

UNIVERSITY OF SALENTO



Department of Engineering for Innovation

Master of Science Degree in Computer Engineering

Degree Thesis in Software Engineering

**Performance Analysis of the Algorand
Technology on low-power devices
based on STM32 MPU**

in collaboration with STMicroelectronics, IDALab
and Algorand Foundation

Supervisors

Prof. Luigi Patrono

Prof. Luca Mainetti

Student

Salvatore Corvaglia

Co-supervisors

Eng. Antonio Vilei

Eng. Teodoro Montanaro

Academic year 2019/2020

...to myself

Abstract

Through the use of sensors and other edge devices and infrastructure, IoT is transforming people lives also revolutionizing the way enterprises operate. Although, one of the major challenges revealed in the exploitation of IoT functionalities regards the need of protecting collected information at all levels of IoT ecosystems. Unfortunately, also due to the growing number of connected devices, data security has become extremely complex. One of the revolutionary technologies that have the potential to address some of the IoT security, privacy and scalability challenges is the Blockchain. Therefore, this thesis discusses the integration of IoT systems and the Blockchain technologies with a specific focus on low-power devices enhanced through the functionalities of Algorand, a self-sustaining, decentralized, Blockchain-based network which has all the potential to shape the future of the Blockchain context, offering decentralization, scalability and security.

Contents

1	Introduction	1
2	Background	5
2.1	DLT and Blockchain	5
2.1.1	Distributed Ledger Technology (DLT)	6
2.1.2	Blockchain	7
2.1.3	The Byzantine Generals Problem	10
2.2	Blockchain Trilemma	11
2.3	Algorand	13
2.4	IoT	34
2.4.1	Low-power Devices	36
2.4.2	Application Performance Monitoring	41
3	State of the art	43
3.1	Blockchain and IoT	43
3.2	Blockchain on IoT low-power devices	45
3.3	Algorand Blockchain applied to IoT	50
4	Requirements Analysis	52
4.1	Preliminary analysis	52
4.2	Requirements definition	57
5	Implementation	59
5.1	Raspberry Pi 4 Model B	59
5.2	STM32MP157A-DK1	63

5.2.1	Preliminary Operations	63
5.3	Interaction with CLI	67
5.4	Interaction with Node	75
6	Results	79
6.1	Block Explorer	79
6.2	Performance analysis	83
7	Conclusion and future works	91

Chapter 1

Introduction

For years now, our way of interacting, working, studying, managing our time and our hobbies has changed thanks to advent of the Internet which has fostered an innovative future that has already become reality. Thanks to its spread, in fact, information become more accessible and usable, communications got faster and simpler, and technological innovations have been introduced to optimize the agendas and business processes.

Different sectors have already benefited from the introduction of Internet in daily lives and, specifically, thanks to the latest technological progress, also the industrial and electronic sectors have been improved leading to a model shift in manufacturing efficiency and operational cost reduction. Generally, this cost reduction comes at the price of dismissing information security, especially when multiple stakeholders are involved.

The promise of a strongly connected world, where previously inactive devices become “smart”, has driven a boom in the uptake in IoT technology over the past years.

IoT involves the extension of internet connectivity beyond personal computers and mobile devices and can reach a wide range of non-internet enabled devices. Once the devices have been embedded with technology, they are brought to life and can communicate with each other, this means they can be monitored and controlled remotely [7].

The Internet of Things (IoT) paradigm is envisaged as one of the most disrupt-

tive concepts in the near future, with large technological, societal and economic impacts. The term Internet of Things describes the large and growing set of digital devices that operate across networks of potentially global scale.

Massive IoT applications involve a large number of low-cost devices, portable/wearable, and they often transmit small data volumes and have several energy consumption requirements. Some of these applications are smart health, smart metering, transportation fleet management, smart buildings, monitoring of large infrastructures or industrial plants, agriculture or natural ecosystems.

The ever-increasing number of IoT devices requires a secure and scalable infrastructure to store and process generated data. In recent years, the Distributed Ledgers Technologies (DLT) and related tools such as Blockchain, have gaining momentum as solutions in such scenario.

In fact, their decentralized and cryptographically secured nature offer new means of securing networks, store data permanently, increase transparency, reduce fraud and abuses of privacy which become increasingly commons. Different companies and researchers is in fact investing time and effort in the exploitation of such technologies in their daily activities.

Although, the spread and the consequent increasing advances in DLTs a also raising the knowledge base needed by new companies looking at this market to benefit from the integration of those technologies. For instance, the growing number of available DLTs (e.g., IOTA, VeChain, IoTeX and MXC) open up to new challenges that were not present in the early birth of these technologies.

Low power consumption has become an important design issue in many electronic microsystems and low-power IoT technologies provide a low-cost, sustainable alternative to conventional Machine to Machine (M2M) connections, allowing devices to operate for several years.

For many remote devices, low-power is essential for preserving battery life, and considering that the number of devices is expected to rise exponentially, the environmental impact also needs to be considered.

Low-power IoT end-devices do not possess enough resources to run a software client for intensive calculations, so the purpose of this work is to investigate on

existing possibilities to enable low-power IoT devices in accessing a Blockchain-based infrastructure that, instead, is usually an high energy consuming technology.

For the purposes of this thesis the Algorand Blockchain [10] has been selected for all the performed activities due to its inclination towards the IoT domain. In fact, although Algorand propounds a bright horizon for financial purposes and lacks several features to be successfully applied to IoT networks, its characteristics are appealing for IoT.

Indeed, its self-sustaining and decentralized based network already supports a wide range of applications. In addition, it does not require communication among users to determine if they are selected to participate in the consensus protocol and it does not postulate computational resources to solve cryptographic puzzles. These characteristics alongside with decentralization, high scalability and security are attractive features for IoT.

Within the described context, this thesis aims at investigating the applicability of the Algorand to the IoT domain and specifically demonstrate the benefits of its application to low-power devices. Different experiments have been conducted: some low-power devices were configured to operate as Algorand nodes and some transactions have been added to the distributed ledger by implementing all the requested cryptographic functions, communication functions and specific operations.

Finally, a performance analysis among all the tested low-power devices was carried out to understand on which device Algorand performs better and, finally, some possible future works are evaluated within the present work.

The thesis is divided into seven chapters: after the present *introduction chapter* that guides readers through the problem formulation, in *second chapter* we present all the background knowledge that guided the work performed within this thesis. At first, we present all the information related to the DLT and Blockchain context, their founding principles (e.g., the Trilemma concept and the most used consensus protocols) and the main characterizing of the Algorand ecosystem and its consensus protocol called Pure Proof-of-Stake.

Finally, the chapter presents an overview on IoT and the low-power devices exploited within this work.

In *third chapter* we investigate the main research interests in the exploitation of IoT and Blockchain with a specific focus on main results from a theoretical, experimental and performance point of view.

In *chapter four* we define the main requirements of the present work and the consequent details of the activities to be performed to achieve all the aims of the present thesis.

In *chapter five* we describe all the performed experiments and the corresponding implemented artifacts.

In *chapter six* we present the results obtained from the carried experiments and discuss the main outcome (in terms of performance evaluation) of the work.

Finally, *chapter seven* draws the conclusions and proposes new possible future works.

This work was carried out in collaboration with STMicroelectronics, global semiconductor leader delivering intelligent and energy-efficient products and solutions.

Chapter 2

Background

In this chapter we describe at first the concepts related to the DLTs and Blockchains, in particular we explain why the scalability, decentralization and security properties represent the core features of this technology.

Later, we describe how consensus protocols are used in Distributed Ledger Technology to solve the algorithms related to the validation and recording of transactions, precisely two of the most famous consensus protocols in the Blockchain technology sector will be presented: Proof-of-Work (PoW) and Proof-of-Stake (PoS).

We present some functional concepts, the main characteristics of the Algorand ecosystem and its consensus protocol called Pure Proof-of-Stake, comparing it with the most spread alternative protocols (i.e., the PoW and PoS already mentioned, but also others).

Finally, the last part is dedicated to the low power devices used for this thesis work.

2.1 DLT and Blockchain

One of the hottest topic of the last decade is for sure the Blockchain one. A growing audience, in fact, is increasingly investigating the possibilities that such technology can bring in their own business. For instance, traditional centralized bodies like banks and governments are starting to take interest into

what this technology can do for them.

However, only a few of this stakeholders know its dependency from the Distributed Ledger Technology (DLT), a term that is, instead, most frequently used in the cryptocurrency domain.

Distributed Ledger Technology (DLT) and Blockchain share a conceptual origin: they are digitalized and decentralized log books of record and often the terms are confused, but they are differentiated by an unshared set of specific features: all Blockchains are DLTs, but not all DLTs are Blockchains [2].

2.1.1 Distributed Ledger Technology (DLT)

Despite confusing acronyms such as DLT in financial and Fintech domain, this technology is relatively easy to understand. A distributed ledger is a database that exists across several locations or among multiple participants.

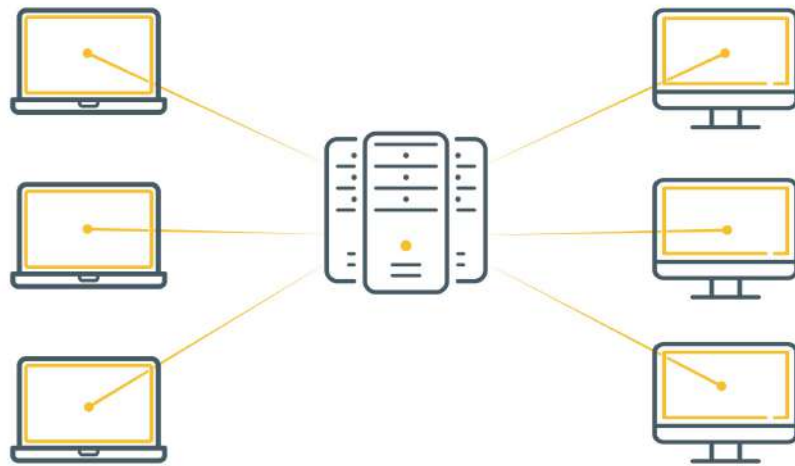


Figure 2.1: Centralized Database

It is the direct competitor of the centralized approach in which, as shown in Figure 2.1, companies leverages on central databases stored in a fixed location to store information.

While in a centralized database there is a single point of failure, a distributed

ledger is distributed on multiple stakeholders to eliminate the need for a central authority or intermediary to process, validate or authenticate transactions. Enterprises use distributed ledger technologies to process, validate, authenticate transactions or other types of data exchanges and typically, these records are only ever stored in the ledger when the consensus has been reached by the parties involved [4].

There are different types of Distributed Ledger Technologies [12] and, as shown in Figure 2.2, the three most popular are: Blockchain, Directed Acyclic Graph (DAG) and Hashgraph.

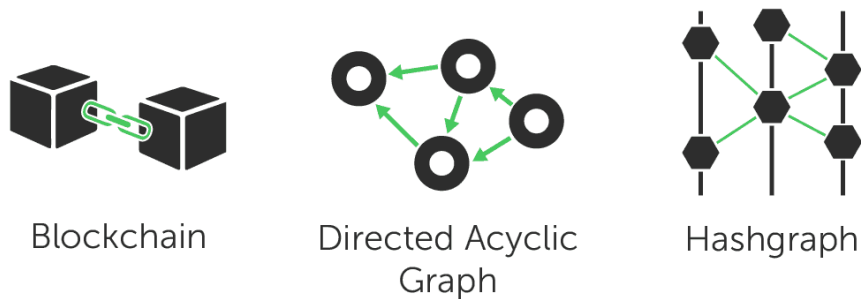


Figure 2.2: Types of DLT

2.1.2 Blockchain

Blockchain technology is a decentralized database that stores a registry of assets and transactions across a peer-to-peer network. The transactions are secured through cryptography, and over time, transaction history is locked in blocks of data that are then cryptographically linked together and secured.

This configuration creates an immutable, unforgettable record of all of the transactions across this network and is replicated on every computer that uses the network.

Blockchain can be regarded as a quality leap from the distributed database technology [13] studied since the seventies. Generally, Distributed Ledger Technologies (DLTs) are designed to deal with database in the form of data shared in a distributed way and Blockchain represents one possible DLT to do it, as

shown in Figure 2.3.



Figure 2.3: DLT evolution

What mostly differentiates the Blockchain unique from other DLTs is the grouping and organization into blocks, in fact the blocks are connected to each other and protected by encryption [5].

The basic components of the Blockchain are:

- **Node**: it is a device on a Blockchain network that enables its running and can be any active electronic device that is connected to the Internet and has an IP address.
- **Transaction**: it consists of the data that represents the values being exchanged and that need to be verified, approved and then archived.
- **Block**: it represents the group of a set of transactions that are merged to be verified, approved and then archived by the participants in the Blockchain.
- **Consensus algorithm**: it is a procedure through which all the peers of the Blockchain network reach a common agreement about the present state of the distributed ledger. Essentially, the consensus protocol makes sure that every new block that is added to the Blockchain is the one and only one version of the truth that is agreed upon by all the nodes in the Blockchain.
- **Ledger**: it is the public ledger in which all transactions carried out in an orderly and sequential manner are recorded with the utmost transparency and immutable. The Ledger consists of the set of blocks that are chained together through an encryption function and thanks to the use of hash.
- **Hash**: it is an operation that allows to map a text and/or numeric string of variable length into a unique string of determined length.

Each block therefore contains several transactions and has a Hash placed in the header. The Hash records all the information related to the block and a Hash that contains the information related to the previous block and actually

creates the chain and links one block to another.

The transaction, instead, contains information related to the public address of the recipient, the characteristics of the transaction and the cryptographic signature that guarantees the security and authenticity of the transaction.

There are three main types of Blockchains, namely **Public**, **Private** and **Consortium (or Federated)** [3].

- A **Public Blockchain** is permissionless, anyone can join the network and read, write, or participate within the Blockchain. A Public Blockchain is decentralized and does not have a single entity which controls the network. Data on a public Blockchain are secure as it is not possible to modify or alter data once they have been validated on the Blockchain.

- A **Private Blockchain** is a permissioned Blockchain and work based on access controls which restrict the people who can participate in the network. There are one or more entities which control the network and this leads to reliance on third-parties to transact. Only the entities participating in a transaction will have knowledge about it, whereas the others will not be able to access it.

- A **Consortium Blockchain** tries to remove the disadvantages of the Private Blockchain where one entity can get all the rights. There are more entities in charge instead of a single in charge and here a group of representatives or individuals or companies come together and take all the decisions for the overall benefits of the Blockchain network.

Figure 2.4 provides a comparison between Public, Private and Consortium Blockchain.

Parameter	Public Blockchain	Private Blockchain	Consortium Blockchain
Read permission	Public	May be public or restricted	May be public or restricted
Immutability	Tampering is difficult	Can be tampered	Can be tampered
Efficiency	Low latency	High latency	High latency
Centralization	Decentralized	Centralized	Partly or fully centralized
Consensus process	Permissionless	Permissioned	Permissioned
Asset	Native Asset	Any Asset	Any Asset

Figure 2.4: Difference between Public, Private and Consortium Blockchain

2.1.3 The Byzantine Generals Problem

One of the main problems on which the Blockchain is founded is the Byzantine Generals Problem. It is a computer-related problem consisting in finding an agreement by communicating through messages between the different components of the network.

Conceived in the 1980s, it describes a dilemma in which isolated participants must communicate to coordinate their actions. The specific dilemma involves a handful of army generals that surround a city, deciding whether to attack it [24]

Each general must decide whether to attack or retreat and it does not matter whether they attack or retreat, as long as all generals agree on a common decision. If they decide to attack, they will only be successful if they move in at the same time, so how do they ensure that they can pull this off?

Sure, they could communicate through messenger, but what happens if the messenger is intercepted with a message that says “we’re attacking at dawn,” and that message is replaced with “we’re attacking tonight”? What happens if one of the generals is malicious and intentionally misleads the others to ensure they’re defeated?

We need a strategy where a consensus can be reached, even if participants be-

come malicious or messages get intercepted. The maintenance of a database is not a life-and-death situation like attacking a city without reinforcements, but the same principle can be applied. If there's no one to oversee the Blockchain and to give "correct" information to users, then the users must be able to communicate amongst themselves.

To overcome the potential failure of one or several users, the mechanisms of the Blockchain must be carefully engineered to be resistant to such setbacks and a system that can achieve this is referred to as Byzantine fault-tolerant.

Byzantine Generals Problem: Formal Definition

Given a system of n components, t of which are dishonest, and assuming only point-to-point channel between all the components.

Whenever a component A tries to broadcast a value x , the other components are allowed to discuss with each other and verify the consistency of A 's broadcast, and eventually settle on a common value y .

The system is said to resist Byzantine faults if a component A can broadcast a value x , and then:

1. If A is honest, then all honest components agree on the value x .
2. In any case, all honest components agree on the same value y .

2.2 Blockchain Trilemma

The Blockchain Trilemma is a concept coined by Vitalik Buterin, founder of Ethereum, an open-source Blockchain-based decentralized software platform launched in 2015.

It refers to the trade-offs that Blockchain projects must make when deciding how to optimize their architecture, by balancing between three of the defining features of the technology: decentralization, security, and scalability.

The Trilemma claims that Blockchain systems can only maximize two of these features at the expense of the third, therefore, all Blockchain systems require trade-offs depending on the specific application and use case of the system [8].

The best way to process the Trilemma difficulty is by analyzing each aspect independently. So the following list examines scalability, decentralization, and security that are also shown in Figure 2.5.

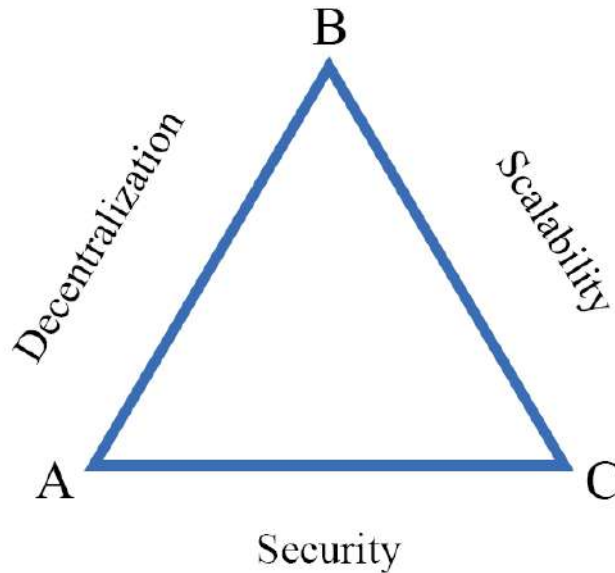


Figure 2.5: Blockchain Trilemma

A) Decentralization

Decentralization, as the word suggests, refers to the degree of diversification in ownership, influence and value in the Blockchain.

Decentralized systems matter because they empower permissionless ownership where anyone can use and build on the platform. Decisions are made by consensus, which means transactions are approved by a group of nodes as opposed to an individual node.

Once these transactions are verified by consensus, they can't be altered after the fact. Therefore, risk isn't placed in one central entity, and trust doesn't rely on another individual when conducting a transaction.

B) Scalability

Scalability is the capacity of a system to dynamically increase the capabilities of a system or a network to address the increasing participation within the sys-

tem or network. In virtual terms, scalability is the need of adding resources, additional with respect to the ones already exploited, to accommodate the increased load that is brought by the growth of the requests.

Therefore, a network is scalable if it can accommodate the increasing data flow by adding an corresponding amount of required resources.

C) Security

Digital security is one of the greatest drawbacks of the Internet since its inception.

When it comes to distributed security, there arises an even greater problem of securing information as is supposed to be public and transparent, thus, there exists a need to carefully evaluate each strand of data that is published on the distributed network.

Distributed systems constitute an innovative solution for maintaining security and, consequently, provide immutability to information that is computed on the distributed database.

2.3 Algorand

As depicted in the previous section, the Blockchain Trilemma poses an important question to the research community: how can we scale fast, be decentralized and do not lose any of the security properties promised by the Blockchain? One of the technologies that declare to solve it is Algorand [11]. In fact, it is able to solve the Trilemma Problem by assuming that it does not exist and that there are not acceptable choice combinations. Indeed, by choosing security and scalability implies giving also up decentralization.

In addition, without scalability, we would have to give up the use of technology on a large scale. Finally, nowadays it is not absolutely possible to give up security.

Algorand was founded by **Silvio Micali**, MIT professor and Turing Prize winner in 2017. It was created as a fully decentralized, secure, and scalable Blockchain which provides a common platform for building products and ser-

vices for a borderless economy.

This innovative Blockchain uses a decentralized Byzantine Agreement protocol that leverages on a Pure Proof-of-Stake (Pure POS) consensus algorithm [18]. One of the most important concepts to be treated in describing a specific Blockchain regards the used consensus protocols due to its importance in the distributed systems scenario.

Two of the most famous consensus protocols in the Blockchain technology sector are the Proof-of-Work (PoW) and the Proof-of-Stake (PoS) consensus protocols. They will be described and detailed in the following subsections to then guide the readers to the description of the Pure Proof-of-Stake protocol used by the Algorand Blockchain [25].

It is at the base of each user's influence on the choice of a new block and gives each user an influence that is proportional to its stake (number of token) in the system.

Proof-of-Work The original concept of Proof-of-Work dates back to 1993, when it was developed to prevent Denial-of-Service (DoS) attacks and other breaches in computer systems such as network spam.

Every time a new transaction is generated, it is delivered to all nodes who gather these data to be afterward inserted into the next block. Transactions are ordered and pairwise hashed, forming a Merkle tree, so that only the root needs to be stored [29]. In this way a transaction verification is performed quickly by checking the consistency of the internal branch's hashes. In order to build the next block, each node must solve a cryptographic puzzle.

The first who finds a solution to this complicated puzzle becomes the leader in charge of closing and deliver the block. Therefore, the faster the puzzle is solved, the greater the chance of generating the next block. This process is called mining and it is performed by the so called miners.

As long as the majority of the honest nodes are encouraged to find a solution to the puzzle, the majority of the CPU power would be in honest hands. Hence, increasing the number of miners, decreases the chance for malicious nodes to

control the 51% of the CPU power, making any attempt of defrauding the network practically infeasible.

By contrast, in case of malicious nodes that take control of the entire network, or a significant part of it, attackers can propose invalid transactions and switch off the trust on which the system relies on.

We can distinguish two types of nodes: full nodes, also called miners, and accounts. Full nodes run the consensus and validate transactions, while accounts are linked to a wallet. People with an account can spend their digital coins and execute transactions, but they do not play any role in the consensus. As result, this type of node does not waste energy to solve the cryptographic puzzle.

Proof-of-Stake The concept of Proof of Stake was firstly introduced in 2012 by Peercoin [23] and, since then, several other Blockchains were based on the same idea. The protocol is quite simple: a node partakes to the consensus procedure (it generates or validates the next block) proportionally to the stakes it bets.

The nodes do not need to compete with each other for solving a mathematical puzzle, resulting in a more efficient and energy saving procedure and as an incentive to keep the nodes loyal, the leader receives a reward for propagating the block.

There are two types of Proof-of-Stake design:

- Chain-based Proof-of-Stake: periodically, a leader is elected in a semi-random fashion and delivers the next block to be added to the chain.
- Consortium consensus-Byzantine fault tolerance (BFT) protocol: a node gets elected in a semi-random fashion and a voting procedure runs in order to find the consensus. In the voting procedure, every node counts proportionally to the stake it bets.

As shown in Figure 2.6, validators are selected to produce the next block based on their stake. Although often designed with random functions to prevent a front-running consensus, a larger amount staked by a validator could give them

a higher chance of producing the next block. Proposed blocks by validators are then propagated to the rest of the set, who verify and add the approved block to the Blockchain.



Figure 2.6: Proof-of-Stake

In addition, there are two types of nodes: a) a node can just have an account and start to spend coins or b) a node can also participate in the consensus, vote and eventually propose a block (only the leader proposes a new block).

A node becomes the leader in a two-step procedure:

- 1) The first step takes into account the amount of money a user deposits into the system.
- 2) The second phase adds some kind of randomness to this process. Particularly, to avoid that wealthy nodes may get richer receiving the reward for delivering invalid transactions, a random election model is adopted.

This last method has a drawback: nodes are more incentivized to keep the token for a long time rather than to spend them.

This activity of voting consists of checking that all data transactions are not being double-spent. This verification is not computationally intensive and the parties can easily misbehave by voting for all blocks.

In fact, with the aim to get a reward, nodes can start supporting several branches of the chain because it is effortless. This bad behaviour can favour malicious parties and make the final agreement hard to find.

Pure Proof-of-Stake Pure Proof-of-Stake (Pure PoS) is a new consensus procedure implemented by Algorand to solve many of the presented problems that affect the Proof-of-Work and Proof-of-Stake.

The Algorand consensus is organized in rounds and for each round two phases are highlighted. During the first phase every participant runs a sortition function to elect the leader (as usual the leader is in charge of proposing the next block).

In the second phase, a subset of members is pseudo-randomly selected to become part of a committee group and runs the Algorand Agreement BA^* [9]. As the name suggests, the BA^* is a customized version of the BFT protocol revisited by Algorand.

The committee members are elected based on their weights, practically, every participant has a weight that is proportional to the stake it owns. Participants with high weights have more chances to be elected as committee members. The committee members vote for the proposed block and may influence the final decision on that block.

Hence, to preserve the correctness of the network, stakeholders (honest participants) are incentivized to become leaders or committee members.

Now there are two questions: how does the Sortition function and Algorand Agreement BA^* work?

We might see the Sortition function as a fair lottery that cannot be manipulated by malicious parties. The lottery elects the leader as well as the committee members and takes into account the number of token owned by each node, in the same way the Proof-of-Stake does. Hence, if a node owns a lot of token, its chance to be elected increases.

The key factor of the Sortition function is the “Verifiable Random Function” (VRF) [26], which is a random number generator that takes a public seed and the node’s role to generate a random value. Sortition returns three values: a random number generated by the VRF, a proof that the output provided by the VRF is correct and a number x that expresses the probability of a node to become a committee member.

Typically, every participant is elected proportionally to the coins it owns. Every other node can later verify the validity of the value x using a VerifySortition function.

Once a leader is elected, he propagates the block to the committee members who run the BA^* . Nobody can know in advance who will be elected to run the consensus in the start of each round.

To prevent Sybil attacks wherein a reputation system is subverted by forging identities in peer-to-peer networks, Algorand assigns a weight to each user. BA^* is designed to guarantee consensus as long as a weighted fraction (a constant greater than $2/3$) of the users are honest. In Algorand, we weigh users based on the money in their account. Thus, as long as more than some fraction (over $2/3$) of the money is owned by honest users, Algorand can avoid forks and double-spending.

Figure 2.7 summarizes the main strengths and weaknesses of each of the consensus procedures explained so far.

Type	Idea	Strength	Weakness
Proof-of-Work	Participants race to solve complex cryptographic puzzles	Hard to control the majority of CPU power	Greedy of energy
Proof-of-Stake	Each participant's influence on the choice of a new block is proportional to its stake	To hold the 51% of the stake is way harder than controlling the 51% of the computing power	Requires little resources for forging a new block
Pure Proof-of-Stake	Users are randomly and secretly selected with a lottery to propose blocks and vote on block proposals	A transaction is confirmed in few seconds	Project that requires more testing and applications

Figure 2.7: Review of types of Consensus

PPoS vs. Proof-of-Work Pure Proof-of-Stake has numerous advantages over Proof-of-Work. At first the Proof-of-Work algorithm is extremely expensive and wasteful. In fact, mining often requires specialized hardware to be competitive and consumes an enormous amount of electricity.

In addition, only professional miners who can invest in racks of specialized mining equipment can expect to make a profit. Accordingly, only they participate in and benefit from block generation and since only one user solves the puzzle and generates the new block, all the efforts of other miners is wasted. By contrast, Algorand's consensus protocol does not require participants to solve cryptographic puzzles in order to propose or validate blocks. Indeed, any

user who is online and possesses stake is eligible to participate in the consensus protocol and block generation does not require any expensive computation. Furthermore, Proof-of-Work leads to a concentration of power and centralization as a result of miners pooling their resources. These mining pools can erase blocks or change the order of blocks if they wish or if they're bribed to do so. With Pure Proof-of-Stake, on the other hand, malicious users do not gain any advantage by splitting their stake into many accounts or by pooling into a single one.

In addition, in Proof-of-Work systems, blocks take a long time to be propagated to the network and such slowness and lack of scalability are insufficient for serving a global economy or any financial application.

With Algorand's low computation and communication overhead, however, blocks are propagated within seconds, therefore, the protocol is able to scale to many users and sustain a high transaction rate.

Finally, with Proof-of-Work, there is a chance that two users could solve for a valid block at the same time. When two nodes get a valid block simultaneously, the Blockchain forks into two because different groups of users may see different candidates for the next block.

In contrast, the Algorand Blockchain never forks. Two blocks can never be added to the chain at once because only one block can have the required threshold of committee votes, at most, one block is certified and written to the chain in a given round.

Accordingly, all transactions are final in Algorand. When the consensus protocol decides on a block, this decision is never changed. Every honest user soon learns of this decision, and no honest user ever thinks that a different block at the same height was chosen.

Once a block appears, users can rely on the transactions it contains immediately and they can be confident that the block will forever be part of the chain.

PPoS vs. Delegated Proof-of-Stake Delegated Proof-of-Stake (DPoS) is an approach in which a fixed number of elected entities, delegates, are selected to create blocks. Delegates are voted into power by the users of the network who each get a number of votes proportional to the number of token they own on the network.

However, what Delegated Proof-of-Stake offers in scalability, it sacrifices in decentralization, and therefore, security. DPoS is inherently centralized, there is no guarantee that all delegates will remain honest.

In contrast to Delegated Proof-of-Stake, Pure Proof-of-Stake does not put a small set of users in charge of block generation, and users do not need to delegate their voting power to the selected few.

Every user may propose and vote on blocks with a probability that is directly proportional to their stake, and there is no special group of users for an attacker to target.

PPoS vs. Bonded Proof-of-Stake Bonded Proof-of-Stake (BPOS) is an approach in which any number of users set aside part of their stake in order to influence block generation. They lock up part of their stake for a certain amount of time and in return they get a chance proportional to that stake to select the next block (their voting power in the protocol is proportional to the amount of stake they are willing to lock up).

Once the deposit is in place, it cannot be removed until a specified amount of time has passed and if these users are dishonest, they forfeit their deposit along with the privilege of participating in the consensus process.

One primary drawback to Bonded Proof-of-Stake is that users reduce their ability to spend their stake by participating in the consensus protocol. In contrast to Bonded Proof-of-Stake, Pure Proof-of-Stake does not require users to set aside part of their stake in order to participate in the consensus protocol, and participating in the consensus protocol does not reduce a user's ability to spend their stake.

Verifiable Random Function (VRF) A Verifiable Random Function is a cryptographic primitive that maps inputs to verifiable pseudorandom outputs. VRFs were introduced by Micali, Rabin, and Vadhan in 1999 [26].

Algorand pioneered the use of VRF to perform secret *cryptographic sortition* [18] to select committees to run the consensus protocol and this allows the Algorand Blockchain to achieve the performance necessary to support many users.

By now, other Blockchain projects are using the idea in the consensus protocols to perform leader or committee selection.

VRF Syntax and Properties A VRF is a triple of algorithms *Keygen*, *Evaluate* and *Verify*.

- $Keygen(r) \rightarrow (VK, SK)$. On a random input, the key generation algorithm produces a verification key VK and a secret key SK pair.
- $Evaluate(SK, X) \rightarrow (Y, \rho)$. The evaluation algorithm takes as input the secret key SK , a message X and produces a pseudorandom output string Y and a proof ρ .
- $Verify(VK, X, Y, \rho) \rightarrow 0/1$. The verification algorithm takes as input the verification key VK , the message X , the output Y and the proof ρ . It outputs 1 if and only if it verifies that Y is the output produced by the evaluation algorithm on inputs SK and X .

Informally, for a fixed key-pair and an input X , a VRF produces a unique pseudorandom verifiable output.

More technically:

- The output Y is unique, that is, it's impossible to find another output (together with a valid proof) for a given key-pair (VK, SK) and input X .
- The output Y is pseudorandom, which means that it looks “random” to any third party that does not see the associated proof ρ .
- The above properties should hold even if the key-pair (VK, SK) is adversarially chosen by the user.

It is interesting to note that while VRF is similar to a signature scheme, it has

crucial distinctions:

- a) in a signature scheme, many possible valid signatures may exist for a given input, and
- b) signatures are unpredictable overall, but some of the signature bits may be highly predictable. VRF outputs satisfy a stronger pseudo-randomness property.

VRF use in the Algorand Blockchain Must remember that at the core of the Algorand Blockchain is a fast Byzantine Agreement protocol. However, the agreement is not performed between all users in the network, it is confined to a small randomly chosen committee of users for each round.

Each user in the Algorand network holds a secret key SK , while her verification key VK is publicly known to everyone. Each user participating in the Algorand network wants to decide if she should be in the committee to run Byzantine Agreement for block r , she needs to do the following:

1. Compute $\text{Evaluate}(SK, Qr) \rightarrow (Y, \rho)$. Here, Qr is a seed string that is available to everyone in the system.
2. Check that Y falls within a certain range $[0, P]$ that depends on the stake the user holds in the system.

If the above check passes, then the user holds a proof consisting of values (Y, ρ) which validates their committee membership for block r . Given (Y, ρ) and the user's verification key VK , anyone can verify that Y is a valid unique output and that it falls within a desired range (hence, validating that the user holding VK was indeed chosen to serve on the committee for block r).

A random self-selected small committee is selected through cryptographic sorition. 1. The uniqueness and pseudo-randomness properties of the VRF are crucial in ensuring that no user can brute-force their way through multiple outputs Y until she finds one that falls within her desired range.

Such attack is mitigated by the uniqueness property of the VRF because once the seed Qr is fixed, the VRF function can only be used to produce a single output. Moreover, the verification key VK must have entered into the system

before block r , at the time when the seed Qr was essentially unpredictable. Altogether, a user cannot bias the output Y to fall within their desired range, which is where the pseudo-randomness property comes into play.

2. The above computation is extremely cheap. Each user only needs to perform a single *Evaluate* function computation to decide if they are selected to serve on the committee.

3. A new and independent committee is chosen for each round of the Byzantine Agreement. This is possible due to the unique property of the Algorand Byzantine Agreement called Player Replaceability [10]. Briefly, different users are able to participate in different rounds of the Byzantine Agreement without having to pass any state to each other.

This allows Algorand to satisfy a high level of security, where a user is allowed to be dynamically corrupted after any message they send as part of the protocol.

Participation Keys An account that participates in the Algorand consensus protocol is eligible and available to be selected to propose and vote on new blocks in the Algorand Blockchain. To reduce exposure, users which must be online to participate in the consensus protocol do not use their spending keys (the keys they use to sign transactions) for consensus, instead, a user generates and registers a participation key for a certain number of rounds [14].

It also generates a collection of ephemeral keys, one for each round, signs these keys with the participation key, and then deletes the participation key. Each ephemeral key is used to sign messages for the corresponding round, and is deleted after the round is over.

Using participation keys ensures that a user's token are secure even if their participating node is compromised. Deleting the participation and ephemeral keys after they are used ensures that the Blockchain is forward-secure and cannot be compromised by attacks on old blocks using old keys.

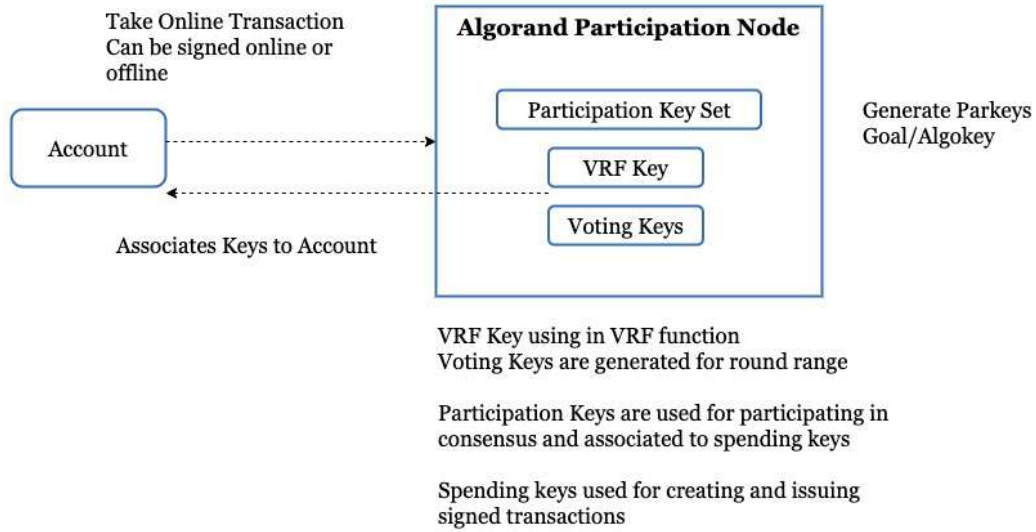


Figure 2.8: Participation Keys

Keys and Addresses Algorand uses Ed25519 [6] high-speed, high-security elliptic-curve signatures. The keys are produced through standard, open-source cryptographic libraries packaged with each of the SDKs. The key generation algorithm takes a random value as input and outputs two 32-byte arrays, representing a public key and its associated private key (these are also referred to as a public/private key pair).

These keys perform important cryptographic functions like signing data and verifying signatures.

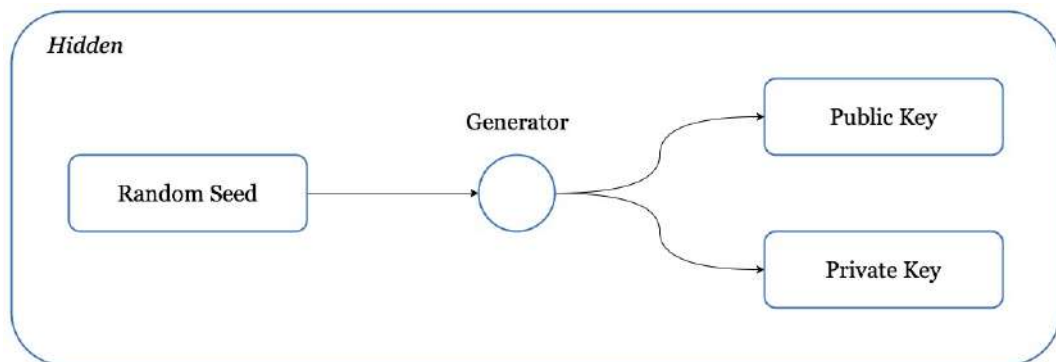


Figure 2.9: Public/Private Key Generation

For reasons that include the need to make the keys human-readable and robust to human error when transferred, both the public and private keys undergo transformations.

The output of these transformations is what the majority of developers, and usually all end-users, see. In fact, the Algorand developer tools actively seek to mask the complexity involved in these transformations.

Public Key to Algorand Address The public key is transformed into an Algorand address, by adding a 4-byte checksum to the end of the public key and then encoding it in base32. The result is what both the developer and end-user recognize as an Algorand address (the address is 58 characters long).

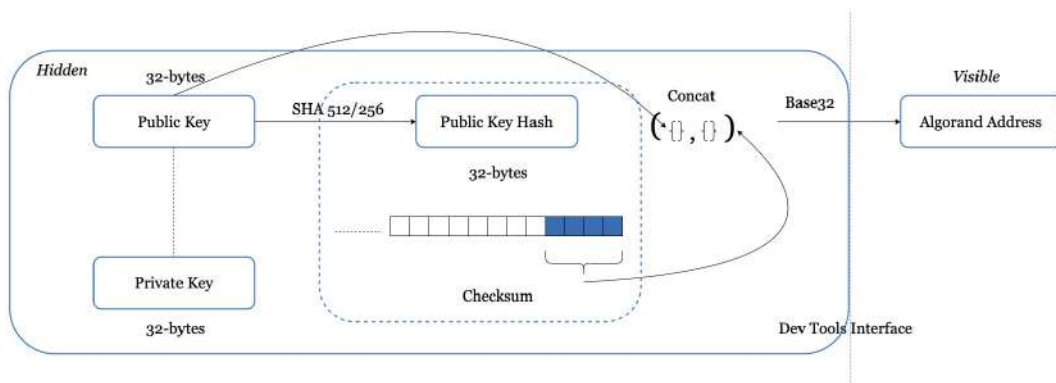


Figure 2.10: Public Key to Algorand Address

Private Key to Base64 private key A Base64 encoded [20] concatenation of the private and public keys is a representation of the private key most commonly used by developers interfacing with the SDKs.

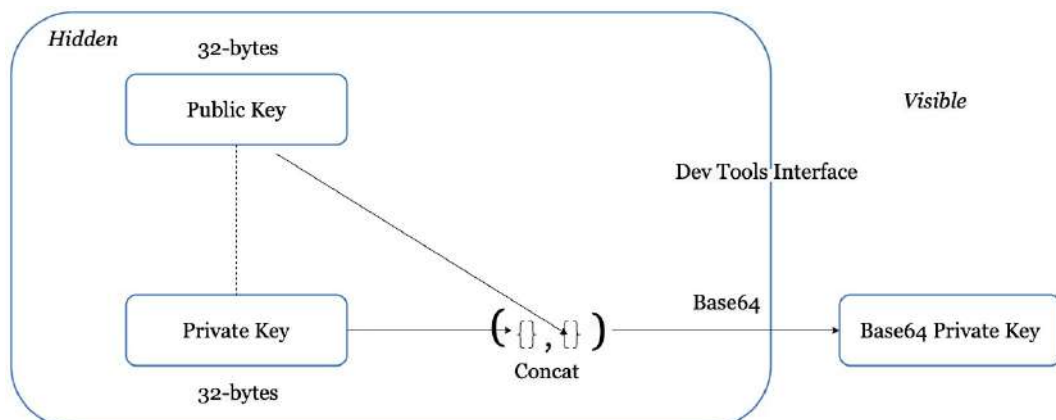


Figure 2.11: Base64 Private Key

Private Key to 25-word mnemonic The 25-word mnemonic is the most user-friendly representation of the private key. It is generated by converting the private key bytes into 11-bit integers and then mapping those integers to the bip-0039 [31] English word list, where integer n maps to the word in the n th position in the list.

By itself, this creates a 24-word mnemonic. A checksum is added by taking the first two bytes of the hash of the private key and converting them to 11-bit integers and then to their corresponding word in the word list.

This word is added to the end of the 24 words to create a 25-word mnemonic (this representation is called the private key mnemonic).

Both the base64 representation of a private key and the private key mnemonic are considered private keys.

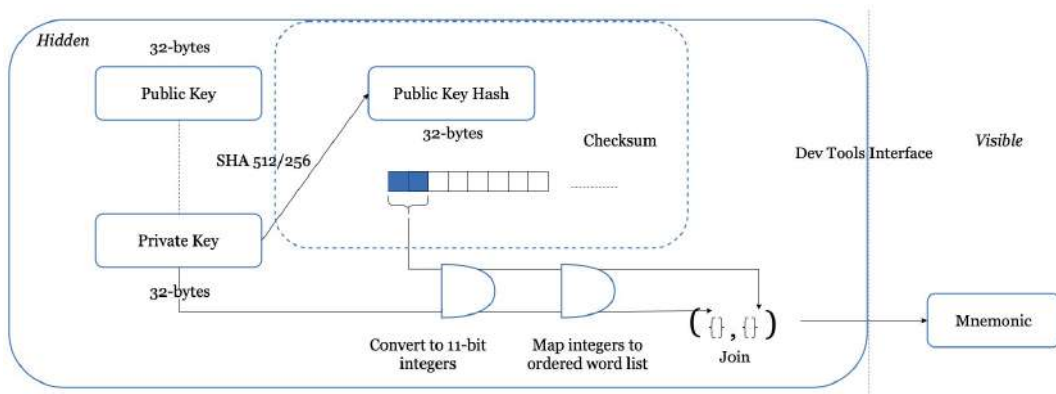


Figure 2.12: Private Key Mnemonic

The Algorand Consensus Protocol Consensus refers to the way blocks are selected and written to the Blockchain. Algorand uses the VRF [26] described before to select leaders to propose blocks for a given round. When a block is proposed to the Blockchain, a committee of voters is selected to vote on the block proposal. If a super majority of the votes are from honest participants, the block can be certified.

What makes this algorithm a Pure Proof of Stake is that users are chosen for committees based on the number of ALGO in their accounts. Committees are made up of pseudorandomly selected accounts with voting power dependent on their online stake (it is as if every token gets an execution of the VRF).

For a committee membership this means higher stake accounts will most likely have more votes than a selected account with less token. Using randomly selected committees allows the protocol to still have good performance while allowing anyone in the network to participate.

Consensus [10] requires three steps to propose, confirm and write the block to the Blockchain. These steps are:

1) Block Proposal

2) Soft Vote

3) CertifyVote

Each is described below, assuming the ideal case when there are no malicious users and the network is not partitioned (i.e., none of the network is down due to technical issues or from DDoS attacks).

All messages are cryptographically signed with the user's participation key and committee membership is verified using the VRF in these steps:

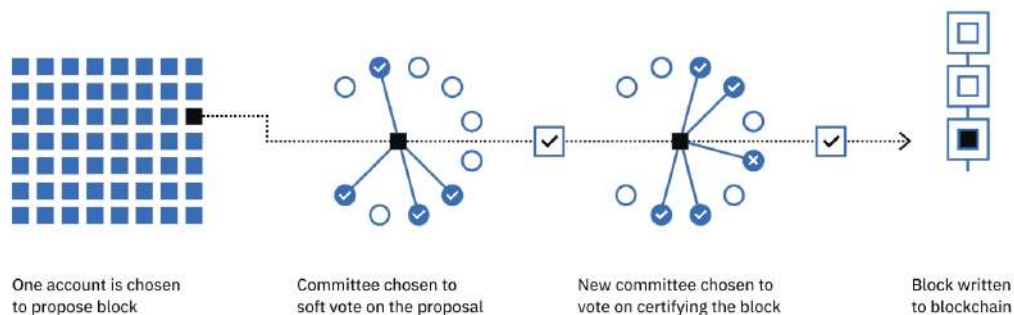


Figure 2.13: Phases of the Consensus

Block Proposal In the block proposal phase, accounts are selected to propose new blocks to the network. This phase starts with every node in the network looping through each online account for which it has valid participation keys, running Algorand's VRF to determine if the account is selected to propose the block.

The VRF acts similar to a weighted lottery where the number of ALGO that

the account has participating online determines the account's chance of being selected. Once an account is selected by the VRF, the node propagates the proposed block along with the VRF output, which proves that the account is a valid proposer.

We then move from the propose step to the soft vote step.

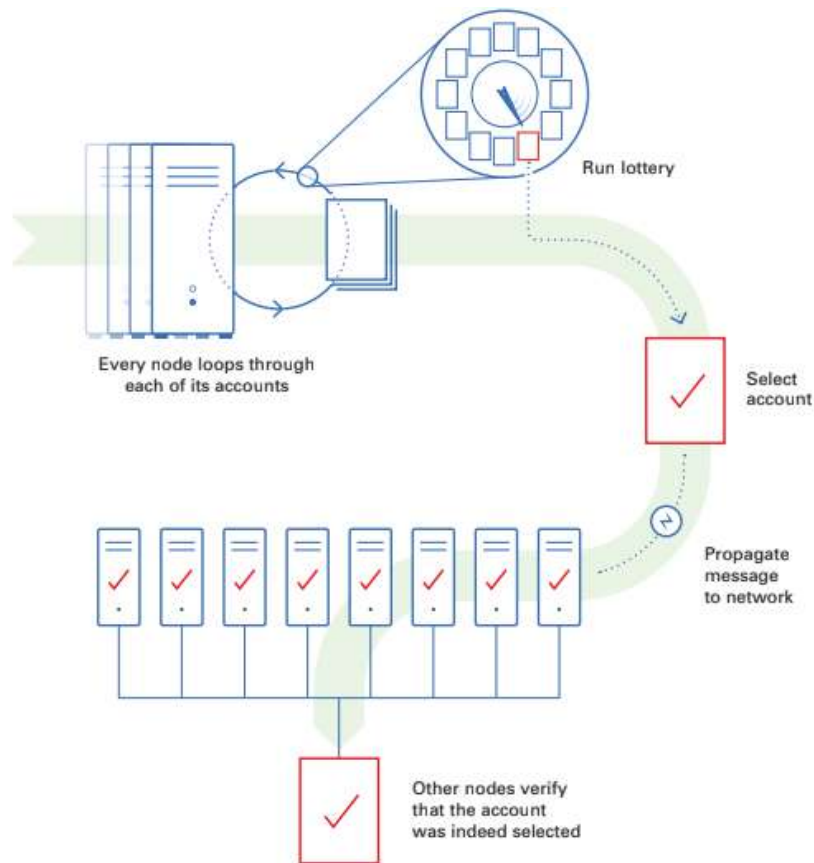


Figure 2.14: Block Proposal

Soft Vote The purpose of this phase is to filter the number of proposals down to one, guaranteeing that only one block gets certified. Each node in the network will get many proposal messages from other nodes.

Nodes will verify the signature of the message and then validate the selection using the VRF proof. Next, the node will compare the hash from each validated winner's VRF proof to determine which is the lowest and will only propagate the block proposal with the lowest VRF hash.

This process continues for a fixed amount of time to allow votes to be propa-

gated across the network.

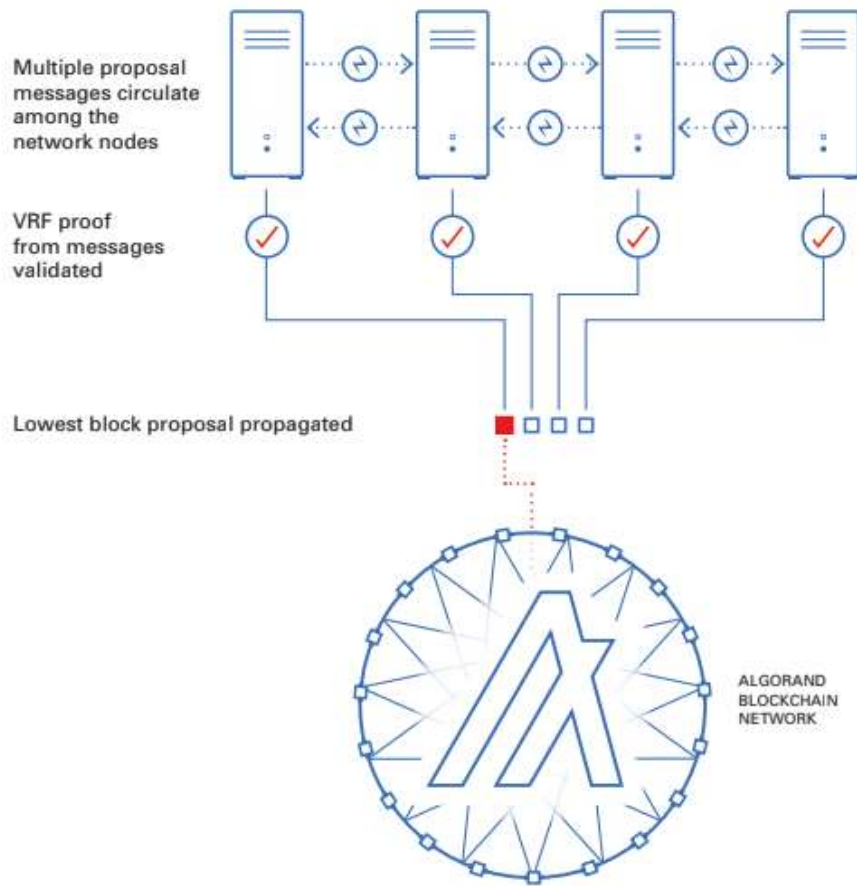


Figure 2.15: Soft Vote - Part One

Each node will then run the VRF for every participating account it manages to see if they have been chosen to participate in the soft vote committee.

If any account is chosen it will have a weighted vote based on the number of ALGO the account has, and these votes will be propagated to the network.

These votes will be for the lowest VRF block proposal calculated at the timeout and will be sent out to the other nodes along with the VRF Proof.

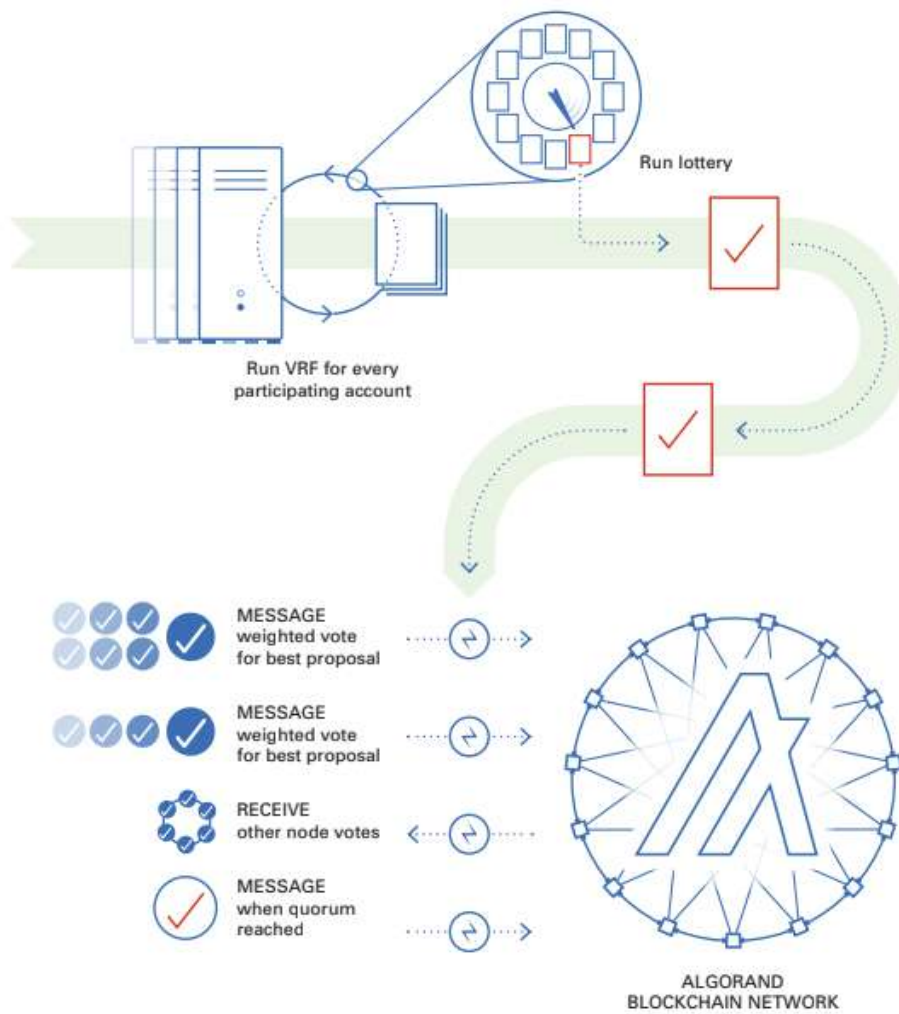


Figure 2.16: Soft Vote - Part Two

A new committee is selected for every step in the process and each step has a different committee size (this committee size is quantified in ALGO).

A quorum of votes is needed to move to the next step and must be a certain percentage of the expected committee size. These votes will be received from other nodes on the network and each node will validate the committee membership VRF proof before adding to the vote tally.

Once a quorum is reached for the soft vote the process moves to the vote step.

Certify Vote A new committee checks the block proposal that was voted on in the soft vote stage for overspending, double-spending, or any other problems. If valid, the new committee votes again to certify the block. This is done in a

similar manner as the soft vote where each node iterates through its managed accounts to select a committee and to send votes.

These votes are collected and validated by each node until a quorum is reached, triggering an end to the round and prompting the node to create a certificate for the block and write it to the ledger. At that point, a new round is initiated and the process starts over.

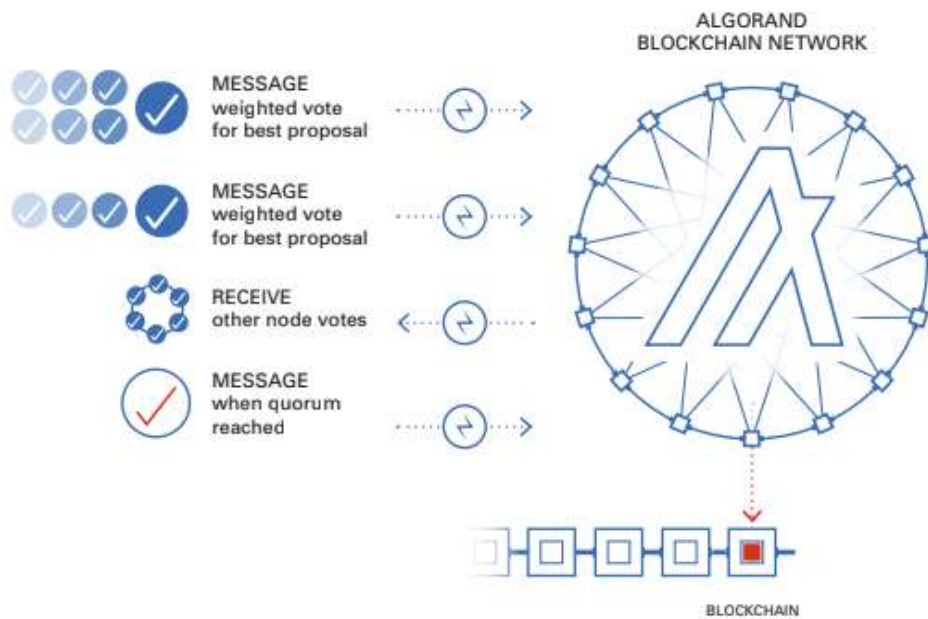


Figure 2.17: Certify Vote

If a quorum is not reached in a certifying committee vote by a certain timeout then the network will enter recovery mode [1].

There is a tool called *carpenter* that helps to visualize the protocol and how it is proceeding, reads the node log file and formats the output. Each line displayed in the tool will have a message that is relative to the step and will display when a proposal or a vote is accepted or rejected.

When accepted votes are displayed, the number of votes is also included in parentheses. When the threshold for the vote is reached a message will be displayed letting the user know the next step in the protocol is about to start. Here is a typical example of what *carpenter*'s output looks like:


```

12849566.0.2:      VoteAccepted(80/469) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(124/2655) 45CHX-12849566.0.1I
12849566.0.2:      VoteAccepted(67/536) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(54/590) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(59/649) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(61/710) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(72/782) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(58/840) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(57/897) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(52/949) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(64/1013) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(61/1074) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(30/1104) 45CHX-12849566.0.2I
12849566.0.2:      VoteAccepted(33/1137) 45CHX-12849566.0.2I
12849566.0.2:      ThresholdReached(1137/1112) 45CHX-12849566.0.2I
12849566.0.0:      RoundConcluded 45CHX- |
12849566.0.0:      RoundStart 45CHX- |
:      ProposalAssembled -12849567.0.0I
12849567.0.1:      ProposalAccepted VE3WF-12849567.0.0I
12849567.0.1:      BlockPipelined VE3WF-12849567.0.0I
12849567.0.0:      BlockAssembled VE3WF-12849567.0.0I
12849567.0.1:      ProposalAccepted ILM0Z-12849567.0.0I
12849567.0.1:      BlockPipelined ILM0Z-12849567.0.0I
12849567.0.0:      BlockAssembled ILM0Z-12849567.0.0I

```

Figure 2.18: Carpenter’s output

Algorand Networks The first challenge to overcome is the tradeoff between decentralization and scalability. Intuitively, it is much faster and more efficient to write and maintain a single database than it is to keep many distributed databases.

The problem becomes more pronounced as expand to a global distribution as the Algorand protocol does. In order to achieve transaction throughput and time to finality that enables this global adoption, the protocol needs to propagate information rapidly throughout the network to ensure all participants have the most recent Blockchain [16].

There is a direct relationship between how fast a message propagates and the throughput of the network: a decrease in propagation speed will either require a reduction in block size or result in an increase in latency between blocks, which translates to a longer time to transaction finality.

To solve this problem, the Algorand network has two types of nodes to simultaneously optimize decentralization and high transaction throughput:

1) **Relay nodes**

2) **Non-Relay nodes**

- Relay nodes serve as central network hubs and maintain connections to many other nodes. Relay nodes have high-bandwidth network connections which allow for highly efficient communication paths, ultimately reducing the number of hops in communication.

They do this by accumulating protocol messages from Non-Relay nodes and other relay nodes connected to them, performing deduplication, signature checks, and other validation steps and then re-propagating the valid messages.

- Non-Relay nodes are connected to much fewer nodes, most of which are relay nodes. They represent an address stake and hold participation keys for proposing and voting on blocks within the consensus algorithm.

As shown in Figure 2.19, there are three Public Algorand Networks paired with the functionality to create private networks using any protocol version.

- **MainNet** is the primary Algorand Network with real value assets, including Algorand's native currency, ALGO.

- **TestNet** mirrors MainNet in terms of its protocol version, but it has test ALGO, available through a faucet, and a different genesis block, which means that the state of accounts and distribution of funds is different.

- **BetaNet** is where new protocol-level features are released for initial testing. Therefore, quality and features may not be final, and protocol upgrades and network restarts are common.

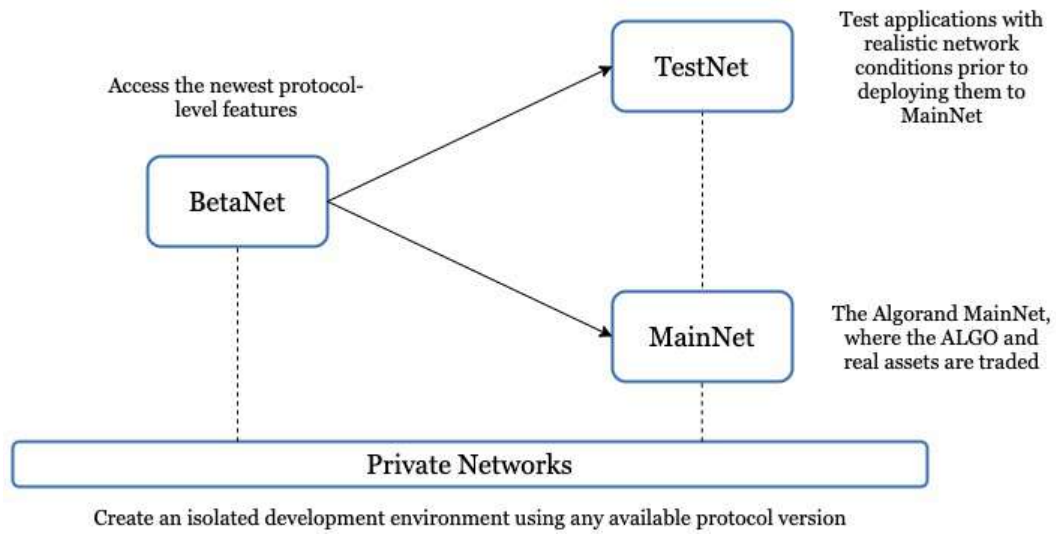


Figure 2.19: Algorand Networks

Figure 2.20 shown the differences between Algorand networks in terms of protocol version, genesis distribution, accessibility and reliability.

	MainNet	TestNet	BetaNet
Protocol Version	Current	Current	Future
Genesis Distribution	Unique	Unique	Unique
ALGO Accessibility	For sale	Free from faucet	Free from faucet
Network Reliability	Most Stable	Very Stable, but restarts are possible	Experimental, frequent restarts

Figure 2.20: Side-by-Side Network Comparison

2.4 IoT

The Internet of Things (IoT) is opening the potential for billions of connected smart devices to communicate with each other. However, it is not straightforward to manage and control these many devices in the future.

Since most IoT applications rely primarily on information from sensors, there is a necessity of specific storage and security for preventing information ma-

nipulation and unauthorized sharing.

Although the term IoT has being so broad and widely used, there is no specific definition. However, in 2012 it was defined by the International Telecommunication Union (ITU) as “a global infrastructure for the information society enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies” [19].

The fundamental characteristics of the IoT are sensing capabilities, connectivity, large scale network, dynamic system, intelligence capabilities, big data, unique identity, autonomous decision and heterogeneity.

Essentially, it transforms every “thing” in a “smart thing”: it enhances every aspect of people daily lives through the power of data collection, artificial intelligence algorithms, and networks.

In addition, as could be predicted, devices have become smaller, cheaper, and more powerful over time and IoT is exploiting these ad-hoc small devices to deliver its precision, scalability, and versatility.

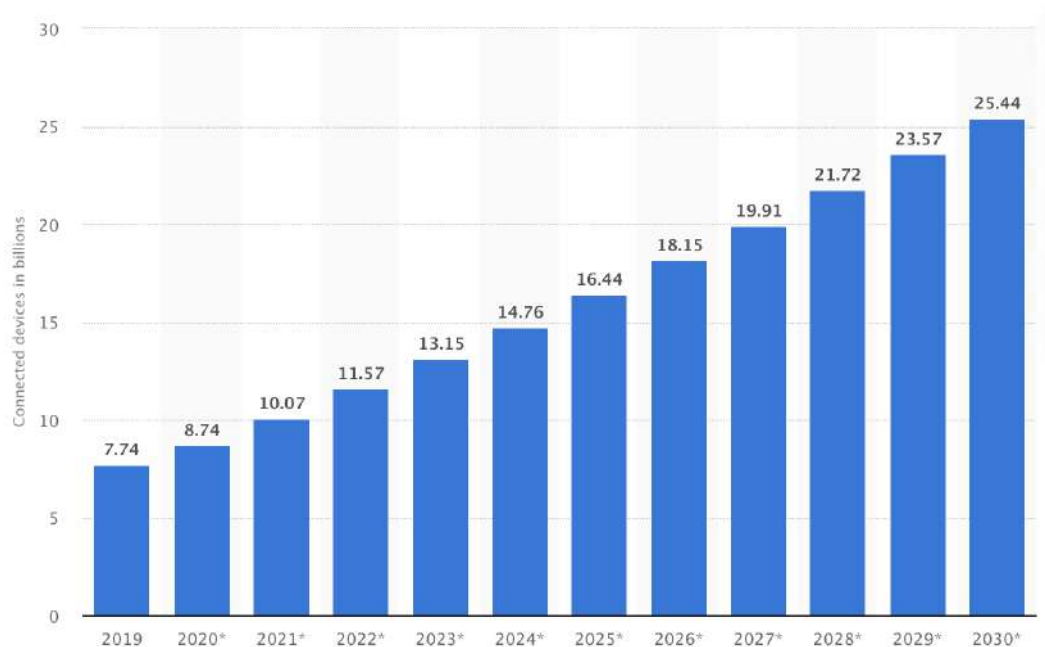


Figure 2.21: Number of IoT connected devices worldwide 2019-2030

As reported in 2.21, according to Statista [33], the number of Internet of Things

(IoT) devices worldwide is forecast to almost triple from 8.74 billion in 2020 to more than 25.4 billion IoT devices in 2030.

In 2020 the highest number of IoT devices is found in China with 3.17 billion devices. In fact, IoT devices are used in all types of industry verticals and consumer markets, with the consumer segment accounting for around 60% of all connected devices in 2020.

2.4.1 Low-power Devices

As Industrial, Consumer, Smart Home, Health and Wellness systems are growing and are becoming more connected also due to the spread of IoT, certain embedded designs are now required to manage high processing loads and complex applications with rich Human Machine Interfaces (HMI).

However, in many IoT applications, to keep costs down, human-machine interaction is reduced to a minimum.

Under these conditions, the ability to harness open-source software stacks while maintaining low power and real-time performance are key requirements.

One of the most important company in the world that provide solutions for embedded systems and IoT is **STMicroelectronics** [34]. It produces different devices grouped in the “STM32” family. Within this thesis, only low-power devices are used and specifically one of these is part of the STM32 (STMicroelectronics) family general-purpose 32-bit microprocessors (MPUs) based on the heterogeneous architecture combining Arm Cortex-A and Cortex-M as shown in Figure 2.22.

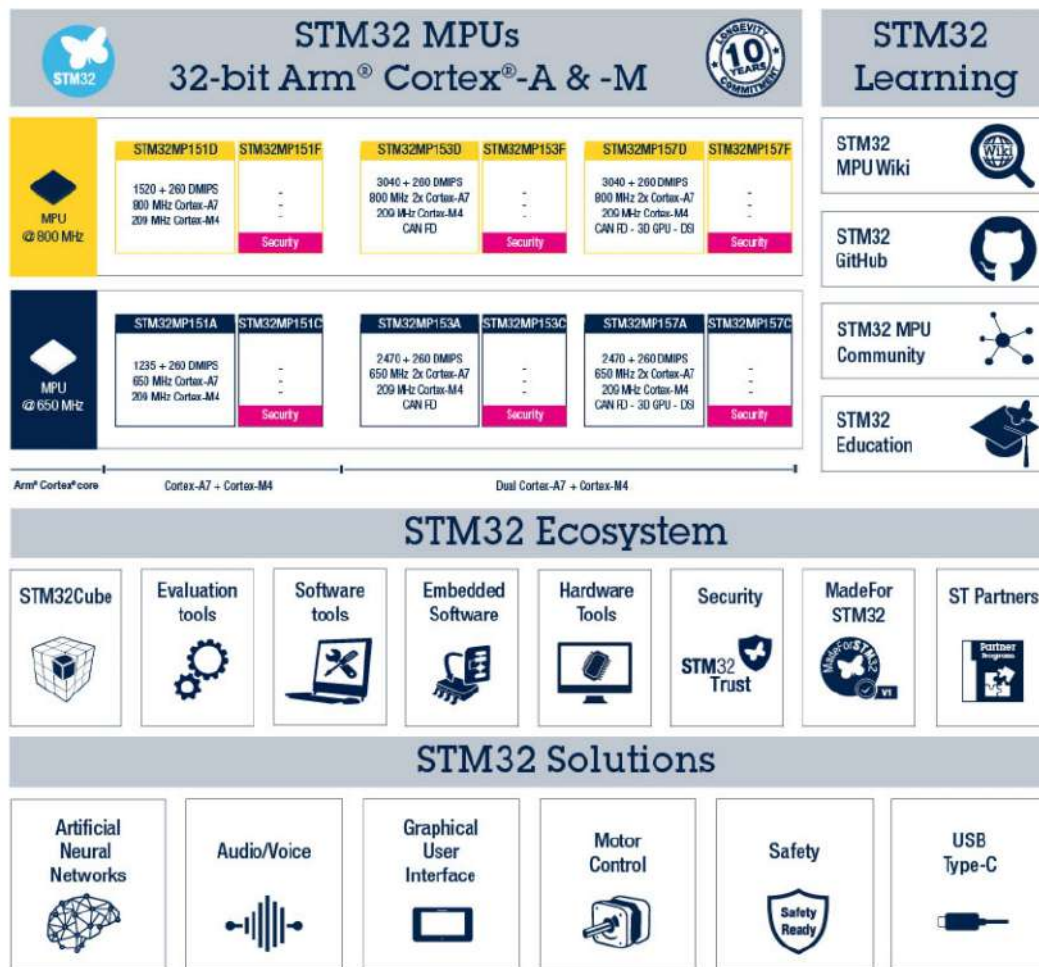


Figure 2.22: STM32 Arm Cortex MPUs

STM32MP1 Series The STM32MP1 Series, that is listed in Figure 2.23, is based on a heterogeneous single or dual Arm Cortex-A7 and Cortex-M4 cores architecture, which strengthens the ability to support multiple and flexible applications achieving best performance and power.

The Cortex-A7 core provides access to open-source operating systems (Linux/Android), while the Cortex-M4 core leverages the STM32 MCU ecosystem.



<div>  <div> STM32MP1 Series Arm® Cortex®-A7 – up to 800 MHz </div>  </div>											
Acceleration <ul style="list-style-type: none"> Dual core Arm® Cortex®-A7 processor L1 and L2 caches 3D Graphic Processing Unit* Floating Point Unit + Arm® Neon™ Arm® Cortex®-M4 209 MHz coprocessor MDMA + DMA LPDDR2/LPDDR3 16/32**-bit 533 MHz DDR3/DDR3L 16/32**-bit 533 MHz Connectivity <ul style="list-style-type: none"> 2 x USB2.0 HS Host USB2.0 OTG FS/HS 3 x SDMMC/SDIO USART, UART, SPI, I2C 2 x (TT)FD-CAN2.0* Gigabit Ethernet IEEE 1588*** FMC (NAND Flash) Camera I/F Dual mode Quad-SPI DSI 2 Gbit/s* 	Product lines	Cortex®-A7 core	f _{core} (MHz)	Cortex®-M4 core	f _{core} (MHz)	3D GPU	f _{core} (MHz)	HW Crypto	FD-CAN	MIPI-DSI	Junction temperature
	STM32MP151A	1	650	1	209	-	-	-	-	-	-40°C to 125°C
	STM32MP151C							•	-	-	
	STM32MP151D	1	800	1	209	-	-	-	-	-	-20°C to 105°C
	STM32MP151F							•	-	-	
	STM32MP153A	2	650	1	209	-	-	-	2	-	-40°C to 125°C
	STM32MP153C							•	-	-	
	STM32MP153D	2	800	1	209	-	-	-	2	-	-20°C to 105°C
	STM32MP153F							•	-	-	
	STM32MP157A	2	650	1	209	•	533	-	2	•	-40°C to 125°C
	STM32MP157C							•	-	-	
	STM32MP157D	2	800	1	209	•	533	-	2	•	-20°C to 105°C
	STM32MP157F							•	-	-	

Figure 2.23: STM32MP1 Series

In particular, STM32MP157 microprocessors are based on the flexible architecture of a Dual Arm Cortex-A7 core running up to 800 MHz and Cortex-M4 at 209 MHz combined:

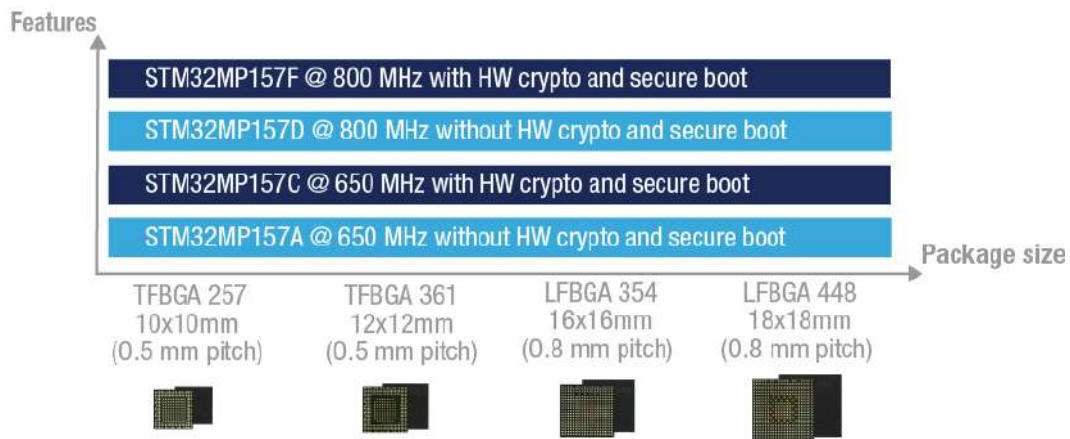


Figure 2.24: STM32MP157

In this work we use **STM32MP157A-DK1** (shown with display removed in Figure 2.25 and Figure 2.26), a Discovery Kit designed as complete demon-

stration and development platforms for STMicroelectronics Arm-based dual Cortex-A7 32 bits and Cortex-M4 32 bits MPUs in the STM32MP1 Series and their STPMIC1 companion chip [35].

The STM32MP157A-DK1 board neither include the “WLAN + Bluetooth”, nor the MB1407 daughterboard DSI display (available on STM32MP157C-DK2 board), so the connection to the device takes place through Ethernet.



Figure 2.25: STM32MP157A-DK1 top view

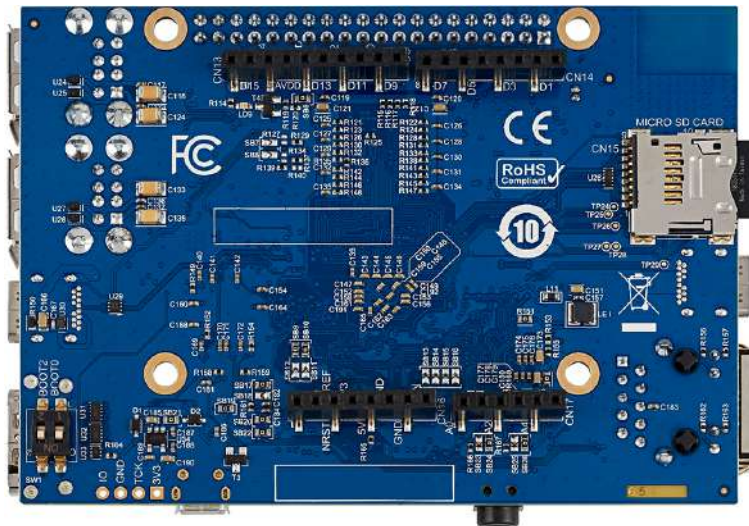


Figure 2.26: STM32MP157A-DK1 bottom view

This Discovery Kit using STM32 MPU OpenSTLinux Distribution software that is a subpart of the STM32 MPU Embedded Software distribution, a set

of software, system build and development tools created to ease the development to be done on top of STM32 MPU devices.

There are three software packages for optimize each development phase of a project: Starter package (STM32MP1Starter), Developer package (STM32MP1Dev) and Distribution package (STM32MP1Distrib).

The one that we use is STM32MP1Starter that is to quickly and easily start with any STM32MP1 microprocessor device and some of its features are shown in Figure 2.27.

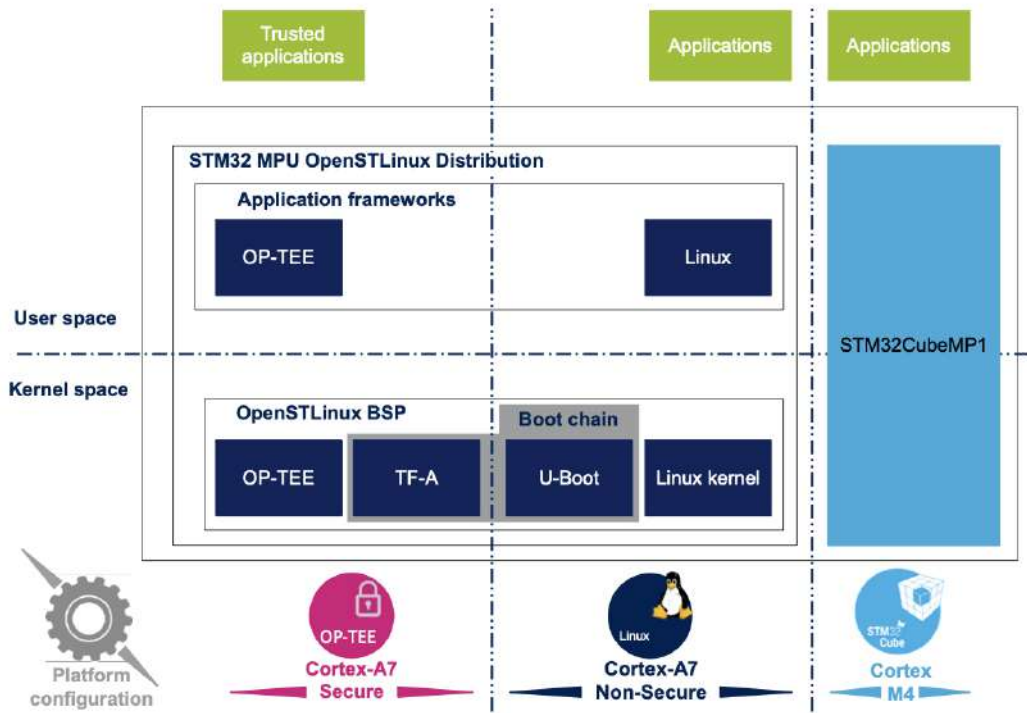


Figure 2.27: STM32MP1Starter

Raspberry Pi 4 Model B Another device that is used within this thesis to make performance comparisons is the Raspberry Pi 4 Model B that offers increases in processor speed, multimedia performance, memory, and connectivity compared to the prior-generation Raspberry Pi 3 Model B+.

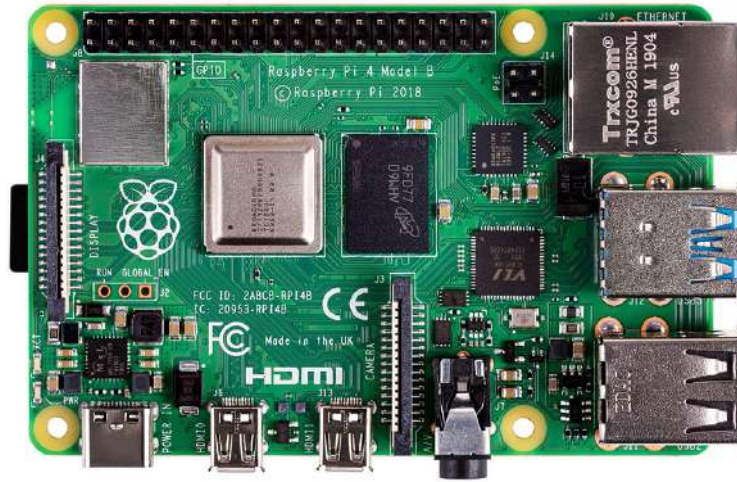


Figure 2.28: Raspberry Pi 4 Model B

This Raspberry model represented in Figure 2.28 includes a Quad Core Broadcom BCM2711 (Cortex A-72) (ARM v8) with 4 GB LPDDR4-2400 SDRAM memory and the operating system installed is Debian-based Raspberry Pi OS (Buster 32-bit Release) [17].

2.4.2 Application Performance Monitoring

The applications have evolved very quickly from standalone applications to client-server to distributed and lastly cloud-based. And that is precisely why application performance management (APM) has evolved to follow suit.

The term APM refers to the management of software application performance to ensure an expected level of service, as measured by performance metrics and user experience monitoring. Application Performance Management solutions aim to detect and pinpoint application performance issues before real users are impacted.

One of these that we use is **Netdata** because it is installed by default on STM32MPU Embedded Software distribution as shown in Figure 2.29.

Netdata is a system that helps collect all possible metrics from systems and applications, visualize these in real-time, and troubleshoot complex performance problems [30].

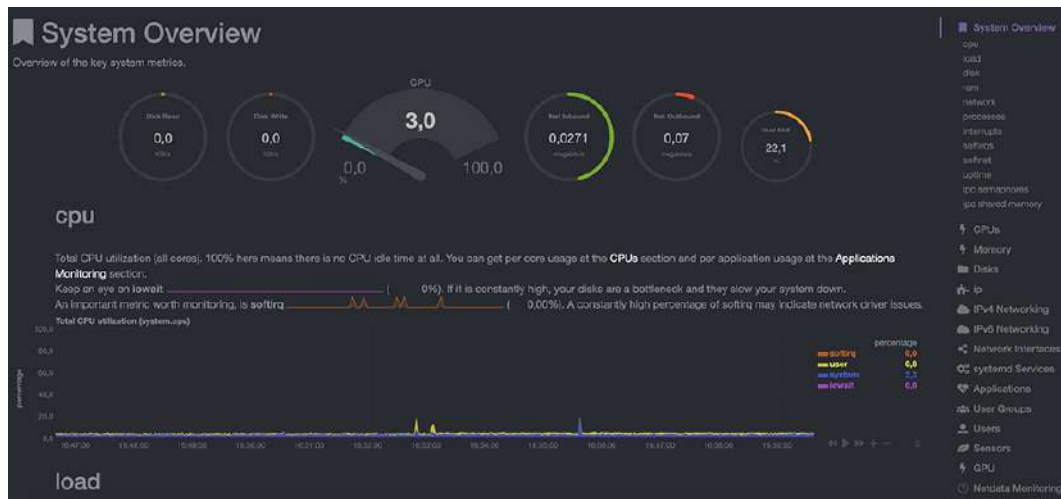


Figure 2.29: Netdata Interface

Netdata uses two components: Netdata Agent and Netdata Cloud:

- Netdata Agent collects thousands of metrics from systems, hardware, and applications with zero configuration.
- Netdata Cloud instead is a web application that gives real-time visibility for entire infrastructure. While Netdata Cloud offers a centralized method of monitoring Agents, metrics data is not stored or centralized in any way, but remains with nodes and is only streamed to browser.

Chapter 3

State of the art

The purpose of this chapter is to present a general review over the state of the art concerning the analysis and performance of low-power devices based on Blockchain and, more formally, on the correlation with IoT technology.

First of all, we define some basic concepts regarding IoT and Blockchain and in the second part we introduce several published papers discussing this correlation. Among the other aspects, the efficiency in energy consumption and the timeliness of transactions as also considered as important factors.

Finally, we briefly introduce some use cases in which the Algorand Blockchain has been already applied in the IoT domain.

3.1 Blockchain and IoT

The Blockchain was firstly used for cryptocurrencies and financial transactions that are executed and stored by all nodes in the network.

This technology shares some common features that involve decentralization, transparency, immutability, security, traceability, cost reduction and autonomy, so suitable for IoT applications such as healthcare, smart homes, smart city, transportation and others.

There are some similarities and differences between IoT and Blockchain, and Figure 3.1 provides a summary of comparison between them.

Items	IoT	Blockchain
Privacy	Lack of privacy	Ensures the privacy of the participating nodes
Bandwidth	IoT devices have limited bandwidth and resources	High bandwidth consumption
System Structure	Centralized	Decentralized
Scalability	IoT considered to contain a large number of devices	Scales poorly with a large network
Resources	Resource restricted	Resource consuming
Latency	Demands low latency	Block mining is time-consuming
Security	Security is an issue	Has better security

Figure 3.1: Comparison between IoT and Blockchain

In fact, in the last few years the two technologies have been used together in different works. This combination can provide countless benefits for various applications. However, the integration is not always an easy task and different researches have already addressed the topic.

Several published papers discuss the integration of IoT and Blockchain.

For example, Ziyang Wang et al. [38] research the combination of IoT and Blockchain from the perspective of model construction and performance evaluation. The presented model uses the Blockchain and InterPlanetary File System (IPFS), a protocol and peer-to-peer network for storing and sharing data in a distributed file system to realize some functions: transaction generation, block packaging and transaction query.

The Blockchain isolates IoT devices and service providers, thereby eliminating the correlation between responses. As the underlying database, IPFS provides better distributed large-capacity storage, concurrency, and query performance for model. The final metric used to evaluate are the average latency and the throughput of transactions in each of independent operations for each group of experiments.

In the same way, Minhaj Ahmad Khan et Khaled Salah [22] presented a review of IoT and its security challenges concerning IoT-layered architectures. The

authors identified security requirements of the IoT system and state of-the-art solutions, presenting Blockchain technology as a vital empowering to resolve most of the security issues of the IoT system.

3.2 Blockchain on IoT low-power devices

The main challenge in integrating Blockchain with IoT is the processing capabilities of end devices. In fact, end IoT devices are low-power devices and cannot handle the processing power and memory capacity.

Due to constraints of IoT ecosystems consisting of heterogeneity of exploited devices, different computing capabilities of involved appliances, and different support to existing encryption algorithms, the time required to perform encryption algorithms for all the objects involved in Blockchain-based IoT ecosystem can be different.

Some solutions like [28] and [27] use these devices to collect the data from sensors and perform computation in a distributed architecture, independently managing their transactions and mining.

In the first work, as a first step, layer-wise security issues are identified in IoT applications, while, then details about the integration of the Blockchain technology in term of addressing IoT security are presented.

In that case, the Blockchain was exploited for authenticating devices through the Ethereum ecosystem.

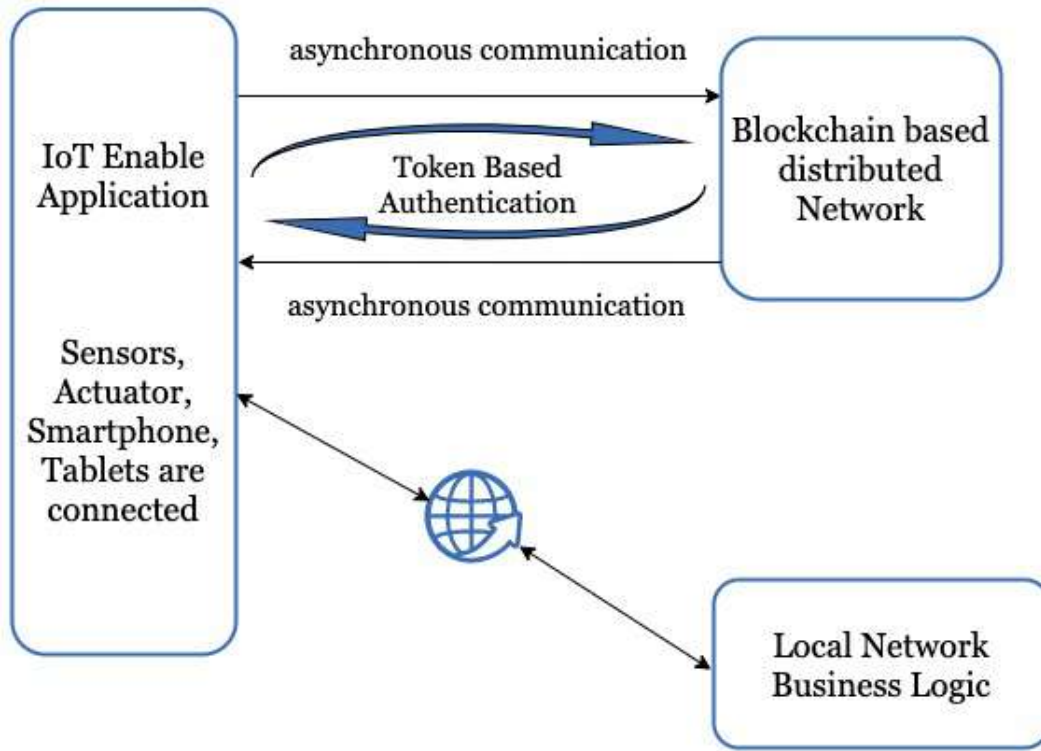


Figure 3.2: Proposed Blockchain based solution architecture

As shown in Figure 3.2, IoT allows the use of wireless or wired devices in various applications. Initially, all the smart devices connected to the applications need to have authentication in the outside network that is Blockchain network. Once devices are registered, they can perform different activity as per their features locally.

Similarly, users are also required to authenticate in the Blockchain network initially. After that, they can monitor or access the different smart object present in the network. The proposed work suggested that only authentication and authorization need to be done in the Blockchain network.

In the second work an Ethereum-based Blockchain on heterogeneous IoT nodes, is again implemented. Some of these nodes are connected to the Blockchain over an Ethernet-based connection, whereas others connect through a WiFi-based connection, forming a hybrid network connection.

The Blockchain nodes they consist of both regular computing platforms such as PCs as well as constrained IoT Edge nodes consisting of Raspberry Pi boards. Further, the devices themselves have different specifications and processing

capabilities, as outlined in Figure 3.3.

Features	Node-1	Node-2	Node-3	Node-4
Device	Raspberry Pi3-B+	Raspberry Pi3-B+	Dell Power Edge T410 server	Raspberry Pi3-B+
Processor	Quad Core 64 bit ARM cortex at 1.2 GHz	Quad Core 64 bit ARM cortex at 1.2 GHz	16x4 Core 64 bit at 2.67 GHz	Quad Core 64 bit ARM cortex at 1.2 GHz
RAM Network Connection	1 GB Ethernet	1 GB Ethernet	32 GB Ethernet	1 GB WiFi

Figure 3.3: Blockchain node specifications for this implementation

Returning to the field of low-power devices, the work [37] describe an implementation in which a private Ethereum network is chosen as a Blockchain platform and devices Raspberry Pi act as a lightweight node to transact with a smart contract in a full node as depicted in Figure 3.4.

In addition, the performance evaluation of two realistic traffic flow, store and access transactions, such as latency, execution time and throughput are highlight.

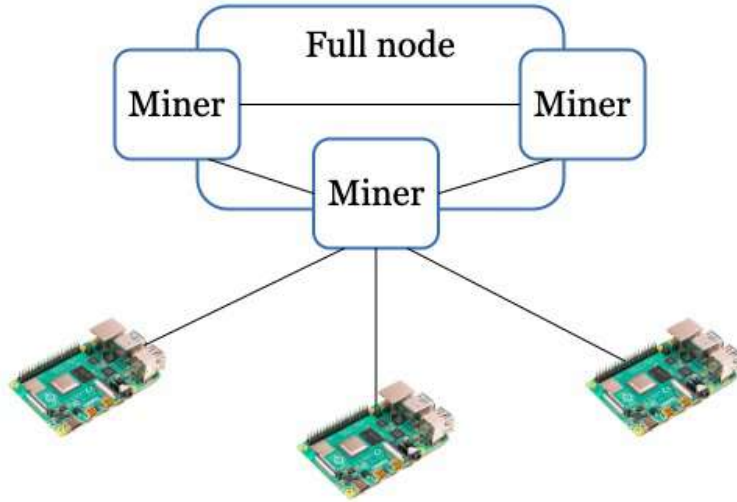


Figure 3.4: Deployment of this system

Another work [36] describe an implementation of secure IoT device based on STM32, a family of 32-bit microcontroller integrated circuits by STMicroelectronics, that enables functionalities of IOTA, a DLT specifically designed for

the use in the IoT that aims to mitigate two customary features of Blockchain solutions that are not well-suited for the IoT: transaction fees and scalability. IOTA uses Tangle that except for the first genesis transaction (genesis), the rest of the transactions are initiated using the Markov Chain Monte Carlo (MCMC) algorithm to select two transactions and verify if they are legal. Since IOTA needs to verify the other two transactions when initiating a transaction, the more transactions initiated per unit time, the higher the throughput of transaction verification, thereby eliminating the dependence of the distributed ledger on miners.

It was demonstrated that the Light Node implemented, using the STM32F746ZG Nucleo development board shown in Figure 3.5, is suitable for joining the IOTA network, indeed, it successfully publishes transactions in the distributed ledger. It was also guaranteed how this Light Node ensures protection of the stored private data and guarantees safe service execution thanks to the security features provided by the STM32 microcontroller.

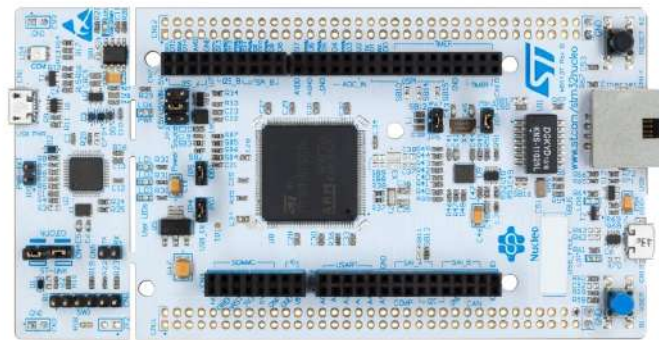


Figure 3.5: STM32F746ZG Nucleo

In Figure 3.6 are report the specs of the IoT platforms employed and of STM32F746ZG board.

Platform	SoC	CPU Core	Frequency	RAM	Cores	Prepare and sign a transaction			PoW
						1 = 1	1 = 2	1 = 3	
STM32F7 Nucleo	STM32F746Z	Cortex-M7	216 MHz	up to 320 kB	1	395 ms	471 ms	568 ms	323.7 s
TI Launchpad	CC2650	Cortex-M3	48 MHz	20 kB	1	6382 ms	7716 ms	-	not feasible
Raspberry Pi 3B	BCM2837	Cortex-A53	1200 MHz	1 GB	4	328 ms	372 ms	-	82.9 s

Figure 3.6: Results obtained using this platform

In another work [21] was proposed an Ethereum Blockchain-based IoT architecture used by low-power IoT devices to communicate with each other, validate transactions, and provide security.

As low-power IoT controller it was used as shown in Figure 3.7 an ESP32, solitary 2.4 GHz Wi-Fi and Bluetooth combo chip planned with the TSMC ultra-low-control 40 nm technology.

It is a module that intended to accomplish the best power and RF execution, appearing flexible and with unwavering quality in a wide assortment of uses and power situations.

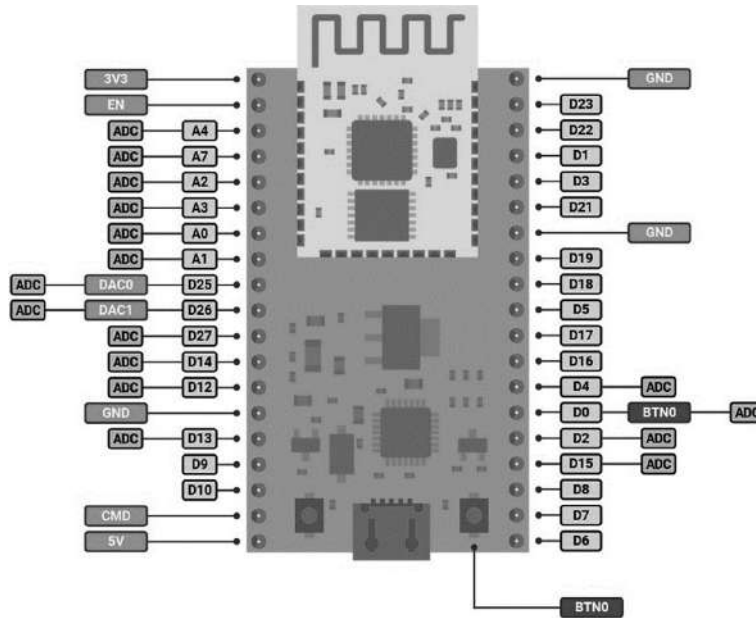


Figure 3.7: ESP32 DevKit

The architecture is suitable for low-power IoT devices and use less computational power in the end device side, but the network usage is intensive and is

due to the resource exhausting consensus mechanism employed by Ethereum and transaction approval time is also not ideal.

Figure 3.8 shows the basic architecture and, in this system, all the low-power devices communicate to the Ethereum Blockchain, and all the nodes are connected to the same Blockchain.

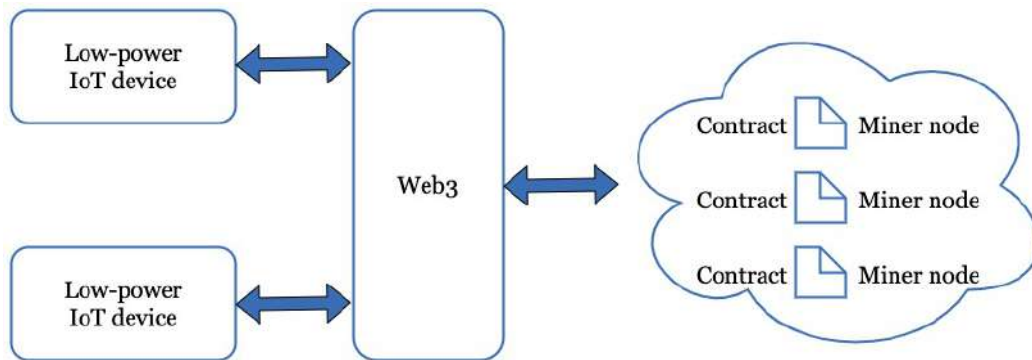


Figure 3.8: Proposed general block diagram

3.3 Algorand Blockchain applied to IoT

Hundreds of organizations, such as fintechs, startups, financial services, institutions, and Decentralized Finance (DeFi) are currently building their applications by using the Algorand Blockchain technology.

Some of them are creating ecosystems and interfaces to the Internet of Things by using this Blockchain.

A practical example that exploits the Blockchain Algorand in the IoT environment is the PlanetWatch project that decentralizes, incentivizes and gamifies air quality monitoring. The company collects data through air sensors to then reports them on Algorand to create an immutable air quality repository accessible to all participants.

The sensor owners, as described in the WhitePaper [32] and illustrated in Figure 3.9, receive PLANET token, an Algorand Standard Asset (ASA), in exchange for contributing data to the repository.

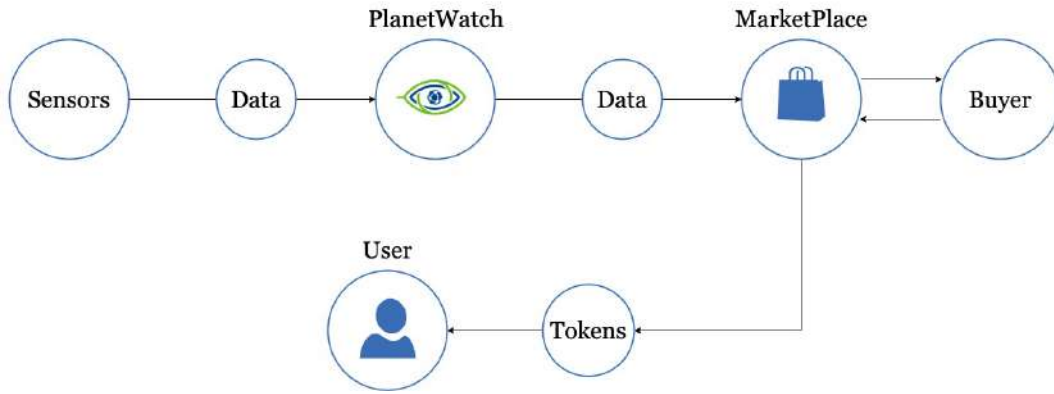


Figure 3.9: PlanetWatch ecosystem

Another company called Climatetrade aims at helping other companies to achieve their sustainability goals by offsetting CO2 emissions and financing climate change projects.

Moreover, Fondazione Ugo Bordoni exploits Algorand to expose technical advantages offered over other available Blockchains.

The Foundation is committed to support and implement the activities of 5G, in the investigation of the potential of Artificial Intelligence and in the design and management of digital networks and services.

In conclusion, a valid and effective Blockchain must be taken to enable fast processing of transactions and coordination among billions of connected devices.

As the number of interconnected devices grows, the distributed ledger technology provides a viable solution to support the processing of the large number of transactions.

Algorand, that enable faster transactions with far lower fees, can be a valid solution, but propounds few features to be successfully applied to IoT networks. The purpose of this thesis is to find a valid solution to exploit this Blockchain on low-power IoT-devices, even though such futures are still in the testing phase, to have a solid base on which future works could be conducted.

Chapter 4

Requirements Analysis

Build Algorand-enabled applications means that applications, directly or indirectly, reads from or writes to the Algorand Blockchain.

Writing to the Algorand Blockchain is synonymous of issuing a transaction that will later be confirmed within a block and reading from the Blockchain means reading back transactions that have been confirmed within prior blocks. In this chapter, taking into account what we learned in *second chapter*, we propose specific solutions to achieve the goal of this thesis.

Initially, we conduct a preliminary analysis on the Algorand development environment to then identify the requirements and design a general architecture for our tests.

4.1 Preliminary analysis

The following is a brief primer on some terms and relationships of the components that comprise the Algorand environment. Figure 4.1 illustrates these components and how they fit together [14].

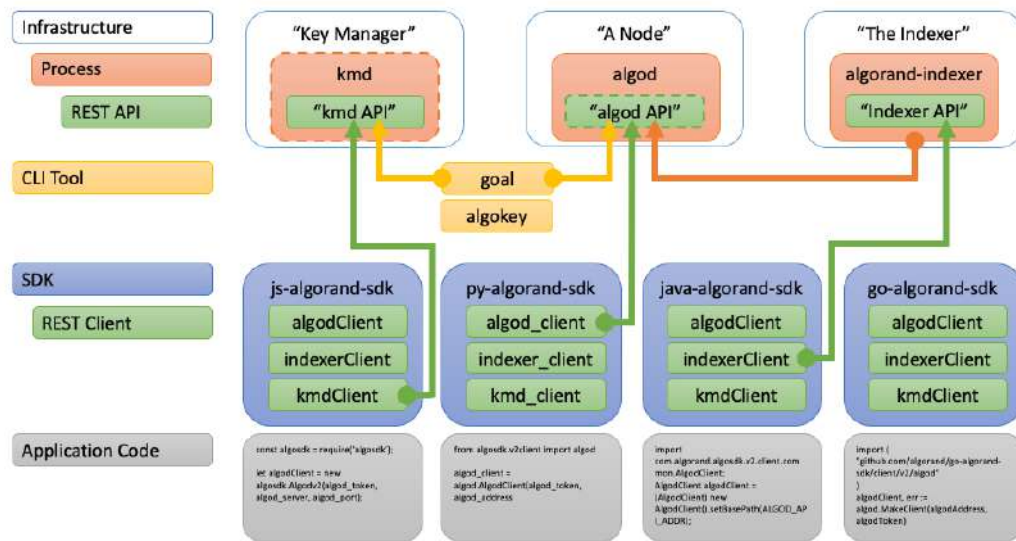


Figure 4.1: Algorand Dev Stack

As depicted in the Figure, three main levels can be identified in the Algorand Dev Stack: an “Infrastructure” layer that represents the processes and application programming interface compliant with the constraints of the REST architecture, an “SDK” layer that facilitates programming of software in a specific programming language and an “Application” layer that interfaces and provides services for application processes.

Algorand Blockchain is a distributed system of nodes each maintaining their local state based on validating the history of blocks and the transactions therein. State data is maintained by the consensus protocol which is implemented within the *algod* daemon, often referred to as the node software (*algod* is the main Algorand process for handling the Blockchain).

We have three command line utilities packaged to interact with Algorand node software: *goal*, *kmd* and *algokey*.

- *goal* is the primary tool for operating a node and it also contains functionality to manage keys, sign and send transactions, create assets and many other.
- *kmd* (key management daemon) handles interacting with clients private keys for accounts. This process is responsible for generating and importing spending keys, signing transactions, and interacting with key storage mechanisms like hardware wallets.

· *algokey* is a command line utility for generating, exporting and importing keys and can be used to sign single and multisignature transactions as well. As shown in Figure 4.1, Algorand officially supports four SDKs for developing applications: Javascript, Java, Python, and Go.

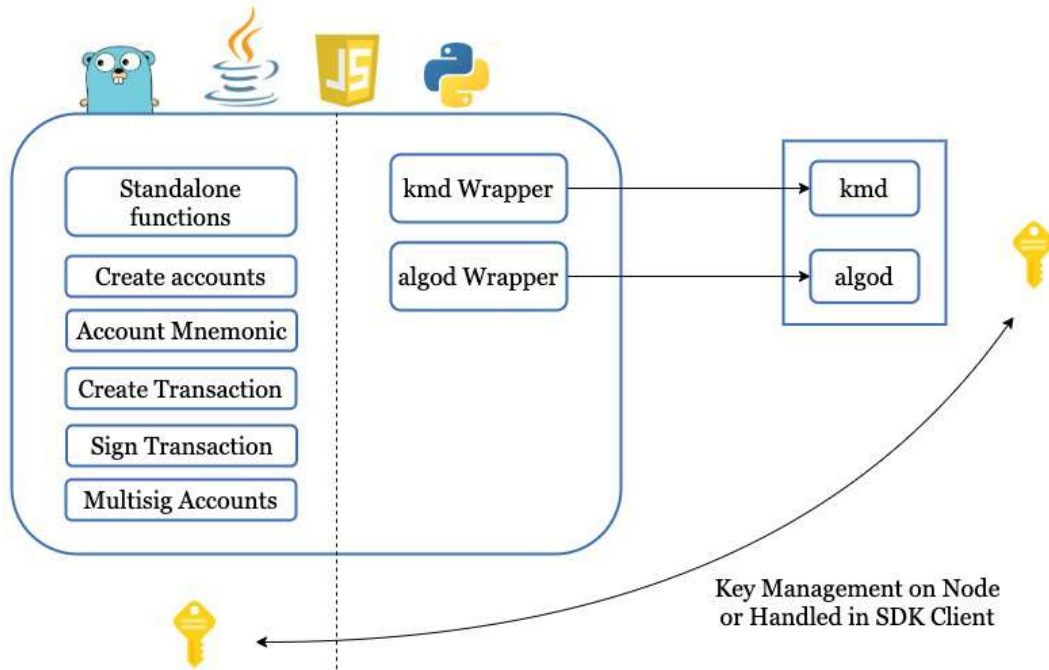


Figure 4.2: SDK Functionality

Algorand Features in Layer-1 In common Blockchain language, layer-1 handles basic payments and the consensus protocol, while layer-2 includes smart contracts and any other functionality not directly connected to the generation of new blocks [16].

Algorand implements core functionalities at layer-1 for three main reasons:

1. Security: any functionality implemented at layer-1 enjoys the highest level of security: that of the consensus protocol itself.
2. Compatibility: layer-1 functionality is interpreted in the same way by all Blockchain participants.
3. Efficiency: core functionalities implemented at layer-1 can take direct advantage of the system to optimize their performance in computation, storage, and communication.

By contrast, relying on virtual machines and other layer-2 apparatus often

results in unnecessary complexities and costs.

Due to this consideration, not all features are suitable for layer-1 implementation, in particular, to be part of the layer-1, it is necessary not to significantly slow down the consensus protocol.

Some of these features are:

- Algorand Smart Contracts (ASC1s) that are trustless programs that execute on chain where users can be confident that the program was run without error and the results were not tampered with.

This smart contracts are written in Transaction Execution Approval Language (TEAL), an assembly-like language, and are processed with a stack machine as shown in Figure 4.3.

The language is a non-turing-complete language that does not support looping, but does support forward branches. The primary purpose of this program is to return true or false.

The program will return true if and only if one positive value is left on the stack at the completion of the program execution, for all other conditions the program will return false.

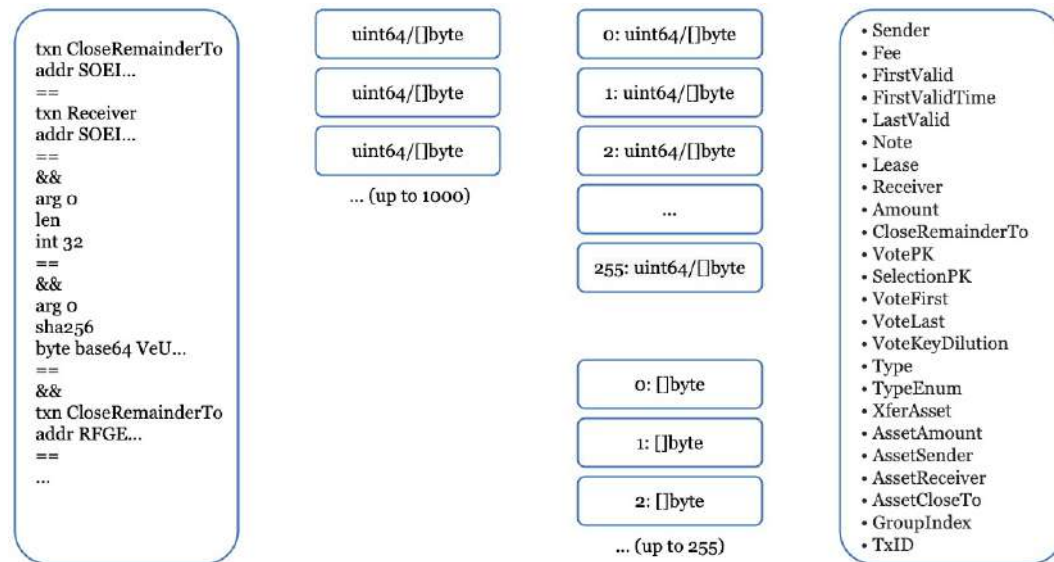


Figure 4.3: TEAL Architecture Overview

- Algorand Standard Assets (ASAs) that represent any type of asset on the Algorand Blockchain and these can include fungible, non fungible, restricted

fungible and restricted non fungible assets.

With these resources it is possible to represent stablecoins, loyalty points, system credits, in-game points and collectable items just to name a few examples in Figure 4.4.

FUNGIBLE TOKEN	NON FUNGIBLE TOKEN	RESTRICTED FUNGIBLE TOKEN	RESTRICTED NON FUNGIBLE TOKEN
In Game Points Stablecoins Loyalty Pointas System Credits Cryptocurrencies	- In Game Items - Supply Chain - Real Estate - Identity - Certifications - Collectables	- Securities - Gov't Issued Fiat - Certifications	- Real Estate - Ownership Registries - Regulatory Certifications

Figure 4.4: Example Asset Types

- Atomic Transfers that offer a secure way to simultaneously transfer a number of assets among a number of parties, specifically, many transactions are grouped together and either all transactions are executed or none of them are executed.

The result of grouping is that each transaction is assigned the same group ID and once all transactions contain this group ID, the transactions can be split up and sent to their respective senders to be authorized.

Figure 4.5 above shows the flow of an Atomic Transfers.

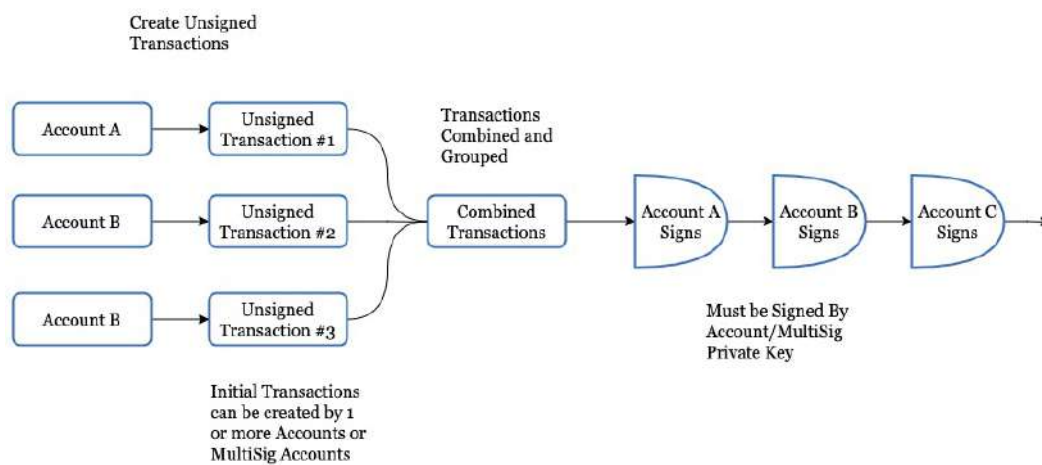


Figure 4.5: Atomic Transfer Flow

- Algorand Rekeying that allows users to change their Private Spending key without the need to change their Public Address.

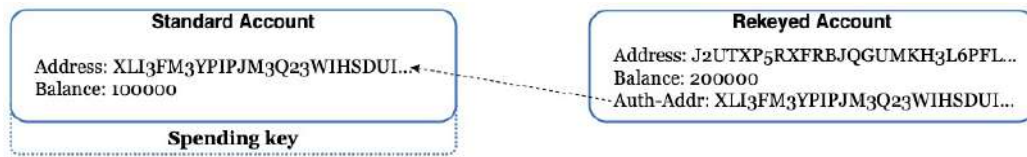


Figure 4.6: Rekeying

Conceptually illustrated in Figure 4.6, a “standard” account uses its private spending key to authorize from its public address. A “rekeyed” account defines the authorized address which references a distinct “foreign” address and thus requires the private spending key(s) thereof to authorize future transactions.

4.2 Requirements definition

In this section we report all the assumptions made to organize the work performed within this thesis with the aim of evaluating the possibility of using Algorand on various devices and compare the performance of each tested devices.

To test the applicability of Algorand to low-power devices, as already discussed, different operations have to be performed:

- Synchronize Algorand Network
- Perform write operations
- Perform read operations

At first, in fact, to perform operations on the Blockchain, the node must be perfectly synchronized.

In addition, the Blockchain is designed to be an append only structure. A user can only add more data, in the form of additional blocks and all previous data is permanently stored and cannot be altered. Therefore, the only operations associated are write operations to add data and read operations that query

and retrieve this data from the Blockchain.

For the purposes of this thesis, we used two different low-power devices present in the market:

- Raspberry Pi 4 Model B
- STM32MP157A-DK1

To make comparisons on the boards we installed first of all the same type of Algorand node, Non-Relay non-archival, using it to perform some features of this Blockchain, in particular synchronization of node, creation of the wallet and related accounts, participation key for helping to support the consensus protocol, private network, transaction, Atomic Transfers and Algorand Standard Asset.

The setup topology is shown in Figure 4.7 where a notebook is used for remote work on the boards using SSH protocol and low-power devices are connected through Ethernet cable on the network ports of the modem/router.

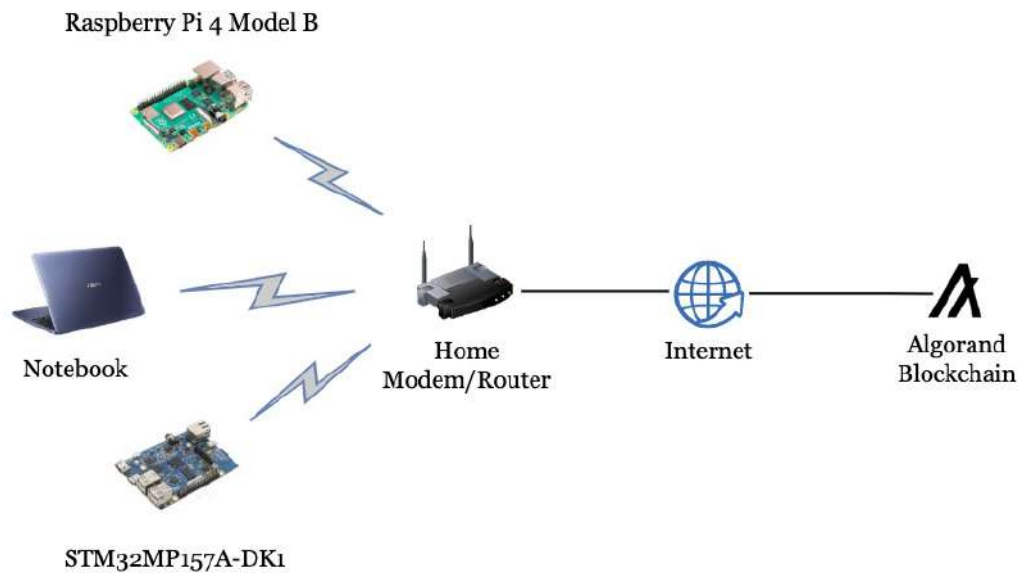


Figure 4.7: Setup topology

Chapter 5

Implementation

This chapter describes the implementation of the operations described in the previous chapter.

In the implementation and unit testing phase we used the Algorand Chain Testnet and a service called *dispenser* to obtain ALGO which have no monetary value, but only to allow development on the test network.

5.1 Raspberry Pi 4 Model B

With the aim of testing the exploitation of Algorand on a low-power device, as a first step, the operations described in the previous chapter were implemented on a Raspberry Pi 4 Model B.

The following list summarizes the three implementation phases:

- Board configuration
- Installation of Algorand node
- Installation of Netdata

At first, to install the Raspberry Pi OS 32-bit based on the official releases of Debian for ARM architecture on an SD card, we used the ‘Raspberry Pi Imager’ tool, that is officially provided By the Raspberry Pi OS team.

After completing all the installation steps of the OS, we created a folder to hold

the install package and files of node and download the updater script, checking that all permissions (e.g., execution permissions) were satisfied. Finally we run the installer within the node directory.

```
pi@raspberrypi:~ $ mkdir ~/node
```

```
pi@raspberrypi:~ $ cd ~/node
```

```
pi@raspberrypi:~/node $ wget https://raw.githubusercontent.com/
algorand/go-algorand-doc/master/downloads/installers/
update.sh
--2021-01-06 12:49:04-- https://raw.githubusercontent.com/
algorand/go-algorand-doc/master/downloads/installers/update.
sh
Resolution of raw.githubusercontent.com (raw.
githubusercontent.com)... 185.199.110.133, 185.199.111.133,
185.199.109.133, ...
Connect to raw.githubusercontent.com (raw.githubusercontent.
com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 17474 (17K) [text/plain]
Save to: "update.sh"
update.sh          100%[=====>]  17,06K  --.-
KB/s      in 0,01s
2021-01-06 12:49:05 (1,23 MB/s) - "update.sh" saved
[17474/17474]
```

```
pi@raspberrypi:~/node $ chmod 544 update.sh
```

```
pi@raspberrypi:~/node $ ./update.sh -i -c stable -p ~/node -d
~/node/data -n
Current Version = 0
updater
updater binary was downloaded
Latest Version = 8590196737
New version found
Checking for files matching: 'channel/stable/
node_stable_linux-arm_' in bucket algorand-releases
Update Downloaded to /tmp/tmp.0wb2o04pM5/8590196737.tar.gz
```

```

Expanding update...
Validating update...
Starting the new update script to complete the installation
...
... Resuming installation from the latest update script
Current Version = 0
Stopping node...
... node not running
Backing up current binary files...
Backing up current data files from /home/pi/node/data...
Installing new binary files...
mv: replace '/home/pi/node/update.sh', overriding mode 0544 (
  r-xr--r--)? y
Installing new data files into /home/pi/node/data...
Copying genesis files locally
Checking for new ledger in /home/pi/node/data
Cannot read genesis file /home/pi/node/data/genesis.json:
  open /home/pi/node/data/genesis.json: no such file or
  directory
Updating genesis files for default network
New genesis ID, resetting wallets
New Ledger - restoring genesis accounts in /home/pi/node/data
New Ledger - importing rootkeys for genesis accounts
Imported 0 keys
Applying migration fixups...
Deleting existing log files in /home/pi/node/data
Install complete - restart node manually

```

Every Algorand node has a data directory that is used to store the ledger and other configuration information. As part of this configuration a *genesis.json* file is used to specify the initial state of the Blockchain, its “genesis block”.

It contains the network name and id, the protocol version and the list of allocated addresses to start the chain.

As part of the installer, a genesis files directory is created under the node’s installed location for binaries and contains additional directories for each of the Algorand networks.

The network can be switched by either replacing the current genesis file located in the data directory with the specific network *genesis.json* or by creating a directory *testnetdata* and copying inside the specific network *genesis.json* file to use the Algorand TestNet network.

```
pi@raspberrypi:~/node $ mkdir testnetdata
```

```
pi@raspberrypi:~/node $ cp genesisfiles/testnet/genesis.json testnetdata/
```

After doing this, it is necessary to install, on the Raspberry Pi 4 Model B, the Netdata monitoring tool that, as explained in *second chapter*, instead, is installed as default tool on the STM32 board used through the STM32MPU Embedded Software distribution.

The recommended way to install the Netdata Agent on the Raspberry is by the one-line *kickstart script* which detects the distribution and installs the required system packages.

```
bash <(curl -Ss https://my-netdata.io/kickstart.sh)
```

Once installed, to open the local Agent Dashboard, as shown in Figure 5.1, we navigate to <http://NODE:19999>, replacing NODE with the hostname or IP address of system.

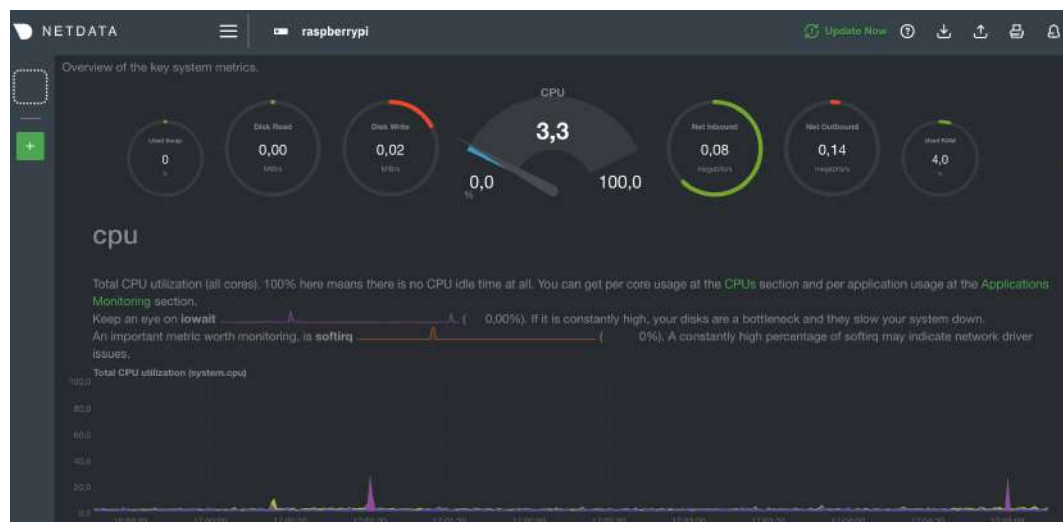


Figure 5.1: Netdata on Raspberry Pi 4 Model B

5.2 STM32MP157A-DK1

The second action performed to test Algorand on low-power devices regards the implementation of the operations described in the previous chapter on STM32MP157A-DK1 board.

5.2.1 Preliminary Operations

As we have already described in *second chapter*, the STM32MPU Embedded Software distribution for STM32 microprocessor platforms supports three software packages. We used the Starter Package that allows to quickly and easily make any STM32 microprocessor development platform up and running, providing the software image for the STM32MPU Embedded Software distribution which includes the binaries for the OpenSTLinux distribution and the STM32CubeProgrammer.

STM32CubeProgrammer, shown in Figure 5.2, is an all-in-one multi-OS software tool for creating partitions into any flash device available on STM32 platforms.

The microSD card that we used as the Flash device is a SanDisk Ultra 64GB HC Class 10 UHS-1.

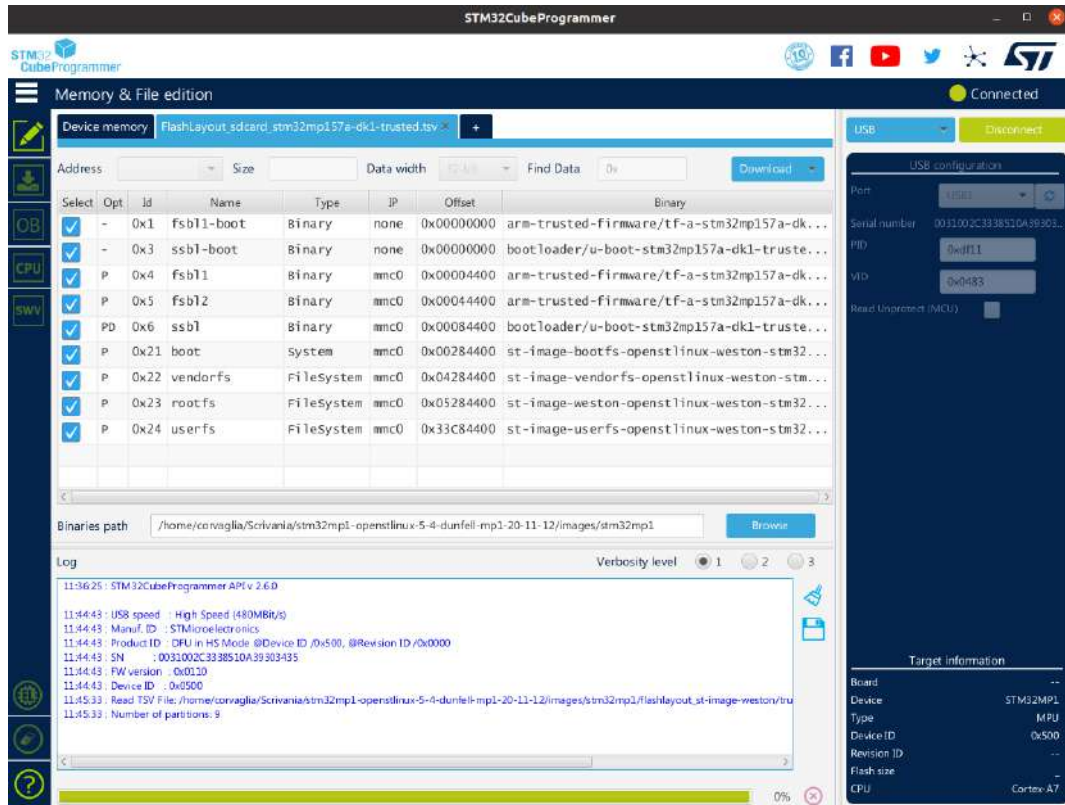


Figure 5.2: Installation of STM32MP1 Starter Package

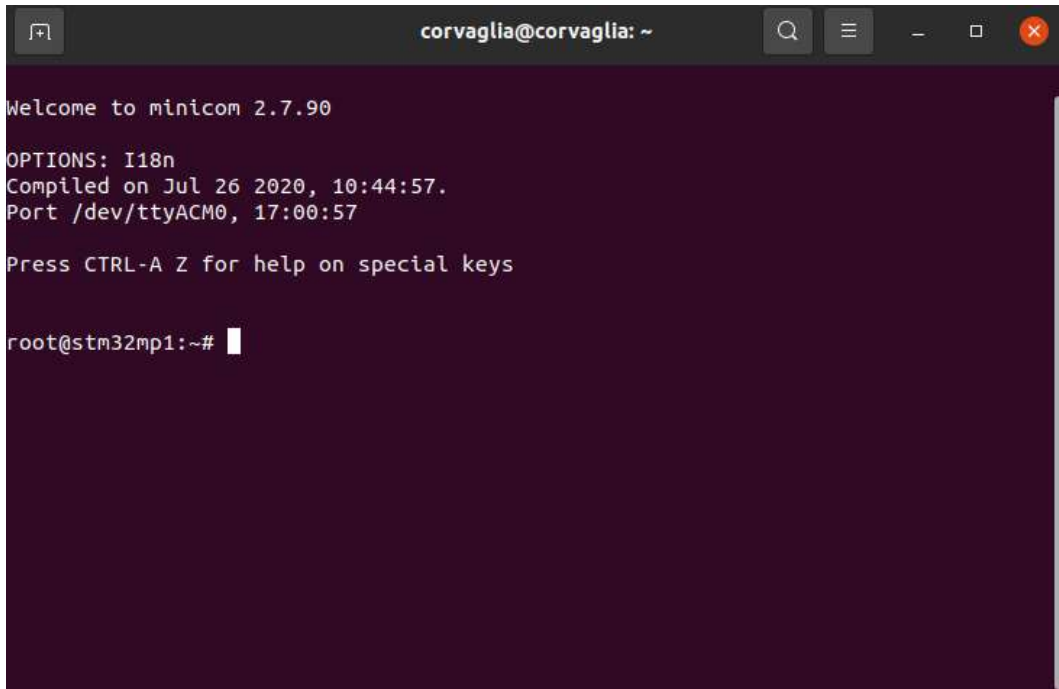
When the boot process is complete, the GTK demo launcher application (written in python3 and GTK) is displayed.

There are two ways to connect and then control the board:

- Remote terminal sessions with *SSH*
- Serial link (UART/USB) with *minicom*

Minicom is a text-based modem control program and terminal emulator for Unix-like operating systems, originally written by Miquel van Smoorenburg and modeled on the popular MS-DOS program Telix.

Figure 5.3 shows how the STM32MP1 board with *minicom* is presented.



```
corvaglia@corvaglia: ~
Welcome to minicom 2.7.90

OPTIONS: I18n
Compiled on Jul 26 2020, 10:44:57.
Port /dev/ttyACM0, 17:00:57

Press CTRL-A Z for help on special keys

root@stm32mp1:~#
```

Figure 5.3: Minicom board

The Algorand Network is composed of two distinct types of nodes: Relay nodes and Non-Relay nodes.

- Relay nodes communicate with other Relay nodes and route blocks to all connected Non-Relay nodes.
- Non-Relay nodes may be connected to several Relay nodes and can also participate in consensus, but they never connect to other Non-Relay nodes.

Nodes can be configured to be archival and indexed: archival when nodes store the entire ledger and, if the indexer is turned on, the search range through the *API REST endpoint* is increased.

Relay nodes are always set to archival mode unlike Non-Relay nodes which, by default, are configured in non-archival mode and store a limited number of blocks (approximately up to the last 1000 blocks) locally. Older blocks are dropped from the local copy of the ledger and this reduces the disk space requirement of the node.

Nodes can be configured with different options that determine some of the capabilities and whether it works as a Relay node or a Non-Relay node. This involves setting parameters in the configuration file (`config.json`) for either the

algod or *kmd* process and is located in the nodes data directory.

Node installation is generally a three to four step process, as shown previously where we installed Non-Relay non-archival Algorand node on Raspberry Pi 4 Model B.

In Figure 5.4 we see where Non-Relay non-archival node has been installed on STM32MP1 board and also here, to use Algorand TestNet network, we created a directory *testnetdata* and copied the *genesis.json* file.

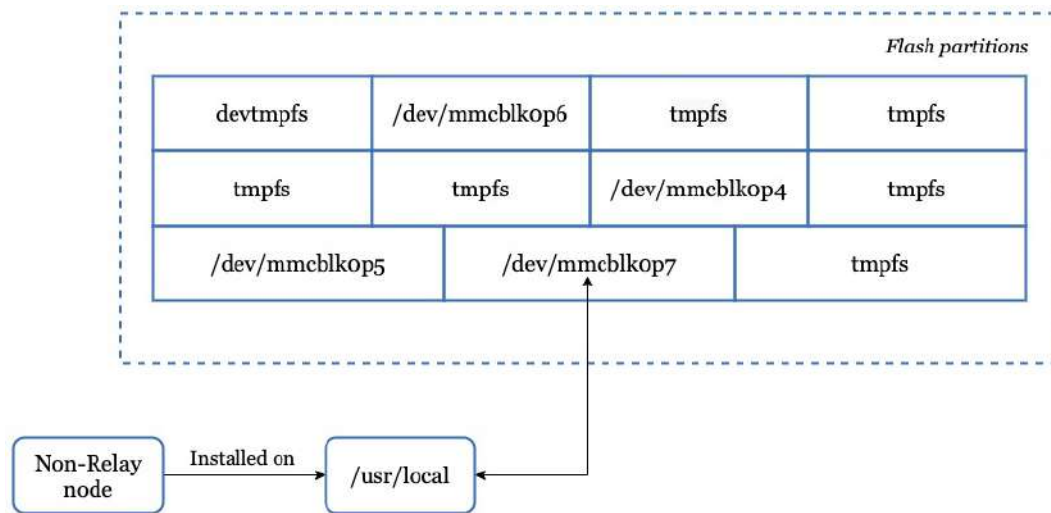


Figure 5.4: Algorand Node on STM32MP157A-DK1

```
root@stm32mp1:/# df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        148M   0  148M   0% /dev
/dev/mmcblk0p6  690M  443M  202M  69% /
tmpfs           214M   64K  214M   1% /dev/shm
tmpfs           214M  8.7M  205M   5% /run
tmpfs           214M    0  214M   0% /sys/fs/cgroup
tmpfs           214M   84M  130M  40% /tmp
/dev/mmcblk0p4   58M   14M   40M  26% /boot
/dev/mmcblk0p5   15M   6.8M   6.7M  51% /vendor
tmpfs           214M  148K  214M   1% /var/volatile
/dev/mmcblk0p7   57G  315M   54G   1% /usr/local
tmpfs           43M    0   43M   0% /run/user/0
```

5.3 Interaction with CLI

When a node is activated, it processes all blocks in the Blockchain, even if it does not store all blocks locally. This operation is performed to verify every block in the Blockchain, validating the chain. The process can be time consuming, but it is essential when running a trusted node.

If a node is stopped it will stop processing blocks and once is restarted, it will start processing blocks where it left off.

```
root@stm32mp1:/usr/local/node# ./goal node start -d
testnetdata/
Algorand node successfully started!
```

```
root@stm32mp1:/usr/local/node# ./goal node status -d
testnetdata/
Last committed block: 12
Time since last block: 0.8s
Sync Time: 6.5s
Last consensus protocol: https://github.com/
algorandfoundation/specs/tree/3
a83c4c743f8b17adfd73944b4319c25722a6782
Next consensus protocol: https://github.com/
algorandfoundation/specs/tree/3
a83c4c743f8b17adfd73944b4319c25722a6782
Round for next consensus protocol: 13
Next consensus protocol supported: true
Last Catchpoint:
Genesis ID: testnet-v1.0
```

There is a feature called *Fast Catchup* that rapidly updates a node using catchpoint snapshots shortening the entire sync (times may vary depending on the number of accounts, number of blocks and the network).

It must be said that *Fast Catchup* only works for non-archival nodes because, if set like this, it does not keep a complete copy of the ledger.

```
root@stm32mp1:/usr/local/node# ./goal node catchup 11320000#5
HSFH424JS07FSQCHDOF4SL7WACEMEZ6JGJ7VKDFVC4SYXA4PQIA -d
testnetdata/
```

```

root@stm32mp1:/usr/local/node# ./goal node status -d
testnetdata/ -w 1000
Last committed block: 52
Sync Time: 898.5s
Catchpoint: 11320000#5
HSFH424JS07FSQCHDOF4SL7WACEMEZ6JGJ7VKDFVC4SYXA4PQIA
Catchpoint total accounts: 266145
Catchpoint accounts processed: 266145
Catchpoint accounts verified: 266145
Catchpoint total blocks: 1000
Catchpoint downloaded blocks: 1000
Genesis ID: testnet-v1.0
Genesis hash: SG01GKSzyE7IEPItTxCByw9x8FmnrCDexi9/c0UJ0iI=

```

When the node is caught up with the rest of the network, the Sync Time is 0.

```

Last committed block: 11208074
Time since last block: 3.5s
Sync Time: 0.0s
Last consensus protocol: https://github.com/
    algorandfoundation/specs/tree/3
    a83c4c743f8b17adfd73944b4319c25722a6782
Next consensus protocol: https://github.com/
    algorandfoundation/specs/tree/3
    a83c4c743f8b17adfd73944b4319c25722a6782
Round for next consensus protocol: 11208075
Next consensus protocol supported: true
Last Catchpoint:
Genesis ID: testnet-v1.0
Genesis hash: SG01GKSzyE7IEPItTxCByw9x8FmnrCDexi9/c0UJ0iI=

```

Wallets and Accounts Wallets, in the context of Algorand developer tools, refer to wallets generated and managed by *kmd* process that stores collections of wallets and allows users to perform operations using the keys stored within these.

Every wallet is associated with a master key represented as a 25-word mnemonic from which all accounts are derived (wallets are stored encrypted on disk).

```
root@stm32mp1:/usr/local/node# ./goal kmd start -d
testnetdata/
Successfully started kmd
```

```
root@stm32mp1:/usr/local/node# ./goal wallet new testwallet -
d testnetdata/
Please choose a password for wallet 'testwallet':
Please confirm the password:
Creating wallet...
Created wallet 'testwallet'
Your new wallet has a backup phrase that can be used for
recovery.
Keeping this backup phrase safe is extremely important.
Would you like to see it now? (Y/n): Y
Your backup phrase is printed below.
Keep this information safe -- never share it with anyone!
One or more non-printable characters were omitted from the
subsequent line:
[steak relief health shine segment bus mountain seed coconut
metal hover scatter save blind man spoil path apology
bachelor lamp pet slender december about noise]
```

```
root@stm32mp1:/usr/local/node# ./goal account new -d
testnetdata/
Please enter the password for wallet 'testwallet':
Created new account with address
XLI3FM3YPIPJM3Q23WIHSDUIDLL5PGKJPJWBRDFJ7YKK0J6BD55IF27C7E
```

```
root@stm32mp1:/usr/local/node# ./goal account new -d
testnetdata/
Please enter the password for wallet 'testwallet':
Created new account with address
ZVWHSTJKQLSNIUR2TKMM2EK25MFD63OVZYDPX0ZWA03UJZNQWHN2WS276E
```

```
root@stm32mp1:/usr/local/node# ./goal account new -d
testnetdata/
Please enter the password for wallet 'testwallet':
Created new account with address 7
ZEEEYMRYL2HIXQBRI6G6TDSJPOAJXL5JONCY7RC5WXVZL7RUDRW4H3WJ4
```

To recover a wallet and any previously generated accounts, it is used the wallet mnemonic. The master derivation key for the wallet will always generate the same addresses in the same order, therefore the process of recovering an account within the wallet looks exactly like generating a new account.

```
root@stm32mp1:/usr/local/node# ./goal wallet new -r
testwallet -d testnetdata/
Please type your recovery mnemonic below, and hit return
when you are done:
steak relief health shine segment bus mountain seed coconut
metal hover scatter save blind man spoil path apology
bachelor lamp pet slender december about noise
Please choose a password for wallet 'testwallet':
Please confirm the password:
Creating wallet...
Created wallet testnetdata
```

```
root@stm32mp1:/usr/local/node# ./goal account new -w
testwallet -d testnetdata/
Please enter the password for wallet 'testwallet':
Created new account with address
XLI3FM3YPIPIJM3Q23WIHSDUIDLL5PGKJPJWBRDFJ7YKK0J6BD55IF27C7E
```

The reasons behind the necessity of using *kmd* are two:

- Public/private key pairs are generated from a single master derivation key and then just remember the single mnemonic that represents the master derivation key (i.e. the wallet passphrase/mnemonic) to regenerate all of the accounts in that wallet.
- There is no way for someone else to determine that two addresses are generated from the same master derivation key and this provides a potential avenue for applications to implement anonymous spending for end users without requiring users to store multiple passphrases.

Using *kmd* requires running a process and storing keys encrypted on disk, but if it is not possible to access to a node or a more lightweight solution is required, there are Standalone accounts that have a low setup cost as you do not need to connect to a separate client that depends on separate hardware.

```

root@stm32mp1:/usr/local/node# ./goal account new
Created new account with address [Address]

root@stm32mp1:/usr/local/node# ./goal account export -a
address<Placeholder>
Exported key for account [Address]: [Passphrase]

```

Another way to create accounts are Multisignature that requires two or more private keys to sign transactions, creating an extra layer of security on an account.

The keys that can sign for the Multisignature account can be stored in separate locations and generated with *kmd*, as Standalone accounts or with a mixture of both.

Multisignature accounts trade off convenience for security because every transaction requires multiple signatures which can be overly complex for a scenario where security or governance is not critical.

```

$ Address1=$(goal account new | awk '{ print $6 }')
$ Address2=$(goal account new | awk '{ print $6 }')
$ Address3=$(goal account new | awk '{ print $6 }')

root@stm32mp1:/usr/local/node# ./goal account multisig new
$Address1 $Address2 $Address3 -T 2
Created new account with address [Multisig_Address]

```

All these steps are performed using the SDKs that also connect to tool *kmd* through a *REST endpoint* and *access token*.

```

1 final String KMD_API_ADDR = "http://192.168.1.72:58816";
2 final String KMD_API_TOKEN = "7
    b2eaf1e05276377fcec58a61bb4f278d8a492c93d0a5a7741c6762499e";

```

Participate in Consensus An account that participates in the Algorand consensus protocol is eligible and available to be selected to propose and vote on new blocks and, to participate, it must generate a valid participation key registering that key online with a special online registration transaction.

A participation key is online if there is a single fully-synchronized node on the

Algorand network that has that key in its ledger directory (the moment a key registration transaction is confirmed by the network it takes 320 rounds for the change to take effect).

It is important to mark an account as offline if it is not participating because, in this way, it is performing bad network behavior and will decrease the honest/dishonest user ratio that underpins the liveness of the agreement protocol, since the network uses the online/offline status of an account to calculate block vote thresholds.

The process of renewing a participation key is simply creating a new participation key and registering it online before the previous key expires.

```
root@stm32mp1:/usr/local/node# ./goal account addpartkey -a
XLI3FM3YPIPIJM3Q23WIHSDUIDLL5PGKJPJWBRDFJ7YKK0J6BD55IF27C7E
--roundFirstValid=11208000 --roundLastValid=14208000 -d
testnetdata
Participation key generation successful
```

```
root@stm32mp1:/usr/local/node# ./goal account partkeyinfo -d
testnetdata
Dumping participation key info from testnetdata...
-----
File:
XLI3FM3YPIPIJM3Q23WIHSDUIDLL5PGKJPJWBRDFJ7YKK0J6BD55IF27C7E
.11208000.14208000.partkey
{
  "acct": "
XLI3FM3YPIPIJM3Q23WIHSDUIDLL5PGKJPJWBRDFJ7YKK0J6BD55IF27C7E",
  "first": 11208000,
  "last": 14208000,
  "sel": "+KYNke48+EdD+OPQXVa0YeB+p0XuSjcyC0lmMFiZaLk=",
  "vote": "Xjxdswv7ud+YkQdVzdT74rDguM7bD13deDhjAFe0L2I=",
  "voteKD": 10000
}
```

```
root@stm32mp1:/usr/local/node# ./goal account
changeonlinestatus --address=
XLI3FM3YPIPIJM3Q23WIHSDUIDLL5PGKJPJWBRDFJ7YKK0J6BD55IF27C7E -
```

```
d testnetdata
```

Private Network Private network is desirable for developers and this allows simulating a multi node network.

These are fully-formed Algorand networks with private, custom genesis ledgers running the current build of Algorand software.

Rather than creating a node instance based on the released *genesis.json*, these networks have their own and need to be manually connected.

Algorand *goal* utility supports commands to create, start, restart, status, stop and delete private networks, as shown in Figure 5.5.

Command	Usage
create	Create private named network
delete	Stop and delete deployed private network
start	Start deployed private network
restart	Restart deployed private network
status	Prints status for all nodes in deployed private network
stop	Stop a deployed private network

Figure 5.5: Commands for private network

Before creating a private network a template needs to be created, which describes how the network and wallets should be configured.

Wallets have three properties: *Name*, *Stake* and *Online*. The wallet stake is specified in percent, and the percent should total 100%. *Online* is set to true or false and specifies whether the account is marked online.

Nodes can be configured with the *Name*, *IsRelay* and *Wallets* properties: *Name* allows to name each of the nodes, *IsRelay* indicates the node is in-

tended to be a Relay (there must be at least one Relay included in any network) and *Wallets* specifies which wallets are assigned to each node.

Each wallet entry has an additional *ParticipationOnly* that indicates that the wallet only has access to participation keys, not rootkeys. This means that the wallet is set up on the node to participate in consensus, but no signing of transactions can occur on that node with the specific wallet.

```
root@stm32mp1:/usr/local/node# ./goal network create -r
privatestm32 -n private -t privatestm32.json
Created new rootkey: /usr/local/node/privatestm32/WalletTwo.
rootkey
Created new rootkey: /usr/local/node/privatestm32/WalletOne.
rootkey
Created new rootkey: /usr/local/node/privatestm32/WalletThree
.rootkey
Created new partkey: /usr/local/node/privatestm32/WalletTwo
.0.3000000.partkey
Created new partkey: /usr/local/node/privatestm32/WalletOne
.0.3000000.partkey
https://github.com/algorandfoundation/specs/tree/3
a83c4c743f8b17adfd73944b4319c25722a6782 100000
Network private created under /usr/local/node/privatestm32
```

```
root@stm32mp1:/usr/local/node# ./goal network start -r
privatestm32
Network Started under privatestm32
```

```
root@stm32mp1:/usr/local/node# ./goal network status -r
privatestm32

[First]
Last committed block: 1024
Time since last block: 3.9s
Sync Time: 0.0s
Last consensus protocol: https://github.com/
algorandfoundation/specs/tree/3
a83c4c743f8b17adfd73944b4319c25722a6782
Next consensus protocol: https://github.com/
```

```

algorandfoundation/specs/tree/3
a83c4c743f8b17adfd73944b4319c25722a6782
Round for next consensus protocol: 1025
Next consensus protocol supported: true

[Node]
Last committed block: 1024
Time since last block: 3.9s
Sync Time: 0.0s
Last consensus protocol: https://github.com/
algorandfoundation/specs/tree/3
a83c4c743f8b17adfd73944b4319c25722a6782
Next consensus protocol: https://github.com/
algorandfoundation/specs/tree/3
a83c4c743f8b17adfd73944b4319c25722a6782
Round for next consensus protocol: 1025
Next consensus protocol supported: true

```

5.4 Interaction with Node

The connection to Algorand takes place through *algod* client that requires a valid *algod REST endpoint IP address* and *algod token* from an Algorand node that is connected to the network you plan to interact with.

As shown previously, we have already a directory *testnetdata* to use the Algorand TestNet network and now we set *REST endpoint IP address* and *token* of node already synchronized on the STM32 board.

To create a transaction, Atomic Transfer and ASA we use AlgoSDK, a Java library for communicating and interacting with the Algorand network.

```

1  // Create an algod client
2  import com.algorand.algosdk.v2.client.common.AlgodClient;
3
4  final String ALGOD_API_ADDR = "http://192.168.1.72";
5  final Integer ALGOD_PORT = 6001;
6  final String ALGOD_API_TOKEN = "62569290418
    bdd347a5df5fe9bb191a79aa89124fdda23541a4c1dc36bf3d335";

```

```

7
8  AlgodClient client = new AlgodClient(ALGOD_API_ADDR,
    ALGOD_PORT, ALGOD_API_TOKEN);

```

Transaction Transactions require a certain minimum set of parameters to be valid and mandatory fields include the round validity range, the fee and the genesis hash for the network which the transaction is valid.

To sign the transaction we need the private key and this creates a new transaction object signed in the SDKs, sending it to the network with the *algod* client.

```

1  // Construct the transaction
2  final String RECEIVER = "7
    ZEEEYMYRL2HIXQBRIBG6TDSJPOAJXL5JONCY7RC5WXVZL7RUDRW4H3WJ4";
3  String note = "First STM32MP157A-DK1 Transaction";
4  TransactionParametersResponse params = client.
    TransactionParams().execute().body();
5  Transaction txn = Transaction.PaymentTransactionBuilder()
6  .sender(myAddress)
7  .note(note.getBytes())
8  .amount(100000000)
9  .receiver(new Address(RECEIVER))
10 .suggestedParams(params)
11 .build();

```

Atomic Transfers Atomic Transfers are implemented as irreducible batch operations where a group of transactions are submitted as a unit and all transactions in the batch either pass or fail. Just like any other transaction can contain ALGO or Algorand Standard Assets that may also be governed from Algorand Smart Contracts.

Combining transactions just means concatenating them into a single file or ordering them in an array so that a group ID can then be assigned.

```

1  // Get node suggested parameters
2  String note1 = "First Atomic Transfer with STM32MP157A-DK1";
3  String note2 = "First Atomic Transfer with STM32MP157A-DK1";

```

```

4 TransactionParametersResponse params = client.
    TransactionParams().execute().body();
5
6 // Create the first transaction
7 Transaction tx1 = Transaction.PaymentTransactionBuilder()
8     .sender(accountA.getAddress())
9     .note(note1.getBytes())
10    .amount(300000000)
11    .receiver(accountC.getAddress())
12    .suggestedParams(params)
13    .build();
14
15 // Create the second transaction
16 Transaction tx2 = Transaction.PaymentTransactionBuilder()
17     .sender(accountB.getAddress())
18     .note(note2.getBytes())
19     .amount(300000000)
20     .receiver(accountA.getAddress())
21     .suggestedParams(params)
22     .build();
23
24 // Group transactions and assign ids
25 Digest gid = TxGroup.computeGroupID(tx1, tx2);
26 tx1.assignGroupID(gid);
27 tx2.assignGroupID(gid);

```

Creation of ASA Algorand Standard Assets are highly customizable with parameters that allow to define total issuance of an asset, name of asset, units of an asset, as well as access controls privileges over an asset.

There are four parameters that correspond to addresses that can authorize specific functionality for an asset:

- Manager Address: the manager account is the only account that can authorize transactions to re-configure or destroy an asset.
- Reserve Address: specifying a reserve account signifies that non-minted assets will reside in that account instead of the default creator account (assets

transferred from this account are “minted” units of the asset).

- Freeze Address: the freeze account is allowed to freeze or unfreeze the asset holdings for a specific account.

- Clawback Address: the clawback address represents an account that is allowed to transfer assets from and to any asset holder.

```
1 // Create the Asset
2 TransactionParametersResponse params = client.
   TransactionParams().execute().body();
3
4 Long assetTotal = 10000000000000000L;
5 String unitName = "STM";
6 String assetName = "STMicroelectronics";
7 String url = "https://www.st.com/";
8 Address manager = account1.getAddress();
9 Address reserve = account1.getAddress();
10 Address freeze = account1.getAddress();
11 Address clawback = account1.getAddress();
12 int decimals = 6;
13 Transaction tx = Transaction.AssetCreateTransactionBuilder().
   sender(account1.getAddress()).assetTotal(assetTotal)
14 .assetDecimals(decimals).assetUnitName(unitName).assetName(
   assetName).url(url)
15 .manager(manager).reserve(reserve).freeze(freeze)
16 .defaultFrozen(false).clawback(clawback).suggestedParams(
   params).build();
```

Chapter 6

Results

In this chapter we present the results of the study with respect to the aim of the study, which was to install a node, write and read on Algorand Blockchain using some of its features and, finally, provide a performance and monitoring analysis on devices used and described in previous chapters.

6.1 Block Explorer

To read on the Blockchains there are the Block Explorers which are tools used to view all transactions online, specifically all current and past transactions. In other words, a Block Explorer view all transactions that have taken place on the Blockchain, the current network hash rate, the activity on addresses and more.

In Algorand there is a Block Explorer called AlgoExplorer [15] that allows to explore and search the Algorand Blockchain for transactions, addresses, smart contracts and assets, also providing historical data, statistics, staking rewards program and other activities.

Below we have the results already implemented in *five chapter*, obtaining in output a transaction ID (TXID) or transaction hash, a unique string of characters that is given to every transaction that is verified and added to the Blockchain.

Figures 6.1, 6.2, 6.3 and 6.4 shows how transactions are read on the Algorand

Block Explorer already described above.

We performed all the following operations using the node installed on STM32MP157A-DK1 board and we used a home connection where the nominal bandwidth is 20 Mbit/s in Download and 1 Mbit/s in Upload.

STM32MP157A-DK1 Transaction On Algorand, blocks are confirmed in seconds and transactions are final as soon as they are incorporated into a block. The executed transaction was confirmed and written to the Blockchain in less than 5 seconds.

```
Signed transaction with txid:
TS46PNQUS45DXBWHTXBYRM63GIGXY4FLIOKLZENGWJGFCOM6DU4Q
Successfully sent tx with ID:
TS46PNQUS45DXBWHTXBYRM63GIGXY4FLIOKLZENGWJGFCOM6DU4Q
Transaction
TS46PNQUS45DXBWHTXBYRM63GIGXY4FLIOKLZENGWJGFCOM6DU4Q
confirmed in round 11800831
```

The screenshot shows the Algo Explorer interface. At the top, there's a search bar and navigation links. The main section is titled "Transaction Overview" and displays the transaction ID, timestamp, block number, type, and status. Below this, the "Transaction Details" section shows the amount, sender, and receiver addresses. At the bottom, there's a "Note" section with tabs for "Message Pack", "Base 64", and "Editor View". The "Editor View" tab is active, showing a hex editor with three rows of data.

Offset	Hex	ASCII
00000000	46 69 72 73 74 20 53 54 4D 33 32 4D 50 31 35 37	First STM32MP157
00000010	41 2D 44 4B 31 20 54 72 61 6E 73 61 63 74 69 6F	A-DK1 Transactio
00000020	6E	n

Figure 6.1: STM32MP157A-DK1 Transaction

STM32MP157A-DK1 Atomic Transfer The following Atomic Transfer was confirmed in less than 5 seconds. The core functions are:

- Create the transactions: this is like creating any kind of transaction in Algorand
- Group the transactions: this involves computing a groupID and assigning that id to each transaction
- Sign the grouped transactions: sign the grouped transactions with their respective private keys
- Send the transactions to the network: combine the transactions and send to the network

```
Successfully sent tx with ID:
D55DP6HCNT5MFT652RWQYTEV7WRW4X6CNNTYKPRGRICFPNXZLF3Q
Transaction
D55DP6HCNT5MFT652RWQYTEV7WRW4X6CNNTYKPRGRICFPNXZLF3Q
confirmed in round 11839043
Group ID: p6KbKj7ZhrVDtbQYAygwlgsyd/DkISebffyyze7TEVpM=
```

The screenshot displays the Algo Explorer interface for an Atomic Transfer transaction. The top navigation bar includes the Algo Explorer logo, a search bar, and links to TESTNET and Wallets. The main content area is titled "Transaction Overview" and shows the Transaction ID, Block (11839043), Type (Transfer), and Status (Completed). Below this, the "Transaction Details" section provides the Group ID, Amount (30 ALG), Sender, and Receiver. At the bottom, a "Message Pack" section shows the transaction data in hex and ASCII format.

Offset	Hex	ASCII
00000000	46 69 72 73 74 20 41 74 6F 6D 69 63 20 54 72 61	First Atomic Transfer with STM32
00000010	6E 79 66 65 72 20 77 69 74 68 20 53 54 4D 33 32	MP157A-DK1
00000020	4D 5D 31 35 37 41 2D 44 4B 31	

Figure 6.2: STM32MP157A-DK1 Atomic Transfer

STMicroelectronics ASA The type of Algorand Standard Asset (ASA) created depend on the parameters that are passed during asset creation and sometimes during asset re-configuration.

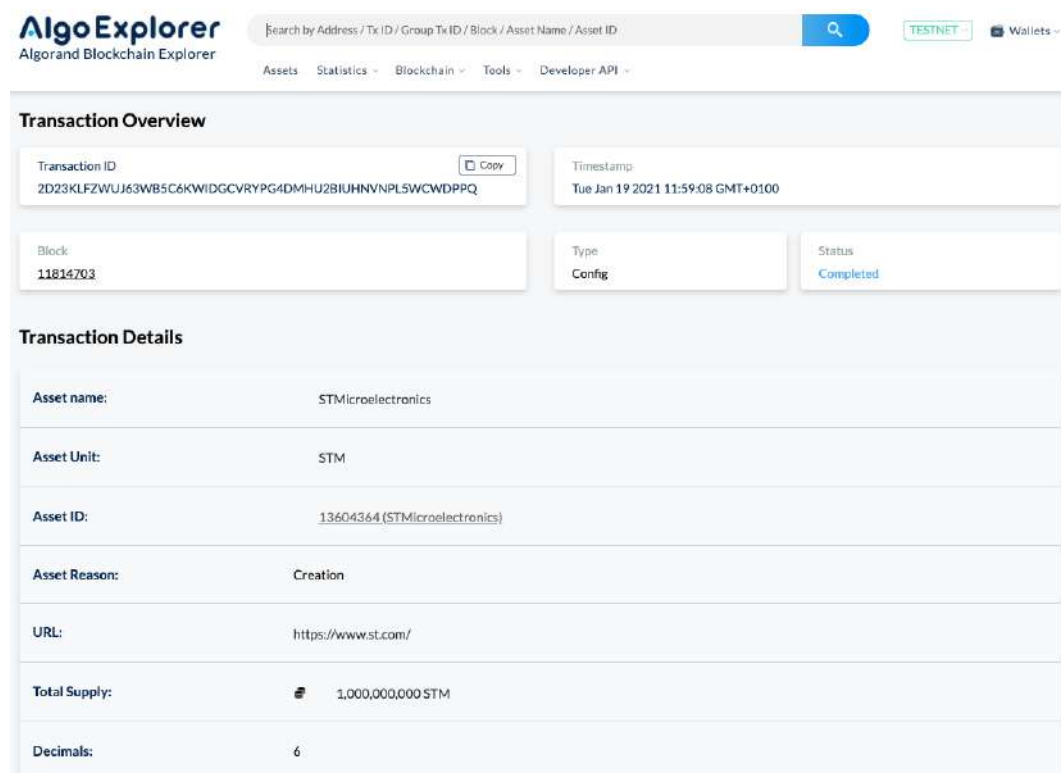
In this case we created an Algorand Standard Asset called STMicroelectronics with unit name STM and 1 billion total number of base units with decimal set to 6.

Transaction ID:
2D23KLFZWUJ63WB5C6KWIDGCVRYPG4DMHU2BIUHNVNPL5WCWDPPQ

Transaction
2D23KLFZWUJ63WB5C6KWIDGCVRYPG4DMHU2BIUHNVNPL5WCWDPPQ

confirmed in round 11814703

AssetID = 13604364



The screenshot shows the AlgoExplorer interface. At the top is a search bar and navigation links. The main content area is titled 'Transaction Overview' and displays the transaction ID, timestamp, block number, type, and status. Below this is the 'Transaction Details' section, which provides a comprehensive list of asset parameters.

Transaction Overview	
Transaction ID	2D23KLFZWUJ63WB5C6KWIDGCVRYPG4DMHU2BIUHNVNPL5WCWDPPQ
Timestamp	Tue Jan 19 2021 11:59:08 GMT+0100
Block	11814703
Type	Config
Status	Completed

Transaction Details	
Asset name:	STMicroelectronics
Asset Unit:	STM
Asset ID:	13604364 (STMicroelectronics)
Asset Reason:	Creation
URL:	https://www.st.com/
Total Supply:	1,000,000,000 STM
Decimals:	6

Figure 6.3: STMicroelectronics ASA

Key Registration Transaction What distinguishes this as a key registration transaction is “type”: “keyreg” with the existence of participation key-related fields, namely “votekey” that is the root participation public key, the VRF public key “selkey”, the dilution for 2-level participation key “votekd”

and “votefst”/“votelst”, respectively first and last round that the participation key is valid.

The values for these fields are obtained by dumping the participation key info on the node where the participation key lives.

```
Transaction id for status change transaction:
RKS5CC0WIC5XA6ZQI40VOX63QUCIDTV7J76J2ALTLQI6QAYRB33A
Transaction
RKS5CC0WIC5XA6ZQI40VOX63QUCIDTV7J76J2ALTLQI6QAYRB33A still
pending as of round 11208301
Transaction
RKS5CC0WIC5XA6ZQI40VOX63QUCIDTV7J76J2ALTLQI6QAYRB33A
committed in round 11208303
```

The screenshot shows the Algo Explorer interface. At the top, there's a search bar and navigation links. The main section is titled "Transaction Overview" and displays the Transaction ID: RKS5CC0WIC5XA6ZQI40VOX63QUCIDTV7J76J2ALTLQI6QAYRB33A. Below this, it shows the Block: 11208303, Type: Keyreg, and Status: Completed. The "Transaction Details" section lists the following information:

Sender:	XU3FM3YPIPIJ3Q23WIHSUIDLLSPGKJPJWRBDEJZYKKOJ6BD55IF27C7E
Vote Key:	Xjxdsww7ud+YkQdVzdT74rDguM7bD13deDhjAFeOL2l=
Selection Key:	+KYNke48+EdD+OPQXVa0YeB+pOXuSjcyC0ImMFIZaLk=
Vote First:	11208000
Vote Last:	14208000
Vote Key Dilution:	10000

Figure 6.4: Key Registration Transaction

6.2 Performance analysis

To have greater clarity which the devices described in *second chapter* work using the Blockchain Algorand, we use first of all Netdata tool for visualize total CPU utilization (all cores) and Bandwidth from Ethernet interface of

STM32MP157A-DK1 board in the various processes of synchronization of the node with *Fast Catchup* feature that rapidly updates it using catchpoint snapshots, shortening the entire node sync. The same method was used with the Raspberry Pi 4 Model B board.

An important thing is that sync times may vary depending on the number of accounts, number of blocks, technical specifications of devices and the network, but we can actually understand how the devices works in the various processes until complete synchronization, as shown in Figures 6.5, 6.6, 6.7, 6.8 and 6.9. In the first phase of sync, we process the total number of accounts included in the current catchpoint.

```
root@stm32mp1:/usr/local/node# ./goal node start -d testnetdata/
Algorand node successfully started!
root@stm32mp1:/usr/local/node# ./goal node catchup 13320000#5F06ATLXZ6A0ISA5U5S6
AI4RGED3R3M3LY54KU3JMKEICUF732XA -d testnetdata/
root@stm32mp1:/usr/local/node# ./goal node status -d testnetdata/ -w 1000
Last committed block: 82
Sync Time: 584.8s
Catchpoint: 13320000#5F06ATLXZ6A0ISA5U5S6AI4RGED3R3M3LY54KU3JMKEICUF732XA
Catchpoint total accounts: 389776
Catchpoint accounts processed: 389776
Catchpoint accounts verified: 0
Genesis ID: testnet-v1.0
Genesis hash: SG01GKSzyE7IEPItTxCByw9x8FmnrCDexi9/c0UJ0iI=
█
```



Figure 6.5: Catchpoint accounts processed on STM32

There are four components to consider for analysis: *system*, *user*, *iowait* and *softirq*.

- *system*: this component is the proportion of time the CPUs spent inside the Linux kernel for operations like context switching, memory allocation and queue handling.
- *user*: this component is the time spent in the *user* space (high value of *user* time indicate a CPU bound workload).
- *iowait*: this component is the time the CPU spent waiting for disk IO requests to complete.
- *softirq*: this component is the portion of time the CPU spent servicing software interrupts generated by the device drivers.

It should also be considered that if *iowait* is constantly high, disks are a bottleneck and they slow system down, furthermore, a constantly high percentage of *softirq* may indicate network driver issues (the network devices are generally the main source of high *softirq* values).

In the second phase we verify the number of accounts from the current catchpoint as part of the catchup.

```

root@stm32mp1:/usr/local/node# ./goal node start -d testnetdata/
Algorand node successfully started!
root@stm32mp1:/usr/local/node# ./goal node catchup 13320000#5F06ATLXZ6A0ISA5U5S6
AI4RGED3R3M3LY54KU3JMKEICUF732XA -d testnetdata/
root@stm32mp1:/usr/local/node# ./goal node status -d testnetdata/ -w 1000
Last committed block: 82
Sync Time: 697.6s
Catchpoint: 13320000#5F06ATLXZ6A0ISA5U5S6AI4RGED3R3M3LY54KU3JMKEICUF732XA
Catchpoint total accounts: 389776
Catchpoint accounts processed: 389776
Catchpoint accounts verified: 389776
Genesis ID: testnet-v1.0
Genesis hash: SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=

```



Figure 6.6: Catchpoint accounts verified on STM32

When we introduced technically the various types of Algorand nodes in *five chapter*, we asserted that by default Non-Relay nodes only store a limited number of blocks locally.

In this phase we download the total number of blocks that are required to complete the current catchpoint catchup.

```
root@stm32mp1:/usr/local/node# ./goal node start -d testnetdata/
Algorand node successfully started!
root@stm32mp1:/usr/local/node# ./goal node catchup 13320000#5F06ATLXZ6A0ISA5U5S6
AI4RGED3R3M3LY54KU3JMKEICUF732XA -d testnetdata/
root@stm32mp1:/usr/local/node# ./goal node status -d testnetdata/ -w 1000
Last committed block: 82
Sync Time: 937.4s
Catchpoint: 13320000#5F06ATLXZ6A0ISA5U5S6AI4RGED3R3M3LY54KU3JMKEICUF732XA
Catchpoint total accounts: 389776
Catchpoint accounts processed: 389776
Catchpoint accounts verified: 389776
Catchpoint total blocks: 1000
Catchpoint downloaded blocks: 1000
Genesis ID: testnet-v1.0
Genesis hash: SG01GKSzyE7IEPItTxCByw9x8FmnrCDexi9/c0UJ0iI=
█
```



Figure 6.7: Catchpoint downloaded blocks on STM32

The last phase consist to download the remaining blocks from that point to

the current block.

```
Last committed block: 13321115
Time since last block: 0.1s
Last committed block: 13321118
Time since last block: 0.3s
Last committed block: 13321122
Time since last block: 0.2s
Last committed block: 13321124
Time since last block: 0.2s
Last committed block: 13321127
Time since last block: 0.0s
Last committed block: 13321130
Time since last block: 0.1s
Last committed block: 13321134
Time since last block: 0.1s
Sync Time: 382.5s
Last consensus protocol: https://github.com/algorandfoundation/specs/tree/3a83c4c743f8b17adfd73944b4319c25722a6782
Next consensus protocol: https://github.com/algorandfoundation/specs/tree/3a83c4c743f8b17adfd73944b4319c25722a6782
Round for next consensus protocol: 13321135
Next consensus protocol supported: true
Last Catchpoint:
Genesis ID: testnet-v1.0
Genesis hash: SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/c0UJ0iI=
```



Figure 6.8: Sync to the current block on STM32

When the node is caught up with the rest of the network, the “Sync Time” is 0.


```

Last committed block: 13329559
Time since last block: 3.5s
Last committed block: 13329559
Time since last block: 4.5s
Last committed block: 13329560
Time since last block: 0.9s
Last committed block: 13329560
Time since last block: 1.9s
Last committed block: 13329560
Time since last block: 2.9s
Last committed block: 13329560
Time since last block: 3.9s
Last committed block: 13329561
Time since last block: 0.4s
Sync Time: 0.0s
Last consensus protocol: https://github.com/algorandfoundation/specs/tree/3a83c4c743f8b17adfd73944b4319c25722a6782
Next consensus protocol: https://github.com/algorandfoundation/specs/tree/3a83c4c743f8b17adfd73944b4319c25722a6782
Round for next consensus protocol: 13329562
Next consensus protocol supported: true
Last Catchpoint:
Genesis ID: testnet-v1.0
Genesis hash: SG01GKSzyE7IEPItTxCBYw9x8FmnrCDexi9/cOUJ0iI=

```



Figure 6.9: Synchronized node on STM32

Taking a general look in terms of performance analysis, the greatest CPU bound workload happens when it downloads the number of blocks until the node is fully synchronized, which from then is less stressful.

We performed all previous phases using the other low-power device, the Rasp-

berry Pi 4 Model B, and also in this case we reach almost the same result, as shown in Figures 6.10 and 6.11.



Figure 6.10: Sync to the current block on Raspberry Pi 4 Model B



Figure 6.11: Synchronized node on Raspberry Pi 4 Model B

Chapter 7

Conclusion and future works

This thesis has been mainly focused on the use of the Algorand Blockchain on low-power devices present on the market, in particular, a discovery kit that is part of STMicroelectronics STM32 family and a Raspberry Pi 4 Model B.

After a comprehensive analysis of the Algorand Blockchain and an analysis of the State of the Art, the thesis presents the design of testbeds for demonstrating the applicability of the technology to low-powered devices. The experimental results show that, as well as the Raspberry Pi 4 Model, already tested in other works, the STMicroelectronics device is also suitable for joining the Algorand network. It is specifically capable of publishing transactions to the Algorand Blockchain in a few seconds and running some of its functions.

All the performed operations have been designed to let the node installed on the board work perfectly with this innovative Blockchain and guarantee decentralization, scalability and security, very important aspects in the IoT ecosystem.

Although all the performed operations successfully demonstrate the applicability of Algorand to boards like the STMicroelectronics STM32MP157A-DK1, as reported on the Algorand official website, some features are still under development, consequently, future works could regard further experiments to deeper analyze particular mechanisms, new proposals to try different methods.

In addition, starting from the work performed within this thesis, some additional future directions could be evaluated for future works: comparison

among the exploitation of various Blockchains on IoT low-power devices, development of a basic C library for Algorand to be used on constrained low-cost devices based on microcontrollers, like STMicroelectronics' STM32 (Cortex-M Architecture), implementation of interesting use cases that could exploit smart contracts for their purposes, and a further performance analysis on the Algorand Mainnet network instead of the Testnet used within this thesis.

List of Figures

2.1	Centralized Database	6
2.2	Types of DLT	7
2.3	DLT evolution	8
2.4	Difference between Public, Private and Consortium Blockchain .	10
2.5	Blockchain Trilemma	12
2.6	Proof-of-Stake	16
2.7	Review of types of Consensus	18
2.8	Participation Keys	24
2.9	Public/Private Key Generation	24
2.10	Public Key to Algorand Address	25
2.11	Base64 Private Key	25
2.12	Private Key Mnemonic	26
2.13	Phases of the Consensus	27
2.14	Block Proposal	28
2.15	Soft Vote - Part One	29
2.16	Soft Vote - Part Two	30
2.17	Certify Vote	31
2.18	Carpenter's output	32
2.19	Algorand Networks	34
2.20	Side-by-Side Network Comparison	34
2.21	Number of IoT connected devices worldwide 2019-2030	35
2.22	STM32 Arm Cortex MPUs	37
2.23	STM32MP1 Series	38
2.24	STM32MP157	38

2.25	STM32MP157A-DK1 top view	39
2.26	STM32MP157A-DK1 bottom view	39
2.27	STM32MP1Starter	40
2.28	Raspberry Pi 4 Model B	41
2.29	Netdata Interface	42
3.1	Comparison between IoT and Blockchain	44
3.2	Proposed Blockchain based solution architecture	46
3.3	Blockchain node specifications for this implementation	47
3.4	Deployment of this system	47
3.5	STM32F746ZG Nucleo	48
3.6	Results obtained using this platform	49
3.7	ESP32 DevKit	49
3.8	Proposed general block diagram	50
3.9	PlanetWatch ecosystem	51
4.1	Algorand Dev Stack	53
4.2	SDK Functionality	54
4.3	TEAL Architecture Overview	55
4.4	Example Asset Types	56
4.5	Atomic Transfer Flow	56
4.6	Rekeying	57
4.7	Setup topology	58
5.1	Netdata on Raspberry Pi 4 Model B	62
5.2	Installation of STM32MP1 Starter Package	64
5.3	Minicom board	65
5.4	Algorand Node on STM32MP157A-DK1	66
5.5	Commands for private network	73
6.1	STM32MP157A-DK1 Transaction	80
6.2	STM32MP157A-DK1 Atomic Transfer	81
6.3	STMicroelectronics ASA	82

6.4	Key Registration Transaction	83
6.5	Catchpoint accounts processed on STM32	85
6.6	Catchpoint accounts verified on STM32	86
6.7	Catchpoint downloaded blocks on STM32	87
6.8	Sync to the current block on STM32	88
6.9	Synchronized node on STM32	89
6.10	Sync to the current block on Raspberry Pi 4 Model B	90
6.11	Synchronized node on Raspberry Pi 4 Model B	90

Bibliography

- [1] Algorand. <https://www.algorand.com>.
- [2] Banco Bilbao Vizcaya Argentaria. What is the difference between dlt and blockchain. *Communications*. <https://www.bbva.com/en/difference-dlt-blockchain>, 19, 2018.
- [3] Hany F Atlam and Gary B Wills. Technical aspects of blockchain and iot. In *Advances in Computers*, volume 115, pages 1–39. Elsevier, 2019.
- [4] Oliver Belin. The difference between blockchain and distributed ledger technology. *TRADEIX*. <https://tradeix.com/distributed-ledger-technology> (accessed June 26, 2019), 2008.
- [5] Marianna Belotti, Nikola Božić, Guy Pujolle, and Stefano Secci. A vademecum on blockchain technologies: When, which, and how. *IEEE Communications Surveys & Tutorials*, 21(4):3796–3838, 2019.
- [6] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012.
- [7] Sharon Shea Brien Posey. Iot devices (internet of things devices). <https://internetofthingsagenda.techtarget.com/definition/IoT-device>, 2021.
- [8] Certik. The blockchain trilemma: Decentralized, scalable, and secure? <https://medium.com/certik/the-blockchain-trilemma-decentralized-scalable-and-secure-e9d8c41a87b3>, 2019.

- [9] Jing Chen, Sergey Gorbunov, Silvio Micali, and Georgios Vlachos. Algorand agreement: Super fast and partition resilient byzantine agreement. *IACR Cryptol. ePrint Arch.*, 2018:377, 2018.
- [10] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.
- [11] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.
- [12] Michael R. Cryptomaniaks. Distributed ledger technology for dummies [2021]. <https://cryptomaniaks.com/guides/distributed-ledger-technology-for-dummies>, 2021.
- [13] RA Davenport. Distributed database technology—a survey. *Computer Networks (1976)*, 2(3):155–167, 1978.
- [14] Algorand Developer. <https://developer.algorand.org>.
- [15] Algo Explorer. <https://algoexplorer.io>.
- [16] Algorand Foundation. <https://algorand.foundation>.
- [17] Raspberry Pi Foundation. Raspberry pi 4 computer model b product brief. <https://datasheets.raspberrypi.org>.
- [18] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.
- [19] OF ITU. Series y: Global information infrastructure, internet protocol aspects and next-generation networks next generation networks—frameworks and functional architecture models.
- [20] Simon Josefsson et al. The base16, base32, and base64 data encodings. Technical report, RFC 4648, October, 2006.

- [21] Sandeep B Kadam and Shajimon K John. Blockchain integration with low-power internet of things devices. In *Handbook of Research on Blockchain Technology*, pages 183–211. Elsevier, 2020.
- [22] Minhaj Ahmad Khan and Khaled Salah. Iot security: Review, blockchain solutions, and open challenges. *Future Generation Computer Systems*, 82:395–411, 2018.
- [23] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19:1, 2012.
- [24] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.
- [25] Cristian Lepore, Michela Ceria, Andrea Visconti, Udai Pratap Rao, Kaushal Arvindbhai Shah, and Luca Zanolini. A survey on blockchain consensus with a performance comparison of pow, pos and pure pos. *Mathematics*, 8(10):1782, 2020.
- [26] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.
- [27] Sudip Misra, Anandarup Mukherjee, Arijit Roy, Nishant Saurabh, Yogachandran Rahulamathavan, and Muttukrishnan Rajarajan. Blockchain at the edge: Performance of resource-constrained iot networks. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):174–183, 2020.
- [28] Bhabendu Kumar Mohanta, Debasish Jena, Somula Ramasubbareddy, Mahmoud Daneshmand, and Amir H Gandomi. Addressing security and privacy issues of iot using blockchain technology. *IEEE Internet of Things Journal*, 8(2):881–888, 2020.
- [29] Satoshi Nakamoto and A Bitcoin. A peer-to-peer electronic cash system. *Bitcoin.–URL: <https://bitcoin.org/bitcoin.pdf>*, 4, 2008.

- [30] Netdata. <https://www.netdata.cloud>.
- [31] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. Mnemonic code for generating deterministic keys. *Online at <https://github.com/bitcoin/bips/blob/master/bip-0039>*. mediawiki, 2013.
- [32] PlanetWatch. Whitepaper. <https://planetwatch.io/white-paper/index-h5.html>, 2021.
- [33] RD Statista. Number of internet of things (iot) connected devices worldwide from 2019 to 2030. *Statista Research Department*. <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide>, 2021.
- [34] STMicroelectronics. <https://www.st.com>.
- [35] STMicroelectronics. Discovery kit with stm32mp157a mpu. <https://www.st.com/en/evaluation-tools/stm32mp157a-dk1.html>.
- [36] Diego Stucchi, Ruggero Susella, Pasqualina Fragneto, and Beatrice Rossi. Secure and effective implementation of an iota light node using stm32. In *Proceedings of the 2nd Workshop on Blockchain-enabled Networked Sensor*, pages 28–29, 2019.
- [37] Nuttakit Vatcharatiansakul and Panwit Tuwanut. A performance evaluation for internet of things based on blockchain technology. In *2019 5th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, pages 1–4. IEEE, 2019.
- [38] Ziyang Wang, Xinghua Dong, Yi Li, Li Fang, and Ping Chen. Iot security model and performance evaluation: A blockchain approach. In *2018 International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, pages 260–264. IEEE, 2018.