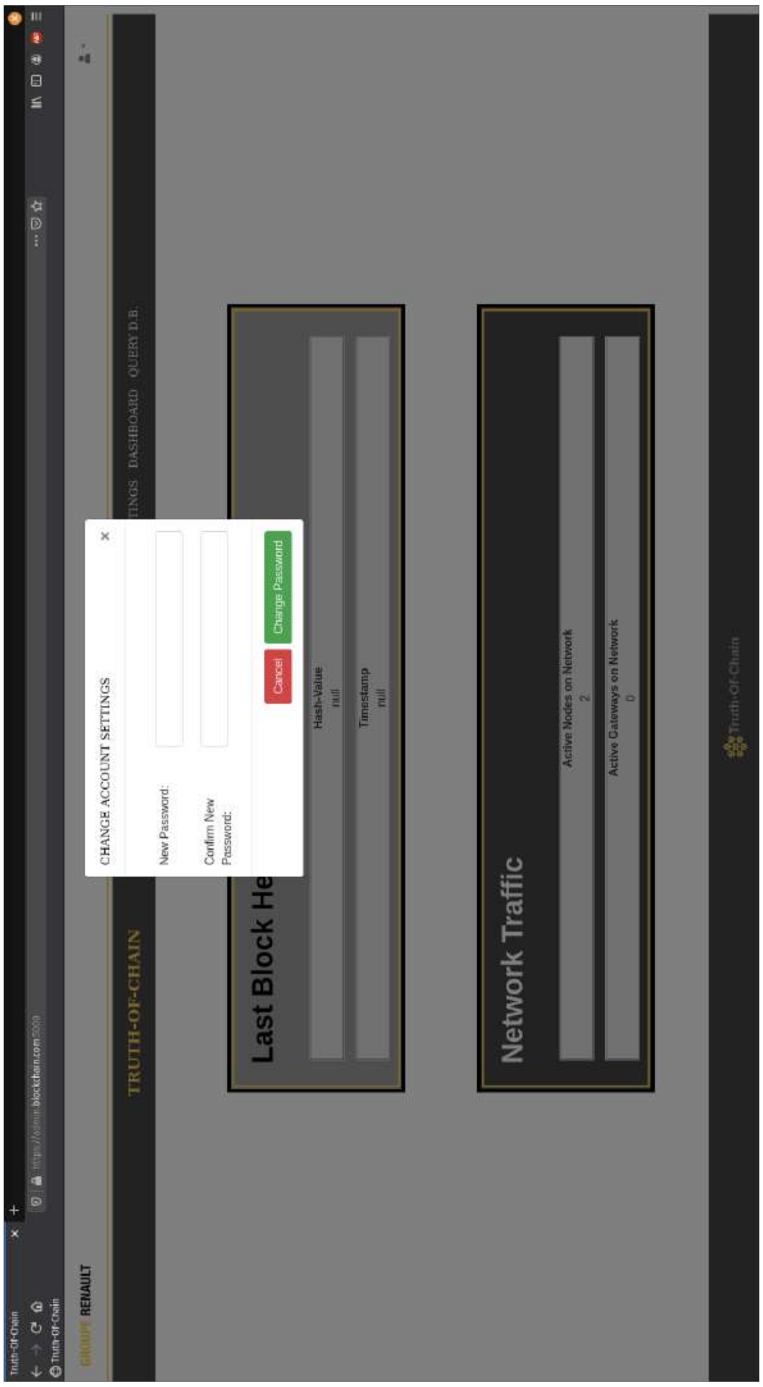
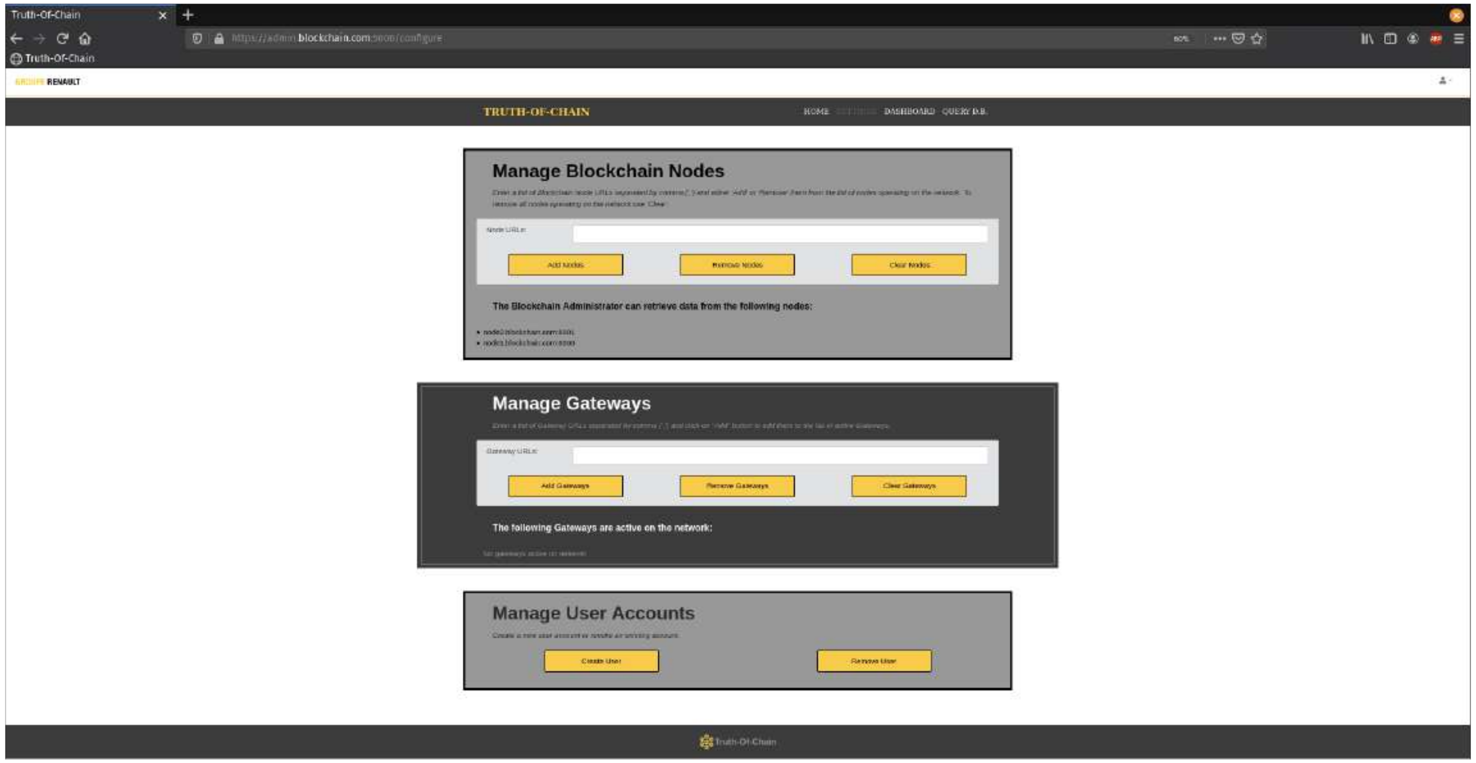


Figure C.3: User Password Change Option.



Manage Blockchain Nodes

Enter a list of Blockchain node URLs separated by comma(,) and either 'Add' or 'Remove' them from the list of nodes operating on the network. To remove all nodes operating on the network use 'Clear'.

The Blockchain Administrator can retrieve data from the following nodes:

- node2.blockchain.com:5001
- node1.blockchain.com:5000

Manage Gateways

Enter a list of Gateway URLs separated by comma (,) and click on "Add" button to add them to the list of active Gateways.

The following Gateways are active on the network:

No gateways active on network

Manage User Accounts

Create a new user account or delete an existing account.

Figure C.4: Blockchain Network Settings.

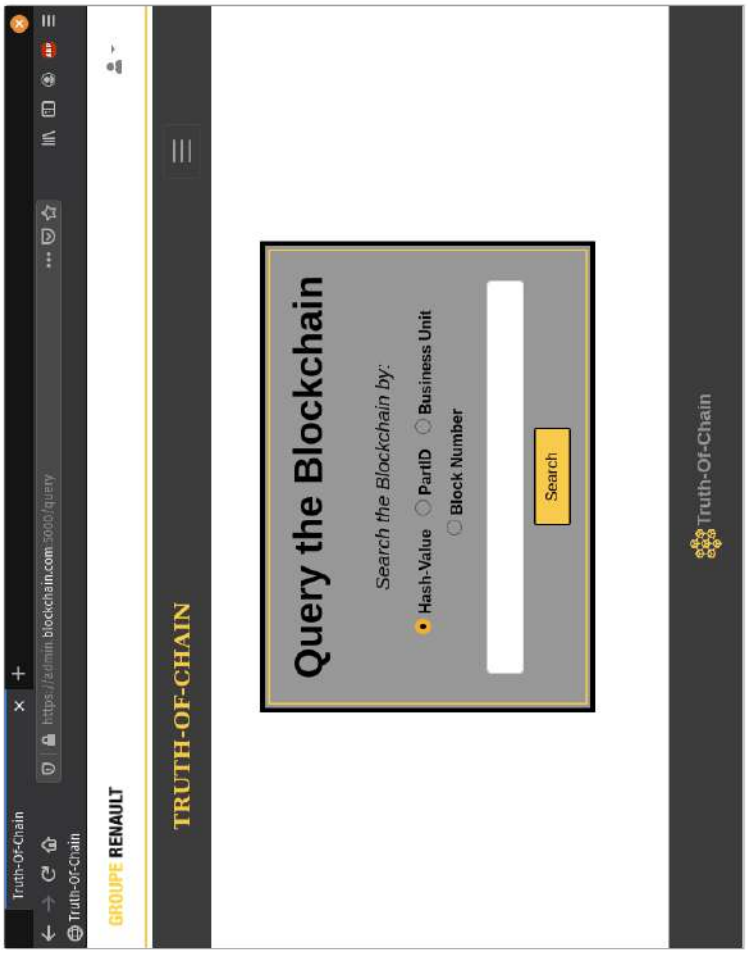


Figure C.5: Query D.B. Function - Open in Internal User/Administrator Account.

Intentionally blank page.

Appendix D

MongoDB

D.1 Installing MongoDB

Download resources and a detailed installation guide are both available at the MongoDB official website[1]. Instructions are provided for all major operating systems, complete with debbuging and troubleshooting tips.

D.2 MongoDB Configuration

The primary daemon process for the MongoDB system is called **mongod**, and handles data requests, data access management, and performs background management operations. To allow control of the configurations used by **mongod** on startup, a configuration file written in YAML Ain't Markup Language (YAML) may be provided. Note that, YAML, a superset of JSON, does not support tab characters for indentation, requiring the use of spaces.

The default configuration file is shown in figure D.1. This file is created during installation of the MongoDB database, but is not used when the **mongod** is initialized, unless specifically requested by the user. For example, the following uses **mongod –config <configuration file>**:

```
$ mongod --config /etc/mongod.conf
```

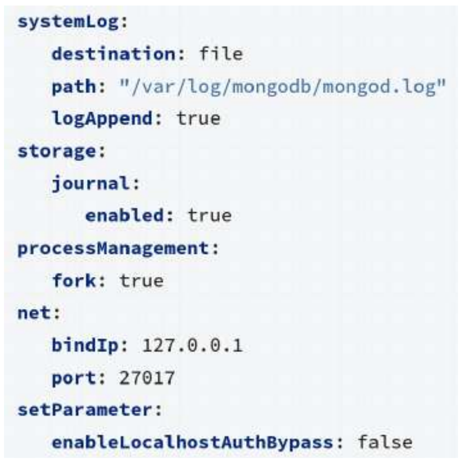
D.3 Interfacing the MongoDB Database

The **mongo** shell is an interactive JavaScript interface to MongoDB, allowing the user to query and update data, as well as perform administrative operations. Launching the **mongo** shell in its default mode can easily be done by typing the following in the command line or terminal:

```
$ mongo
```

This prompt can be further built-upon by adding options when launching the **mongo** shell. For instance, in case of the developed framework, as all connections are TLS encrypted, so does need be the **mongo** session. Additionally, to avoid certificate conflict,

[1]MongoDB Documentation - *Install MongoDB*. Accessed 26 May, 2020. URL: https://docs.mongodb.com/manual/installation/



```
systemLog:
   destination: file
   path: "/var/log/mongodb/mongod.log"
   logAppend: true
storage:
   journal:
      enabled: true
processManagement:
   fork: true
net:
   bindIp: 127.0.0.1
   port: 27017
setParameter:
   enableLocalhostAuthBypass: false
```

Figure D.1: Default MongoDB Configuration File.

a URL was bound to the MongoDB's IP address, which also needs to be provided. Under these circumstances, the following command is required to allow a successful shell connection to the database:

```
$ mongo --tls --tlsCertificateKeyFile /etc/Certificates/client1.pem --
    tlsCAFile /etc/Certificates/ca.pem --host mongodb.blockchain.com
```

D.3.1 Basic MongoDB Shell Commands

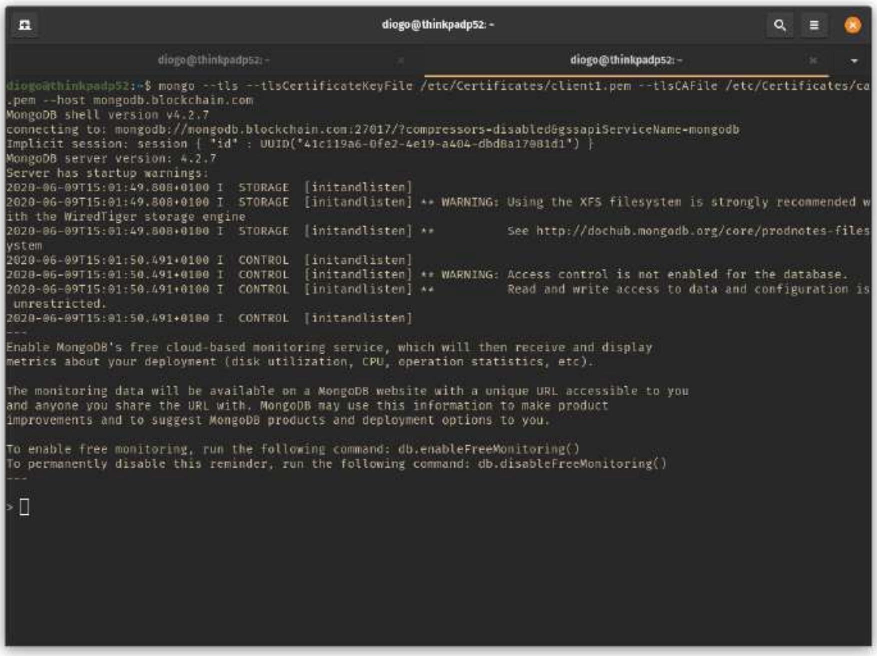
For a correct configuration of the **mongod** instance and deployment of the **mongo** shell, the following result should be obtained, figure D.2. Using the **show dbs** command, all existing databases should be listed. In a clean MongoDB install three databases are present: admin; config; and local. Creating or choosing an existing database is done with the **use** command. For instance **use TestDatabase**, would create a database named TestDatabase. Within databases, collections exist and they can also be viewed using **show collections**.

Figure D.3 shows the creation of a database, **TestDatabase**, and figure D.4 the creation of a collection, **TestCollection**. Some data is inserted in the TestCollection collection, figure D.5, and the output is visualized. It is possible to see that a **ObjectID** is created, as no **_id** field was provided by the user. If this value is given manually, this field is no longer created automatically by MongoDB.

To see what data is present in a collection, the **find()** method is called, figure D.6. Using the second, **pretty()**, command, parses the JSON data in a more human-readable format. Lastly, deleting an entry is shown in figure D.7, with the new **find()** method output returning nothing, as expected.

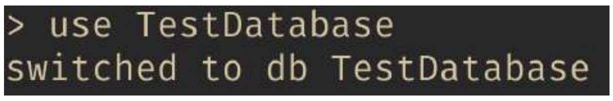
A more detailed explanation into the available commands and their uses is found in the MongoDB reference guide[2].

[2]MongoDB Documentation - *Guides - Getting Started*. Accessed 26 May, 2020. URL: `https://docs.mongodb.com/guides/`



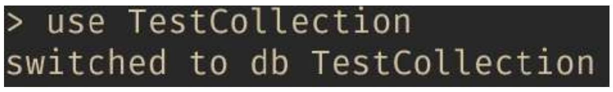
```
diogo@thinkpadp52:~$ mongo --tls --tlsCertificateKeyFile /etc/Certificates/client1.pem --tlsCAFile /etc/Certificates/ca
.pem --host mongodb.blockchain.com
MongoDB shell version v4.2.7
connecting to: mongodb://mongodb.blockchain.com:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("41c119a6-0fe2-4e19-a404-dbd8a17081d1") }
MongoDB server version: 4.2.7
Server has startup warnings:
2020-06-09T15:01:49.808+0100 I  STORAGE  [initandlisten]
2020-06-09T15:01:49.808+0100 I  STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended w
ith the WiredTiger storage engine
2020-06-09T15:01:49.808+0100 I  STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-files
ystem
2020-06-09T15:01:50.491+0100 I  CONTROL  [initandlisten]
2020-06-09T15:01:50.491+0100 I  CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2020-06-09T15:01:50.491+0100 I  CONTROL  [initandlisten] **          Read and write access to data and configuration is
 unrestricted.
2020-06-09T15:01:50.491+0100 I  CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

Figure D.2: MongoDB Shell.



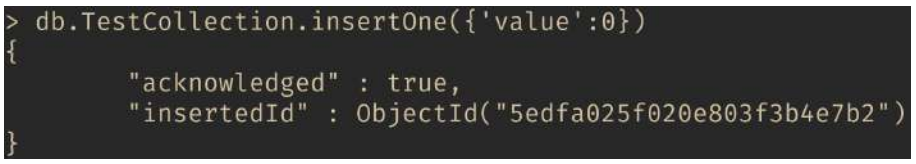
```
> use TestDatabase
switched to db TestDatabase
```

Figure D.3: Creating a Database.



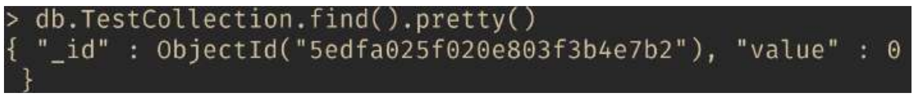
```
> use TestCollection
switched to db TestCollection
```

Figure D.4: Creating a Collection.



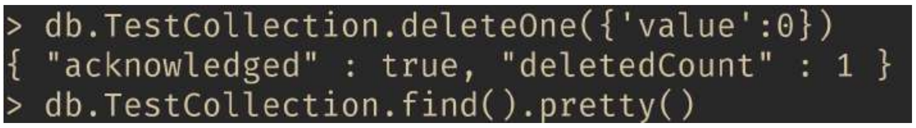
```
> db.TestCollection.insertOne({'value':0})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5edfa025f020e803f3b4e7b2")
}
```

Figure D.5: Inserting Values in Created Collection.



```
> db.TestCollection.find().pretty()
{ "_id" : ObjectId("5edfa025f020e803f3b4e7b2"), "value" : 0
 }
```

Figure D.6: Querying for Values in Collection.



```
> db.TestCollection.deleteOne({'value':0})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.TestCollection.find().pretty()
```

Figure D.7: Deleting Values in Collection.

Appendix E

Proposed Framework Step-by-Step Deployment Guide

E.1 Step 1 - Software Installation

Firstly it is necessary to retrieve all required software for the framework's deployment, including: MongoDB database (as per appendix D instructions); and Python, version 3.7.x or higher, along with the correct version of the **pip** installer. Python can be downloaded for all major operating systems directly from the Python Foundation's website[1], but note that in the case of a Linux-based OS, most distributions come with Python pre-installed.

Next are the requirements for use with Python itself, with necessary modules being installed using the **pip** command. A **requirements.txt** file exists within the Truth-Of-Chain source code folder to aid in the process, and the following command in a UNIX Bash terminal or Windows command line is sufficient:

```
$ pip install -r requirements.txt
```

Note that it is first recommend to check the installed **pip** version using **pip -V**. If an error occurs, it is likely that **pip3** is installed, which would lead to the command used being:

```
$ pip3 install -r requirements.txt
```

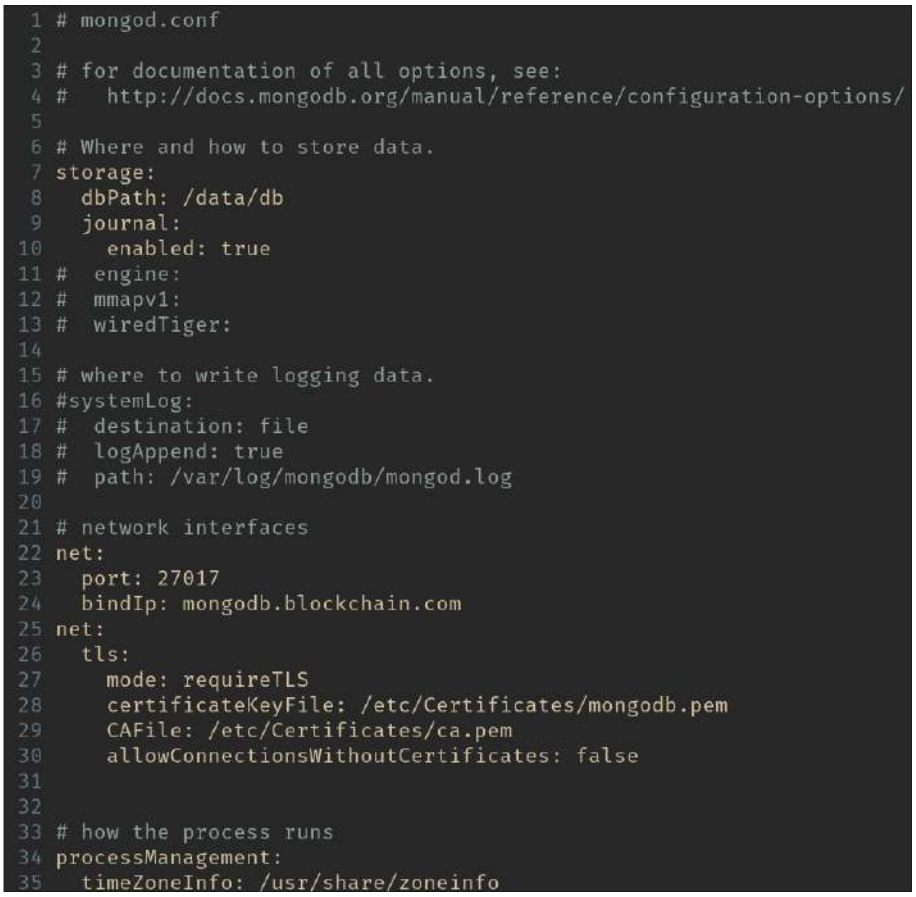
E.2 Step 2 - Hosts Table Modification

This step depends on whether or not the platform is being used for a simulation or for commercial deployment. In case it is used for simulation, fictitious addresses must be provided to the **hosts** table found in any major operating system. Otherwise, this step can either be skipped or the real-life external IP address of the several network elements (i.e., Blockchain Administrator, Mining Nodes, Database, and Gateways) can be bound to a different common name. For instance, instead of using a public IP such as **217.129.220.139**, bind this IP to **admin.blockchain.com**.

[1]Python - *Download the Latest Source Release.* Accessed 26 May, 2020. URL: `https://www.python.org/downloads/`

E.3 Step 3 - Configure MongoDB

MongoDB does not, by default, use TLS encrypted connections. This must be addressed through the edit of the configuration file, such as is shown in figure E.1. As seen, parameters referring to the **mongod.log** file are commented out of use, as sometimes the location of this file is problematic and might hinder the database's deployment.



```
# mongod.conf

# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: /data/db
  journal:
    enabled: true
#  engine:
#  mmapv1:
#  wiredTiger:

# where to write logging data.
#systemLog:
#  destination: file
#  logAppend: true
#  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: mongodb.blockchain.com
net:
  tls:
    mode: requireTLS
    certificateKeyFile: /etc/Certificates/mongodb.pem
    CAFile: /etc/Certificates/ca.pem
    allowConnectionsWithoutCertificates: false


# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo
```

Figure E.1: Proposed MongoDB Configuration File.

The value inserted in the **bindIP** parameter is merely suggestive, with the actual value depending on values attributed on the **hosts** table or the public IP address of the server running the MongoDB. The **mongod** daemon is now ready to be initiated, as demonstrated in appendix D.

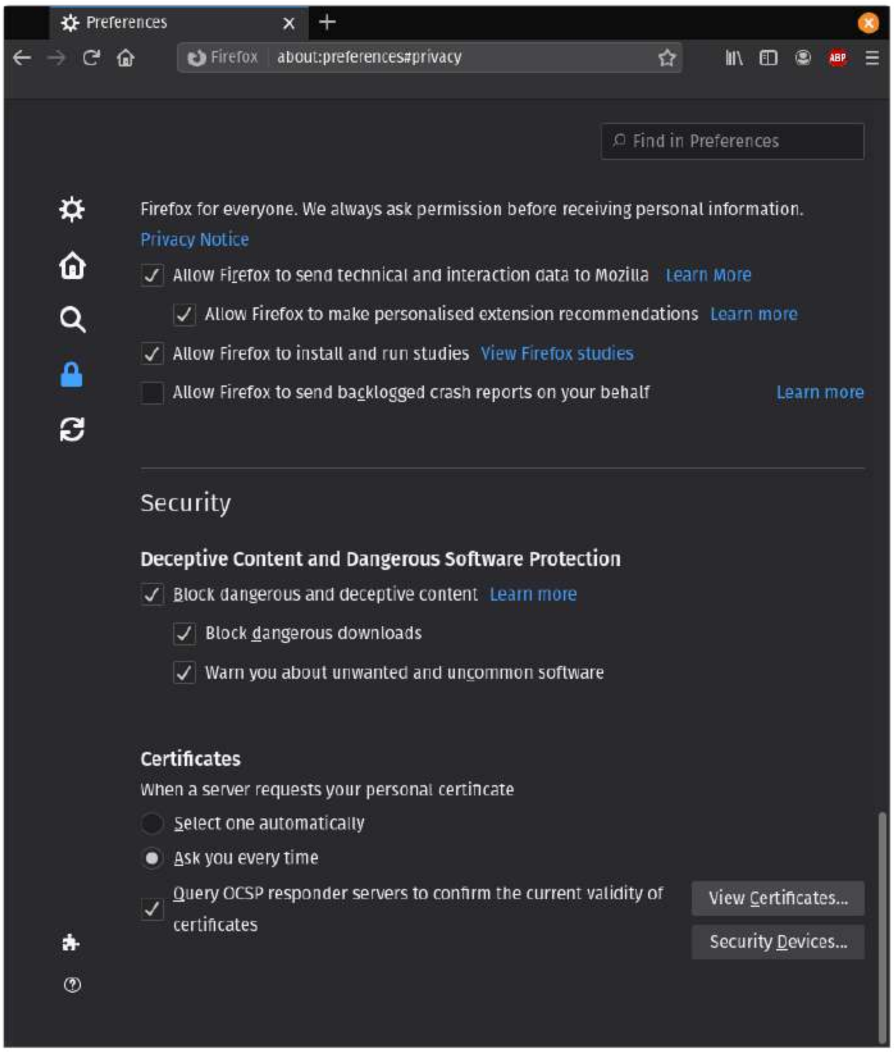
E.4 Step 4 - Configure the Remaining Network Elements

Three more configuration files - **config.json** - need to be edited for each one of the remaining network elements: Blockchain Administrator, Mining Nodes and Gateways. After this step, within each corresponding source code folder, the Python scripts can be run using the command: **python3 <file name>**.

Attention: file paths under Windows follow different rules from Linux-based operating systems; and as such this must be corrected.

E.5 Step 5 - Accessing the Frontend Interface

After Step 5, access to the frontend interface is possible. To achieve this, it is first necessary to provide the appropriate TLS certificates to the browser, otherwise connection will be refused by the Blockchain Administrator. As an example, Mozilla's Firefox Browser 77.0.1 (64 b-bit) is used. Go to **Preferences** -> **Security** -> **Certificates** and choose the **View Certificates** option, figure E.2. The **Certificate Manager** will pop-up and now the user's public key can be imported in the **Your Certificates** tab, figure E.3, and a new authority must be added in the **Authorities** tab, figure E.4. This is the public key corresponding to the custom CA created.



Firefox for everyone. We always ask permission before receiving personal information.
Privacy Notice

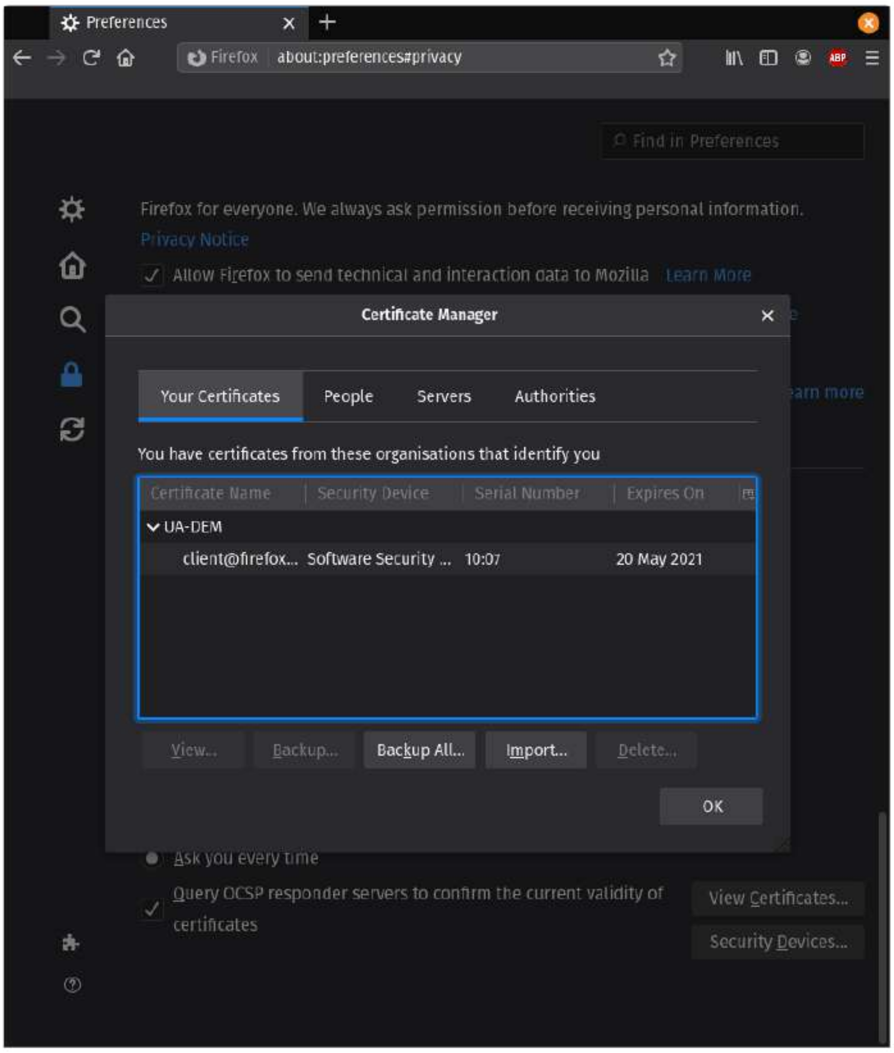
✓ Allow Firefox to send technical and interaction data to Mozilla Learn More
✓ Allow Firefox to make personalised extension recommendations Learn more
✓ Allow Firefox to install and run studies View Firefox studies
Allow Firefox to send backlogged crash reports on your behalf Learn more

Security

Deceptive Content and Dangerous Software Protection
✓ Block dangerous and deceptive content Learn more
✓ Block dangerous downloads
✓ Warn you about unwanted and uncommon software

Certificates
When a server requests your personal certificate
Select one automatically
[✓ Ask you every time]
✓ Query OCSP responder servers to confirm the current validity of certificates

Figure E.2: Security Preferences in Firefox Browser.



Firefox for everyone. We always ask permission before receiving personal information.

✓ Allow Firefox to send technical and interaction data to Mozilla

You have certificates from these organisations that identify you

Certificate Name	Security Device	Serial Number	Expires On
UA-DEM			
client@firefox...	Software Security ...	10:07	20 May 2021

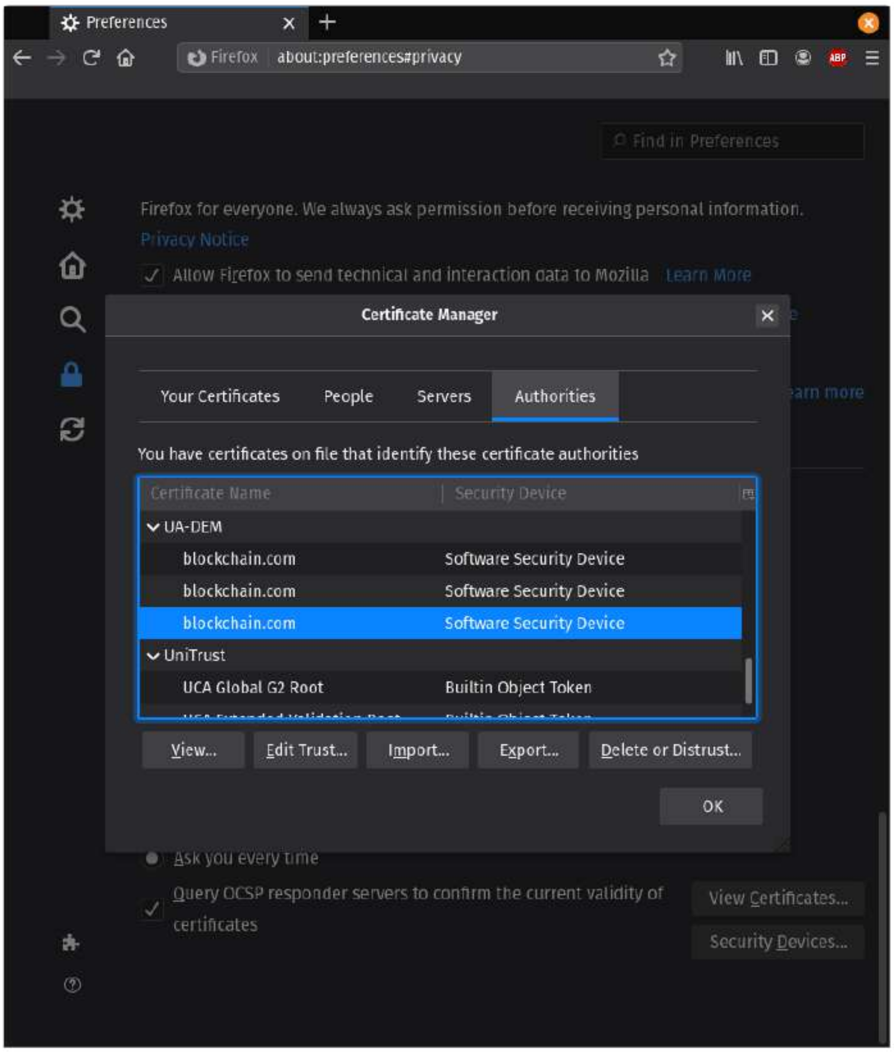
● Ask you every time

✓ Query OCSP responder servers to confirm the current validity of certificates

Figure E.3: Adding a Personal Certificate in Firefox Browser.

E.6 Step 6 - Configuring the Network in the Frontend Interface

The last step is done in the fronted interface, with the network elements deployed in Step 4 being added to the blockchain system in the **Settings** page. Note that access for this procedure is reserved to an administrator account.


Preferences

Firefox about:preferences#privacy

Find in Preferences

Firefox for everyone. We always ask permission before receiving personal information.
Privacy Notice

✓ Allow Firefox to send technical and interaction data to Mozilla Learn More

Certificate Manager

Your Certificates | People | Servers | Authorities

You have certificates on file that identify these certificate authorities

Certificate Name	Security Device
UA-DEM	
blockchain.com	Software Security Device
blockchain.com	Software Security Device
blockchain.com	Software Security Device
UniTrust	
UCA Global G2 Root	Builtin Object Token

View... | Edit Trust... | Import... | Export... | Delete or Distrust...

OK

● Ask you every time

✓ Query OCSP responder servers to confirm the current validity of certificates

View Certificates...

Security Devices...

Figure E.4: Adding a Certificate Authority in Firefox Browser.